SpeedFormula Dokumentáció

[Fejlesztői dokumentáció 2](#_Toc189633208)

[Bevezetés 2](#_Toc189633209)

[Rendszerkövetelmény és eszközök 2](#_Toc189633210)

[Telepítési útmutató 3](#_Toc189633211)

[Funkcionalitások 3](#_Toc189633212)

[Architektúra és technológiai részletek 4](#_Toc189633213)

[Tesztelés 4](#_Toc189633214)

[Karbantartás 4](#_Toc189633215)

[Kliens-oldali kód 5](#_Toc189633216)

[Auth.service.ts: 5](#_Toc189633217)

[App.component.ts: 11](#_Toc189633218)

[regisztracio.component.ts 16](#_Toc189633219)

[Szerver-oldali kód 18](#_Toc189633220)

[Hírek lekérése – HírekGetController 18](#_Toc189633221)

[F1-es versenyek lekérése – SessionsGetController 19](#_Toc189633222)

[Csapatok lekérése – CsapatokGetController 19](#_Toc189633223)

[Versenyzők lekérése – VersenyzőkGetController 20](#_Toc189633224)

[Üzenet küldése – ChatPost 20](#_Toc189633225)

[Összes chatüzenet lekérése – ChatGet 21](#_Toc189633226)

[Az összes üzenet törlése – ChatDeleteAll 23](#_Toc189633227)

[Regisztráció (RegisztracioPostController) 24](#_Toc189633228)

[Bejelentkezés (LoginPostController) 25](#_Toc189633229)

[Kijelentkezés (LogoutPostController) 26](#_Toc189633230)

[Profillekérés (ProfilGetController) 27](#_Toc189633231)

[Profil módosítása (ProfilePatchController) 27](#_Toc189633232)

[Profil törlése (ProfilDeleteController) 27](#_Toc189633233)

[Felhasználói dokumentáció 27](#_Toc189633234)

[Bevezetés 27](#_Toc189633235)

[Rendszerkövetelmény 28](#_Toc189633236)

*A megtekintendő tartalomhoz mutasson a kurzorral a tartalomjegyzék adott részére, majd használja a ctrl + bal klikket.*

# Fejlesztői dokumentáció

## Bevezetés

Ez a dokumentáció a SpeedFormula weboldal fejlesztéséhez szükséges részleteket tartalmazza. A weboldal egy modern, dinamikus alkalmazás, amely a felhasználók számára élő közvetítéseket, interaktív tartalmakat és közösségi funkciókat kínál. A rendszer célja, hogy a Formula 1 rajongóinak folyamatos, valós idejű élményt biztosítson a legfrissebb információkkal és kommunikációs lehetőségekkel.

* **Frontend:** Angular, HTML, CSS, JavaScript (ES6+), Typescript – A felhasználói felület fejlesztéséhez és az alkalmazás dinamikus működéséhez.
* **Backend:** Node.js – A szerveroldali logika kezelése, adatkezelés és API szolgáltatás.
* **Adatbázis:** MySQL – Az adatok tárolása, mint felhasználói adatok, élő közvetítési adatok, üzenetek stb.
* **Biztonság:** JWT (JSON Web Token) alapú hitelesítés, bcrypt – A jelszavak biztonságos tárolására és ellenőrzésére.

A projekt célja, hogy gyors, dinamikus és felhasználóbarát élményt nyújtson, miközben biztosítja a magas szintű biztonságot és stabilitást. A fejlesztési környezet figyelembe veszi az alkalmazás teljesítményét, skálázhatóságát és a cross-browser kompatibilitást.

A dokumentációban részletesen ismertetésre kerül:

* Szükséges rendszer és eszköz követelmény
* Telepítési útmutató a weboldal felépítésére és beüzemelésére
* Weboldal funkcionalitások
* Használt architektúrák és technológiák
* Tesztelési folyamatok ismertetése
* Karbantartási folyamatok ismertetése
* Kliens-oldali kód dokumentációja
* Szerver-oldali kód dokumentációja

Ez a dokumentáció azok számára készült, akik részt kívánnak venni a fejlesztési folyamatban, és segít abban, hogy minden fejlesztő tisztában legyen az alkalmazás (weboldal) működésével, struktúrájával, valamint a fejlesztési irányelvekkel és szabályokkal.

## Rendszerkövetelmény és eszközök

Minimum:

* Op. rendszer: Windows 10, 11
* Processzor: Intel Core i5-7500 / AMD Ryzen 3 3200G
* RAM: 4 GB
* Tárhely: 10 GB
* Hálózat: Széles sávú internetkapcsolat
* Eszközök:
  + Node.js 16.x
  + npm 8.x
  + Angular CLI 19.x
  + Web szerver: Apache
  + Adatbázis: MySQL
  + Verziókezelő: Git

Ajánlott:

* Op. rendszer: Windows 10, 11
* Processzor: Intel Core i5-8400 / AMD Ryzen 5 3400G
* RAM: 8 GB
* Tárhely: 25 GB
* Hálózat: Széles sávú internetkapcsolat
* Eszközök:
  + Node.js 18.x
  + npm 8.x
  + Angular CLI 19.x
  + Web szerver: Apache
  + Adatbázis: MySQL
  + Verziókezelő: Git

## Telepítési útmutató

A weboldal fejlesztéséhez és futtatásához szükséges környezet előkészítése.

1. Repository klónozás

git clone <https://github.com/SzaboKevin2004/Speedformula.git>

1. Környezeti változók beállítása

A .env.example fájlt másolja át .env néven, és adja meg a szükséges értékeket (adatbázis kapcsolat, API kulcsok stb.).

1. Függőségek telepítése

npm install

1. A szerver futtatása

npm start

A weboldal ekkor a http://localhost:4200 címen lesz elérhető.

## Funkcionalitások

Ebben a szekcióban a SpeedFormula által kínált főbb funkciókat és azok működését ismertetjük.

1. Felhasználói regisztráció és bejelentkezés

A regisztráció során a felhasználó megadja az e-mail címét, felhasználónevét és jelszavát.

Bejelentkezéskor a rendszer összehasonlítja a megadott adatokat az adatbázisban tárolt értékekkel.

Token alapú hitelesítést (pl. JWT) alkalmazunk a biztonságos hozzáférés érdekében.

1. Adatkezelés és profilkezelés

A felhasználók frissíthetik profiladataikat, például nevüket, e-mail címüket vagy jelszavukat.

A rendszer támogatja a profilképek feltöltését és módosítását.

Lehetőség van téma módosításra bejelentkezett fiókkal

1. Élő adás chat

Élő adáshoz chat szolgáltatás, amelyet bárki megtekinthet.

Bejelentkezett fiókkal lehetőség van üzenetküldésre.

## Architektúra és technológiai részletek

1. Frontend  
   A frontend a(z) Angular keretrendszert használja, amely biztosítja a dinamikus és reszponzív felhasználói élményt.

CSS technológia a megjelenés testreszabásához.

1. Backend  
   A backend Node.js-en alapul, az Express.js keretrendszer használatával.  
   RESTful API-t biztosít az adatkezeléshez és a frontenddel való kommunikációhoz.
2. Adatbázis  
   Az adatok tárolása MySQL adatbázisban történik.  
   Az adatbázis-struktúra magában foglalja a felhasználók, élő chat, tokenkezelés, fórum tábláit.
3. Hitelesítés és biztonság  
   Token alapú hitelesítés JWT segítségével.  
   Jelszavak biztonságos tárolása bcrypt algoritmussal.

## Tesztelés

1. Egységtesztek

A fejlesztés során a Jest keretrendszert használjuk az egyes komponensek és funkciók tesztelésére.

1. Integrációs tesztek

Az API végpontok helyes működését Postman segítségével teszteljük.

Tesztadatbázis használatával biztosítjuk, hogy a tesztelés ne befolyásolja az éles adatokat.

1. Felhasználói tesztek

A rendszer „béta verzióját” osztálytársak tesztelik, akik visszajelzéseket adnak a funkcionalitásról és a felhasználói élményről.

## Karbantartás

1. Verziókezelés

A projekt fejlesztése Git verziókezeléssel történik.

1. Frissítések és hibajavítások

Rendszeres frissítések biztosítják az új funkciók bevezetését és a hibák javítását.  
Minden frissítést a Trello munkaterünkre naplózunk.

1. Kapcsolattartás

E-mail-cím: [speedformulahu@gmail.com](mailto:speedformulahu@gmail.com)

## Kliens-oldali kód

A kód a frontend alkalmazás részeit tartalmazza, ami a megjelenésért és funkció kezelésekért felelős.

### Auth.service.ts:

1. Bejelentkezett állapot kezelése

private felhBejelentkezettE = new BehaviorSubject<boolean>(false);

felhBejelentkezettE$ = this.felhBejelentkezettE.asObservable();

Leírás:  
Ez a BehaviorSubject tárolja a felhasználó bejelentkezési állapotát (true ha be van jelentkezve, false ha nincs). Az felhBejelentkezettE$ egy Observable, amelyen keresztül bárhol lekérdezhetjük a bejelentkezett állapotot.

2. Profilkép (Avatar) kezelése

private randomKep = new BehaviorSubject<string>("");

randomKep$ = this.randomKep.asObservable();

Leírás:  
A randomKep változó tárolja a felhasználó profilképének elérési útvonalát. Az alapértelmezett érték üres string. A randomKep$ egy Observable, így bárhol hozzáférhetünk az aktuálisan beállított képhez.

3. Felhasználónév kezelése

private felhasznaloNev = new BehaviorSubject<string>("");

felhasznaloNev$ = this.felhasznaloNev.asObservable();

Leírás:  
A felhasznaloNev tárolja a felhasználó felhasználónevét, és az alapértelmezett értéke egy üres string. A felhasznaloNev$ egy Observable, amelyen keresztül bármikor lekérhetjük a felhasználó aktuális nevét.

4. Téma szín beállítása

private szamSzin = new BehaviorSubject<number>(1);

szamSzin$ = this.szamSzin.asObservable();

Leírás:  
A szamSzin változó tárolja a felhasználó által választott téma színének értékét (pl. világos, sötét, stb.). Az alapértelmezett érték 1, ami világos témát jelent. A szamSzin$ egy Observable, amelyen keresztül bármikor lekérhetjük a témát.

5. Regisztráció

regisztracio(regisztracioData: { felhasznalonev: string, email: string, password: string, confirm\_password: string, kep: number }) {

  return this.http.post(`${this.url}/regist`, regisztracioData).pipe(

    catchError((error) => {

      return throwError(() => error);

    })

  );

}

Leírás:  
Ez a metódus a felhasználó regisztrációját végzi el, elküldve az adatokat az API végpontra. A regisztracioData paraméter tartalmazza a felhasználó nevét, email címét, jelszót, jelszó megerősítést és a profilkép kiválasztott indexét.

6. Bejelentkezés

bejelentkezes(loginData: { felhasznalonev: string, email: string, password: string }) {

  return this.http.post<AuthResponse>(`${this.url}/login`, loginData).pipe(

    catchError((error) => {

      return throwError(() => error);

    }),

    tap((response) => {

      this.setTheme(response.tema);

      this.setToken(response.token);

      this.setPfpId(response.pfp);

      this.setFelhasznaloNev(response.username);

    })

  );

}

Leírás:   
A bejelentkezes metódus végzi el a felhasználó bejelentkezését. Az API válaszában kapott adatokat (token, pfp, username, tema) elmenti, hogy a felhasználó a következő oldalbetöltésnél is bejelentkezve maradjon.

7. Profil módosítása

profilModositas(

  felhasznalonev: string | undefined = undefined,

  email: string | undefined = undefined,

  password: string | undefined = undefined,

  tema\_id: number | undefined = undefined,

  kep: number | undefined = undefined

) {

  const token = localStorage.getItem('token');

  if (!token) {

    throw new Error('Nincs bejelentkezett felhasználó!');

  }

  return this.http.patch(`${this.url}/profil`,

    { felhasznalonev, email, password, tema\_id },

    {

      headers: {

        'Authorization': `Bearer ${token}`,

        'Content-Type': 'application/json',

      }

    }).pipe(

      catchError((error) => {

        return throwError(() => error);

      })

    );

}

Leírás:  
A profilModositas metódus lehetővé teszi a felhasználói profil frissítését (felhasználónév, email, jelszó, téma, stb.). A módosításokat csak akkor lehet végrehajtani, ha a felhasználó be van jelentkezve, mivel az API kéréshez szükség van egy érvényes tokenre.

8. Üzenet küldése a chatbe

sendChatMessage(uzenet: string) {

  const token = localStorage.getItem('token');

  if (!token) {

    throw new Error('Nincs bejelentkezett felhasználó!');

  }

  return this.http.post(`${this.url}/chat`,

    { uzenet },

    {

      headers: {

        'Authorization': `Bearer ${token}`,

        'Content-Type': 'application/json',

      }

    }).pipe(

      catchError((error) => {

        return throwError(() => error);

      }),

      tap(() => {

        this.getMessages();

      })

    );

}

Leírás:  
A sendChatMessage metódus lehetővé teszi, hogy a felhasználó üzenetet küldjön a chat alkalmazásba. A sikeres üzenetküldést követően újra lekérdezi az üzeneteket (getMessages).

9. Üzenetek lekérése a chatből

getMessages(): Observable<any[]> {

  const token = localStorage.getItem('token');

  let headers = new HttpHeaders().set('Content-Type', 'application/json');

  if (token) {

    headers = headers.set('Authorization', `Bearer ${token}`);

  }

  return this.http.get<any[]>(`${this.url}/chat`, { headers }).pipe(

    catchError((error) => {

      return throwError(() => error);

    }),

    tap((messages) => {

      this.uzenetekSubject.next(messages);

    })

  );

}

Leírás:  
A getMessages metódus lekéri az összes chat üzenetet az API-ból, és frissíti az uzenetekSubject-et az új üzenetekkel. Az API kéréshez szükséges a felhasználó tokenje.

10. Kijelentkezés

kijelentkezes() {

  if (typeof window !== 'undefined' && window.localStorage) {

    localStorage.removeItem('token');

    localStorage.removeItem('pfp');

    localStorage.removeItem('username');

    localStorage.removeItem('tema');

  }

  window.location.reload();

  this.felhBejelentkezettE.next(false);

  this.router.navigate(['/']);

}

Leírás:  
A kijelentkezes metódus végzi el a felhasználó kijelentkezését, eltávolítva a localStorage-ból a felhasználó adatait, és frissíti az alkalmazást. A felhasználó visszairányításra kerül az alapértelmezett oldalra (/).

11. Profilképek elérési útjai

private kepEleres = [

  "../assets/pfp/pfp\_black.png",

  "../assets/pfp/pfp\_grey.png",

  "../assets/pfp/pfp\_green.png",

  "../assets/pfp/pfp\_dark-green.png",

  "../assets/pfp/pfp\_blue.png",

  "../assets/pfp/pfp\_dark-blue.png",

  "../assets/pfp/pfp\_pink.png",

  "../assets/pfp/pfp\_dark-pink.png",

  "../assets/pfp/pfp\_magenta.png",

  "../assets/pfp/pfp\_dark-magenta.png",

  "../assets/pfp/pfp\_yellow.png",

  "../assets/pfp/pfp\_dark-yellow.png",

  "../assets/pfp/pfp\_brown.png",

  "../assets/pfp/pfp\_dark-brown.png",

  "../assets/pfp/pfp\_cyan.png",

  "../assets/pfp/pfp\_dark-cyan.png",

  "../assets/pfp/pfp\_orange.png",

  "../assets/pfp/pfp\_dark-orange.png",

  "../assets/pfp/pfp\_purple.png",

  "../assets/pfp/pfp\_dark-purple.png",

  "../assets/pfp/pfp\_red.png",

  "../assets/pfp/pfp\_dark-red.png"

];

Leírás:  
A kepEleres tömb tartalmazza a felhasználói profilképek elérési útvonalait. Ezek a képek a felhasználó számára választhatók a profil beállításakor.

12. Téma beállítás

setTheme(tema: number) {

  this.szamSzin.next(tema);

  if (typeof window !== 'undefined' && window.localStorage) {

    localStorage.setItem('tema', tema.toString());

  }

}

Leírás:  
A setTheme metódus lehetővé teszi a felhasználó téma beállítását. A kiválasztott téma értéke (tema) elmentésre kerül a localStorage-ba is, így a beállítás megmarad az oldal újratöltése után.

### App.component.ts:

Ez az alkalmazás fő komponense, amely felelős a navigációs menü kezeléséért, a felhasználói profil megjelenítéséért és a felhasználói bejelentkezés állapotának kezeléséért. Az alkalmazás témáját és stílusait is itt lehet dinamikusan módosítani az aktuális felhasználói beállítások alapján. A komponens tartalmazza az összes szükséges logikát a megfelelő felhasználói élmény biztosításához, figyelve a különböző URL-ek és navigációs események változására.

1. Navigációs Menü Megjelenítése

this.navMegjelenites = !(jelenlegiUrl.includes('bejelentkezes') ||

jelenlegiUrl.includes('regisztracio') ||

jelenlegiUrl.includes('hiba') ||

jelenlegiUrl.includes('visszajelzes'));

Leírás:  
Ez a kódrészlet felelős a navigációs menü megjelenítéséért. A menü csak akkor lesz látható, ha az aktuális URL nem tartalmazza a következő oldalakat: bejelentkezes, regisztracio, hiba, vagy visszajelzes. Így a navigációs menü elrejtésre kerül, amikor ezekre az oldalakra navigálunk.

2. Felhasználói Bejelentkezés Állapotának Kezelése

this.authservice.felhBejelentkezettE$.subscribe(bejelentkezettE => {

  this.bejelentkezesMegjelenites = !bejelentkezettE;

  this.profilMegjelenites = bejelentkezettE;

});

Leírás:  
Ez a kódrészlet figyeli a felhasználó bejelentkezés állapotát az AuthService szolgáltatás felhBejelentkezettE$ adatfolyamán keresztül. Amennyiben a felhasználó be van jelentkezve (bejelentkezettE igaz), a profil menü megjelenik, míg a bejelentkezési lehetőség elrejtésre kerül. Ha a felhasználó nincs bejelentkezve, a helyzet ellentéte történik meg.

3. Profilkép Frissítése

this.authservice.randomKep$.subscribe(kep => {

  this.randomKep = kep;

  console.log(kep);

});

Leírás:  
Ez a kódrészlet az AuthService által biztosított randomKep$ adatfolyamot figyeli, és frissíti a profilkép URL-jét a komponensben. Minden alkalommal, amikor a felhasználó profilképe frissül, ez a kódrészlet automatikusan elmenti az új képet, és a felhasználó profiljában megjeleníti azt.

4. Felhasználóneve Frissítése

this.authservice.felhasznaloNev$.subscribe(nev => {

  this.felhasznaloNev = nev;

});

Leírás:  
Ez a kódrészlet figyeli az AuthService által biztosított felhasznaloNev$ adatfolyamot, és frissíti a felhasználó nevét a komponensben. Minden alkalommal, amikor a felhasználó neve frissül (pl. bejelentkezés után), ez a kódrészlet automatikusan elmenti az új nevet, és a felhasználó profiljában megjeleníti azt.

5. Téma Beállítása

this.authservice.szamSzin$.subscribe(szam => {

  if (szam === 1) {

    this.sotet = true;

    this.vilagos = false;

    this.voros = false;

    this.temaSzin = 'feketeK';

    this.temaSzin2 = 'feketeK2';

    this.temaSzinN = 'feketeN';

    this.temaSzinBetu = 'feketeBetu';

    this.temaSzinHover = 'feketeH';

    this.temaSzinGordulo = 'feketeG';

    this.temaSzinHr = 'feketeHr';

    if (typeof document !== 'undefined') {

      document.body.className = 'feketeBg';

    }

  } else if (szam === 2) {

    this.sotet = false;

    this.vilagos = true;

    this.voros = false;

    this.temaSzin = 'feherK';

    this.temaSzin2 = 'feherK2';

    this.temaSzinN = 'feherN';

    this.temaSzinBetu = 'feherBetu';

    this.temaSzinHover = 'feherH';

    this.temaSzinGordulo = 'feherG';

    this.temaSzinHr = 'feherHr';

    if (typeof document !== 'undefined') {

      document.body.className = 'feherBg';

    }

  } else if (szam === 3) {

    this.sotet = false;

    this.vilagos = false;

    this.voros = true;

    this.temaSzin = 'vorosK';

    this.temaSzin2 = 'vorosK2';

    this.temaSzinN = 'vorosN';

    this.temaSzinBetu = 'vorosBetu';

    this.temaSzinHover = 'vorosH';

    this.temaSzinGordulo = 'vorosG';

    this.temaSzinHr = 'vorosHr';

    if (typeof document !== 'undefined') {

      document.body.className = 'vorosBg';

    }

  } else {

    this.sotet = true;

    this.vilagos = false;

    this.voros = false;

    this.temaSzin = 'feketeK';

    this.temaSzin2 = 'feketeK2';

    this.temaSzinN = 'feketeN';

    this.temaSzinBetu = 'feketeBetu';

    this.temaSzinHover = 'feketeH';

    this.temaSzinGordulo = 'feketeG';

    this.temaSzinHr = 'feketeHr';

    if (typeof document !== 'undefined') {

      document.body.className = 'feketeBg';

    }

  }

});

Leírás:  
Ez a kódrészlet az alkalmazás témáját frissíti a szamSzin$ adatfolyamon keresztül, amely az AuthService által biztosított adatokat figyeli. Három lehetséges téma közül választhat a felhasználó:  
Sötét téma: A megfelelő osztályok beállítása mellett sötét háttér és világos szöveg jelenik meg.  
Világos téma: Fehér háttér és sötét szöveg jelenik meg.  
Vörös téma: Vörös háttér és szöveg jelenik meg.  
A megfelelő színvilág alkalmazásával az oldalak háttérszínét dinamikusan módosítja a kódrészlet.

6. Fehér Oldal Ellenőrzése

isFeherOldal(): boolean {

  return this.router.url === '/visszajelzes' ||

  this.router.url === '/bejelentkezes' ||

  this.router.url === '/regisztracio' ||

  this.router.url === '/hiba';

}

Leírás:  
Ez a segédfüggvény ellenőrzi, hogy a jelenlegi oldal fehér háttérrel rendelkezik-e. Ha az URL tartalmazza a /visszajelzes, /bejelentkezes, /regisztracio vagy /hiba útvonalakat, akkor fehér háttérrel rendelkező oldalnak tekinti.

7. Profil Menü Megjelenítése

profilMenu() {

  this.profilMenuMegjelenites = !this.profilMenuMegjelenites;

}

Leírás:  
Ez a metódus a profil menü megjelenítését kapcsolja be és ki. Ha a menü jelenleg nem látható, akkor láthatóvá teszi, ha már látható, akkor elrejti.

8. Profil Menü Elrejtése

profilMenuHA() {

  if (this.profilMenuMegjelenites) {

    this.profilMenuMegjelenites = false;

  }

}

Leírás:  
Ez a metódus a profil menü eltüntetésére szolgál. Ha a menü jelenleg meg van nyitva, akkor bezárja azt.

9. Kijelentkezés

kijelentkezes() {

  this.authservice.kijelentkezes();

  this.router.navigate(['/']);

}

Leírás:  
A kijelentkezes() metódus a felhasználót kijelenti az alkalmazásból, eltávolítja a helyi tárolóból (localStorage) a felhasználó adatait, majd visszairányítja őt az alapértelmezett kezdőoldalra (/).

### regisztracio.component.ts

Változók és Kezdeti Állapotok

A következő változókat használjuk a komponensben, amelyek segítenek tárolni a regisztrációs űrlap értékeit, kezelni a hibákat és sikerüzeneteket, illetve a megjelenítési logikát.

felhasznalonev: string = '';

email: string = '';

password: string = '';

confirm\_password: string = '';

hiba: boolean = false;

hibaUzenet: string = '';

siker: boolean = false;

sikerUzenet: string = '';

keretMegjelenites: boolean = true;

felhasznalonev, email, password, confirm\_password: Ezek a változók tárolják az űrlap adatbeviteli mezőinek értékeit. A felhasználó itt adja meg:

A felhasználónevét (felhasznalonev)

Az e-mail címét (email)

A jelszavát (password)

A megerősítő jelszót (confirm\_password).

hiba: Ez a logikai változó a regisztráció során felmerülő hibák kezelésére szolgál. Ha hibát észlelünk, értéke true lesz, és megjelenítjük a hibaüzenetet.

hibaUzenet: Ha a regisztráció hibába ütközik, itt jelenik meg a hibaüzenet, amely a felhasználónak tájékoztatást ad a problémáról.

siker: Ez a logikai változó azt jelzi, hogy a regisztráció sikeres volt-e. Ha true, akkor sikeres regisztrációról van szó, és megjelenítjük a sikerüzenetet.

sikerUzenet: Ha a regisztráció sikeres, ez a változó tartalmazza a sikerüzenetet, amelyet a felhasználónak megjelenítünk.

keretMegjelenites: A regisztrációs űrlap láthatóságát szabályozza. Ha a regisztráció sikeres, a űrlap eltűnik (keretMegjelenites = false), és csak a sikerüzenet jelenik meg.

authService: Az AuthService szolgáltatás felelős a regisztrációs kérés elküldéséért a backend felé. Az Angular HttpClient szolgáltatás segítségével kezeli az HTTP kéréseket.

router: A Router szolgáltatás az Angular beépített eszköze, amely lehetővé teszi az útvonalak kezelését. Ebben az esetben használjuk a regisztrációs folyamat utáni navigálásra, hogy átirányítsuk a felhasználót a bejelentkezési oldalra.

Hiba állapot nullázása:  
A metódus elején töröljük az esetleges korábbi hibákat. A hiba változót false-ra állítjuk, és a hibaUzenet üzenetet is kiürítjük, hogy ha volt valamilyen korábbi hiba, az ne zavarja a jelenlegi regisztrációs folyamatot.

regisztracioData Objektum Létrehozása:  
A felhasználó által megadott adatokat összegyűjtjük egy regisztracioData objektumba. Az objektumban szereplő adatok a regisztrációhoz szükségesek, beleértve a felhasználó nevét, e-mail címét, jelszavát, megerősítő jelszót, és egy véletlenszerűen generált képet (a kep változó).

Regisztrációs Kérelem Küldése:  
Az authService.regisztracio(regisztracioData) meghívásával elindítjuk az HTTP POST kérést a backend felé. Az Observable válaszát figyeljük.

Sikeres Válasz Kezelése:  
Ha a regisztráció sikeres, a next blokkban az űrlap eltűnik (keretMegjelenites = false), és a sikeres regisztrációról üzenetet jelenítünk meg (sikerUzenet).

Ezt követően egy 1,5 másodperces késleltetés után átirányítjuk a felhasználót a bejelentkezési oldalra (this.router.navigate(['/bejelentkezes'])).

Hiba Kezelése:  
Ha hiba történik (pl. az e-mail cím már foglalt), akkor a catch blokkban az error változóban lévő hibaüzenetet jelenítjük meg. Ha a hibaüzenet elérhető (err.error?.message), akkor azt jelenítjük meg, egyébként egy általános hibaüzenetet adunk.

## Szerver-oldali kód

A kód egy Node.js alapú backend alkalmazás egy részét tartalmazza, amely az F1-es hírek, versenyzők, csapatok és versenynaptár adatait szolgáltatja külső API-k segítségével. Az export default objektum több aszinkron függvényt tartalmaz, amelyek különböző adatokat kérnek le és továbbítanak egy Express.js alapú HTTP válaszként.

### Hírek lekérése – HírekGetController

Ez a függvény az F1-hez kapcsolódó híreket kéri le a NewsAPI szolgáltatásból. A következőképpen működik:

NewsAPI kliens jön létre a következő paranccsal:

const newsapi = new NewsAPI('0839a4ca5d3247f0af119d1e3053bca2');

Az everything végpontot hívja meg, és a hírek keresési kulcsszava "F1", a nyelv pedig magyar ("hu").

const válasz = await newsapi.v2.everything({

    q: 'F1',

    language: 'hu',

    sortBy: 'publishedAt'

});

Amennyiben a válasz státusza "ok", akkor végigmegy az összes cikk listáján (articles tömbön) és ellenőrzi, hogy van-e hozzárendelt kép (urlToImage). Ha nincs, egy alapértelmezett képet állít be.

const alapKep = 'https://pcdn.hu/articles/images-xl/f/o/r/forma1-709165.jpg';

válasz.articles.forEach(article => {

    if(!article.urlToImage){

        article.urlToImage = alapKep;

    }

});

A módosított híreket visszaküldi a kliensnek (res.status(200).json(válasz.articles)).

Ha hiba történik a lekérés során, a rendszer egy hibaüzenetet küld (400-as vagy 500-as státuszkóddal).

### F1-es versenyek lekérése – SessionsGetController

Ez a függvény a versenynaptár adatait kérdezi le az OpenF1 API-ról.

A fetch függvényt használja az adatok letöltésére:

const alapSessions = await fetch('https://api.openf1.org/v1/session');

A letöltött adatokat egyszerűen továbbítja a kliens felé.

return res.status(200).json(alapSessions);

Ha hiba történik, akkor egy 500-as státuszkódú hibaválaszt küld.

### Csapatok lekérése – CsapatokGetController

Ez a végpont a Forma–1-es csapatok listáját kéri le az Ergast API-ból.

Az API egyszerre csak 1000 elemet tud visszaadni, ezért három külön lekérést hajt végre különböző offset értékekkel:

const válasz = await fetch('http://ergast.com/api/f1/constructors.json?limit=1000');

const válaszkettő = await fetch('http://ergast.com/api/f1/constructors.json?limit=1000&offset=101');

const válaszhárom = await fetch('http://ergast.com/api/f1/constructors.json?limit=1000&offset=202');

Ha bármelyik kérés sikertelen (!válasz.ok || !válaszkettő.ok || !válaszhárom.ok), akkor egy 500-as státuszkódú hibaüzenet érkezik vissza.

Ha minden sikeres, az API válaszait JSON formátumban összefűzi:

const d1 = await válasz.json();

const d2 = await válaszkettő.json();

const d3 = await válaszhárom.json();

const csapatok = {

    első: d1.MRData.ConstructorTable.Constructors,

    második: d2.MRData.ConstructorTable.Constructors,

    harmadik: d3.MRData.ConstructorTable.Constructors

};

Végül ezt az objektumot adja vissza válaszként.

### Versenyzők lekérése – VersenyzőkGetController

Ez a végpont a Forma–1-es versenyzők adatait kérdezi le az OpenF1 API-ból.

Az adatokat egy egyszerű fetch hívással tölti le:

const válasz = await fetch('https://api.openf1.org/v1/drivers');

Ha a lekérés sikertelen (!válasz.ok), 500-as státuszkódú hibaüzenetet küld.

Ha sikeres, a válasz JSON formátumban visszaküldésre kerül.

### Üzenet küldése – ChatPost

Ez az endpoint egy új chatüzenetet hoz létre az adatbázisban.

Hitelesítés: A req.headers.authorization fejlécből kinyeri a Bearer token-t, majd dekódolja a JWT segítségével:

const token = req.headers.authorization?.split(' ')[1];

const dekódolt = jwt.verify(token, process.env.JWT\_SECRET);

Ha a token érvénytelen vagy lejárt, a szerver egy 401-es hibát küld vissza.

Felhasználó ellenőrzése: Az adatbázisból kikeresi a felhasználót id alapján:

Felhasználó.findByPk(dekódolt.id).then(async(felhasználó) => { ... });

Ha a felhasználó nem található, 404-es hibát küld vissza.

Üzenetellenőrzés: Ha a kérelemben nincs uzenet, a szerver 400-as státuszkódú hibát küld vissza.

Chatüzenet létrehozása:

const chat = Chat.build({

    felhasznalo\_id: dekódolt.id,

    uzenet: req.body.uzenet

});

await chat.save();

A chatobjektumot létrehozza, elmenti az adatbázisba, majd 201-es státuszkóddal visszajelez.

Lehetséges hibák kezelése:

Token lejárt vagy érvénytelen (401-es hiba)

Adatbázis-hiba (500-as hiba)

### Összes chatüzenet lekérése – ChatGet

Ez az endpoint az adatbázisból lekérdezi az összes üzenetet, a küldő felhasználónévvel és profilképpel együtt.

Üzenetek lekérése az adatbázisból:

const chat = await Chat.findAll({

    order: [['createdAt', 'ASC']],

    include: [{ model: Felhasználó, attributes: ['felhasznalonev', 'kep'] }]

});

Az üzenetek létrehozási dátum szerint rendezve (createdAt) érkeznek vissza.

Formázás: A kapott adatokat egy könnyen feldolgozható JSON formátumba alakítja:

const formazottChat = chat.map(message => ({

    id: message.id,

    uzenet: message.uzenet,

    elküldve: message.createdAt,

    felhasznalo: message.felhasználó.felhasznalonev,

    kep: message.felhasználó.kep

}));

Ez biztosítja, hogy a frontenden megfelelő formátumban jelenjenek meg az üzenetek.

Válasz küldése:

return res.status(200).json(formazottChat);

Ha sikeres, az üzenetek 200-as státuszkóddal visszaküldésre kerülnek.

Hibakezelés:

Adatbázishiba esetén: 500-as státuszkód és hibaüzenet küldése.

3. Egy adott üzenet törlése – ChatIdDelete

Ez az endpoint egy konkrét üzenet törlésére szolgál az id alapján.

JWT hitelesítés:

const token = req.headers.authorization?.split(' ')[1];

const dekódolt = jwt.verify(token, process.env.JWT\_SECRET);

A hitelesítéshez használt kód megegyezik a ChatPost végpontban látottakkal.

Üzenet ellenőrzése az adatbázisban:

const chat = await Chat.findOne({ where: { id } });

if (!chat) {

    return res.status(404).json({ error: true, message: "A megadott üzenet nem található!" });

}

Ha az üzenet nem található, akkor 404-es státuszkód kerül visszaküldésre.

Felhasználó jogosultságának ellenőrzése:

if (felhaszáló.szerep\_id !== 1 && chat.felhasznalo\_id !== felhasználó.id) {

    return res.status(403).json({ error: true, message: "Nincs jogosultságod a törölni!" });

}

Csak az adminisztrátorok (szerep\_id === 1) és az üzenet eredeti küldője törölheti az üzenetet.

Üzenet törlése az adatbázisból:

await Chat.destroy({ where: { id } });

return res.status(200).json({

    error: false,

    message: "Üzenet sikeresen törölve!"

});

Hibakezelés:

Nincs jogosultság (403-as hiba)

Token lejárt vagy érvénytelen (401-es hiba)

Adatbázis-hiba (500-as hiba)

### Az összes üzenet törlése – ChatDeleteAll

Ez az endpoint minden üzenetet töröl az adatbázisból.

Üzenetek törlése:

await Chat.destroy({ where: {} });

Ez a parancs minden üzenetet eltávolít az adatbázisból.

Sikeres válasz küldése:

res.status(200).json({

    error: false,

    message: "Összes üzenet sikeresen törölve!"

});

Hibakezelés:

Adatbázis-hiba esetén 500-as státuszkód és hibaüzenet küldése.

### Regisztráció (RegisztracioPostController)

A regisztrációs végpont lehetővé teszi új felhasználók létrehozását az adatbázisban. A következő lépések történnek:

1.1. Adatok ellenőrzése

A kód ellenőrzi, hogy minden kötelező mező ki van-e töltve:

if (!req.body.felhasznalonev || !req.body.email || !req.body.password || !req.body.confirm\_password) {

    return res.status(400).json({ error: true, message: "Minden mező kitöltése kötelező!" });

}

1.2. Egyediség ellenőrzése

A rendszer megnézi, hogy a megadott felhasználónév vagy e-mail cím már létezik-e az adatbázisban:

const létezőfelhasznalonev = await Felhasználó.findOne({where:{felhasznalonev:req.body.felhasznalonev}});

if(létezőfelhasznalonev){

    res.status(409).json({ error: true, message:"A felhasználónév már foglalt!" });

    return;

}

Ha az e-mail cím már használatban van, akkor a következő üzenetet kapja a felhasználó:

if(létezőEmail){

    res.status(409).json({ error: true, message:"Az email cím már foglalt!" });

    return;

}

1.3. E-mail és jelszó ellenőrzése

A rendszer ellenőrzi, hogy az e-mail formátuma helyes-e:

if(!isEmail(req.body.email)){

    res.status(400).json({ error: true, message:"Érvénytelen email cím!" });

    return;

}

A jelszó ellenőrzése egy regex segítségével történik:

if (!/^(?=.\*[A-Z])(?=.\*\d).{8,}$/.test(req.body.password)) {

    return res.status(400).json({ error: true, message: "A jelszónak minimum 8 karakternek kell lennie, és tartalmaznia kell egy nagy betűt és számot!" });

}

1.4. Jelszó titkosítása és adatbázisba mentés

A jelszó bcrypt segítségével titkosításra kerül:

let titkosPassword = await bcrypt.hash(req.body.password, 10);

A felhasználói adatok mentése:

const felhasználó = Felhasználó.build({

    felhasznalonev: req.body.felhasznalonev,

    email: req.body.email,

    password: titkosPassword,

    szerep\_id: szerep.id,

    tema\_id: 1,

    kep: req.body.kep

});

felhasználó.save();

Sikeres regisztráció esetén:

res.json({ error: false, message: "Sikeres regisztráció!", felhasználó\_id: felhasználó.id });

### Bejelentkezés (LoginPostController)

A bejelentkezési végpont a JWT segítségével hitelesíti a felhasználót.

2.1. Adatok ellenőrzése

A rendszer ellenőrzi, hogy a felhasználó vagy e-mail meg lett-e adva:

if(!req.body.felhasznalonev&& !req.body.email){

    return res.status(400).json({ error:true, message:"Felhasználónév/email megadása kötelező!" });

}

2.2. Felhasználó keresése az adatbázisban

A felhasználó vagy az e-mail alapján történik a keresés:

let felhasználó;

if (req.body.felhasznalonev){

    felhasználó = await Felhasználó.findOne({where:{felhasznalonev}});

}else{

    felhasználó = await Felhasználó.findOne({where:{email}});

}

2.3. Jelszó ellenőrzése

A rendszer a beírt jelszót a titkosított jelszóval hasonlítja össze:

const jó=await bcrypt.compare(req.body.password, felhasználó.password);

if(!jó){

    return res.status(401).json({ error: true, message: "Hibás jelszó!" });

}

2.4. JWT token generálása

A JWT titkos kulccsal aláírásra kerül:

const token=jwt.sign({ id: felhasználó.id, felhasznalonev:felhasználó.felhasznalonev }, process.env.JWT\_SECRET, { expiresIn: '6h' });

Sikeres bejelentkezés esetén:

res.status(200).json({ success: true, token, username: felhasználó.felhasznalonev });

### Kijelentkezés (LogoutPostController)

A kijelentkezési végpont csak egy visszajelzést ad a kliensnek:

res.status(200).json({ success: true, message: "Sikeres kijelentkezés!" });

### Profillekérés (ProfilGetController)

A JWT token segítségével a felhasználó adatai lekérhetők:

const dekódolt=jwt.verify(token, process.env.JWT\_SECRET);

const felhasználó= await Felhasználó.findOne({

    where: { id: dekódolt.id },

    include:[{ model: Szerep, attributes: ['szerep\_neve'] }],

    attributes:{exclude:['password']}

});

### Profil módosítása (ProfilePatchController)

A felhasználó módosíthatja adatait, például a felhasználónevét, e-mail címét vagy profilképét:

if(req.body.felhasznalonev!==""){

    felhasználó.felhasznalonev=req.body.felhasznalonev;

}

if(req.body.email!==""){

    felhasználó.email=req.body.email;

}

### Profil törlése (ProfilDeleteController)

A rendszer törli a felhasználót az adatbázisból:

const törlés=await Felhasználó.destroy({ where: { id: dekódolt.id } });

if(törlés===0){

    res.status(404).json({ error: true, message: "Felhasználó nem található!" });

}

# Felhasználói dokumentáció

## Bevezetés

Ez a dokumentáció a SpeedFormula weboldal használatához szükséges információkat tartalmazza. A weboldal egy modern, dinamikus platform, amely lehetővé teszi a felhasználók számára, hogy élő közvetítéseket nézzenek, interaktív tartalmakat élvezzenek, és részt vegyenek a közösségi élményben. A célunk, hogy a Formula 1 rajongói számára valós idejű, könnyen elérhető élményt nyújtsunk, valamint biztosítsuk a legfrissebb információkat és kommunikációs lehetőségeket. A felhasználói dokumentáció segít abban, hogy könnyedén csatlakozhasson a weboldalhoz, regisztrálhasson új fiókot, beléphessen és teljes mértékben kihasználhassa az oldalon elérhető funkciókat. Az alkalmazás célja, hogy egyszerű és élvezetes felhasználói élményt biztosítson minden látogató számára.

A dokumentáció tartalmazza az alábbi témaköröket:

* Rendszerkövetelmény
* Weboldalon található tartalmak, lehetőségek
* Hogyan csatlakozzon a weboldalhoz és regisztráljon új felhasználói fiókot
* Hogyan léphet be és kezelheti felhasználói fiókját
* Az elérhető beállítások és testreszabási lehetőségek
* Hogyan használja a weboldalt és az interaktív funkciójait

## Rendszerkövetelmény

Minimum:

* Op. rendszer: Windows 7
* Processzor: Intel Core i5-6500 / AMD Ryzen 3 1300X
* RAM: 4 GB
* Tárhely: 100 MB szabad hely
* Internet sebesség: 25 Mbps
* Böngésző: Google Chrome 58 / Mozilla Firefox 54 / Opera 45 / Microsoft Edge 79 / Apple Safari 12

Ajánlott:

* Op. rendszer: Windows 10, 11
* Processzor: Intel Core i5-7500 / AMD Ryzen 5 3400G
* RAM: 8 GB
* Tárhely: 1 GB szabad hely
* Internet sebesség: 50 Mbps
* Böngésző (legújabb): Google Chrome / Mozilla Firefox / Opera / Microsoft Edge / Apple Safari