

**SpeedFormula Dokumentáció**

Tartalomjegyzék

[Bevezetés 3](#_Toc192208351)

[Projekt jellemzés 3](#_Toc192208352)

[Használt csoportmunka eszközök 4](#_Toc192208353)

[Fejlesztői dokumentáció 5](#_Toc192208354)

[Rendszerkövetelmény és eszközök 5](#_Toc192208355)

[Telepítési és futtatási útmutató 5](#_Toc192208356)

[Telepítés menete 5](#_Toc192208357)

[Környezet előkészítés menete 9](#_Toc192208358)

[Funkcionalitások 15](#_Toc192208359)

[Architektúra és technológiai részletek 16](#_Toc192208360)

[Karbantartás 16](#_Toc192208361)

[Frontend komponens dokumentációja 16](#_Toc192208362)

[Routing működése, frontend végpontok felépítése 16](#_Toc192208363)

[Hitelesítés, API kommunikáció és Adatkezelés 18](#_Toc192208364)

[auth.service.ts: 18](#_Toc192208365)

[forum.service.ts 28](#_Toc192208366)

[elo.service.ts 29](#_Toc192208367)

[api.service.ts 31](#_Toc192208368)

[visszajelzes.service.ts 31](#_Toc192208369)

[Elkészített komponensek és működésük 32](#_Toc192208370)

[Backend komponens dokumentációja 32](#_Toc192208371)

[Felhasználói dokumentáció 32](#_Toc192208372)

[Rendszerkövetelmény 32](#_Toc192208373)

[Használati útmutató 33](#_Toc192208374)

[Tesztelés 33](#_Toc192208375)

[Frontend 33](#_Toc192208376)

[Backend 33](#_Toc192208377)

[Terheléses teszt 33](#_Toc192208378)

[Továbbfejlesztési lehetőségek 33](#_Toc192208379)

[Irodalomjegyzék 33](#_Toc192208380)

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

# Bevezetés

A weboldal egy modern, dinamikus alkalmazás, amely a felhasználók számára élő közvetítés megosztást, interaktív tartalmakat és közösségi funkciókat kínál.

Ez a dokumentáció a SpeedFormula weboldal fejlesztéséhez és használatához szükséges részleteket tartalmazza, amik 2 fő részre csoportosíthatóak. Fejlesztői dokumentációra és Felhasználói dokumentációra.

A fejlesztői 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

A fejlesztői 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. Ehhez kapcsolódó részletek megértéséhez szükséges ezekben, vagy ezekhez hasonló rendszerekben szerzett tapasztalat/tudás, mint például Angular keretrendszer, Node.js futtatókörnyezet, MySQL adatbázis.

A felhasználói 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

A felhasználói dokumentáció segít abban, hogy könnyedén csatlakozhasson egy felhasználó a weboldalhoz, regisztrálhasson új fiókot, beléphessen és teljes mértékben kihasználhassa az oldalon elérhető funkciókat.

# Projekt jellemzés

A SpeedFormula egy olyan weboldal, amely kifejezetten a magyar Forma-1 rajongók számára készült. A célja, hogy egyetlen platformon biztosítson hozzáférést minden fontos információhoz, hírekhez és közösségi funkciókhoz, lehetővé téve a szurkolók számára, hogy aktívan részt vehessenek a sport körüli beszélgetésekben

Főbb funkciók és követelmények:

* Fórum: A felhasználók kérdéseket tehetnek fel, válaszolhatnak mások kérdéseire, valamint posztokat hozhatnak létre a Forma-1-gyel kapcsolatos témákban.
* Hírek: A legfrissebb hírek és események összegyűjtése, hogy a rajongók mindig naprakészen értesüljenek a sport világáról.
* Élő adás és chat: Versenyek közvetítése élőben (beágyazott élő adás), ahol a felhasználók egy valós idejű chatben beszélgethetnek egymással, megoszthatják véleményüket és reakcióikat.

Igények és elvárások:

* Felhasználóbarát felület: Könnyen kezelhető, modern dizájn, amely minden korosztály számára egyszerű navigációt biztosít.
* Reszponzivitás: A weboldal (és aloldalai) minden eszközön (mobil, tablet, asztali gép) használható legyen és megfelelően jelenjen meg.
* Biztonság: A felhasználó email címe, jelszava és egyéb adatai megfelelően titkosítva legyenek, azok bizalmasan legyenek kezelve és tárolva.
* Skálázhatóság: A rendszernek képesnek kell lennie a növekvő, mégis széleskörű „felhasználó bázist” kiszolgálni, teljesítménybeli problémák megléte nélkül.

# Használt csoportmunka eszközök

A SpeedFormula weboldal fejlesztésekor különböző csoportmunka-eszközöket alkalmaztunk a hatékony együttműködés, feladatkezelés és kommunikáció érdekében.

1. **GitHub** – Verziókezelés és párhuzamos fejlesztés

A projekt verziókezelésére és a kód tárolására a GitHub platformot használtuk. Az alábbi főbb funkciókat használtuk:

* Git verziókezelés a forráskód nyomon követésére és verzióinak kezelésére.
* Repository kezelése, ahol a teljes projektkód elérhető.
* Branching és Pull Request-ek a párhuzamos fejlesztés érdekében.

1. **Trello** – Feladatkezelés és projektmenedzsment

A Trello segítségével a közösen szerveztük a fejlesztési folyamatot és a teendőket.

* Felelősök meghatározása
* Fejlesztési szakaszok követése (Naplózás tábla)
* Határidők (Ütemezés tábla) a korai tervezési fázisban

1. **Messenger** – Kommunikáció és megbeszélések

A gyors és egyszerű kommunikációhoz a Messenger csoportos üzenetküldő alkalmazást használtunk:

* Azonnali kommunikáció a csapattagok között.
* Gyors döntéshozatal fejlesztési kérdésekben.
* Heti online meetingek a haladás és az elkövetkezendők egyeztetésére.

# Fejlesztői dokumentáció

## 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:

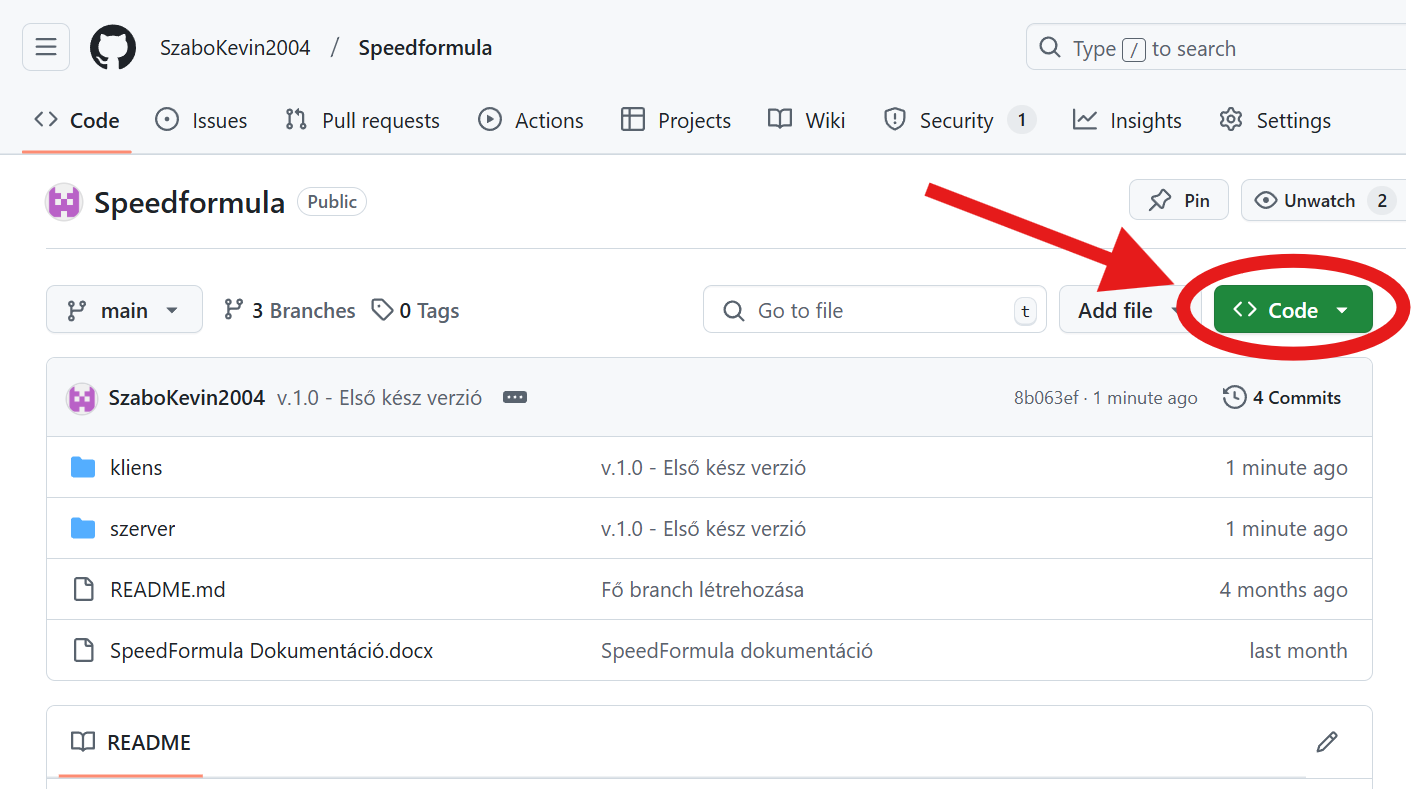
* Op. rendszer: Windows 10, 11
* Processzor: Intel Core i5-8400 / AMD Ryzen 5 3400G
* RAM: 8 GB
* Tárhely: 25 GB
* Hálózat: Széles sávú internetkapcsolat
* Eszközök:
  + Node.js 18.x
  + npm 8.x
  + Angular CLI 19.x
  + Web szerver: Apache
  + Adatbázis: MySQL
  + Verziókezelő: Git

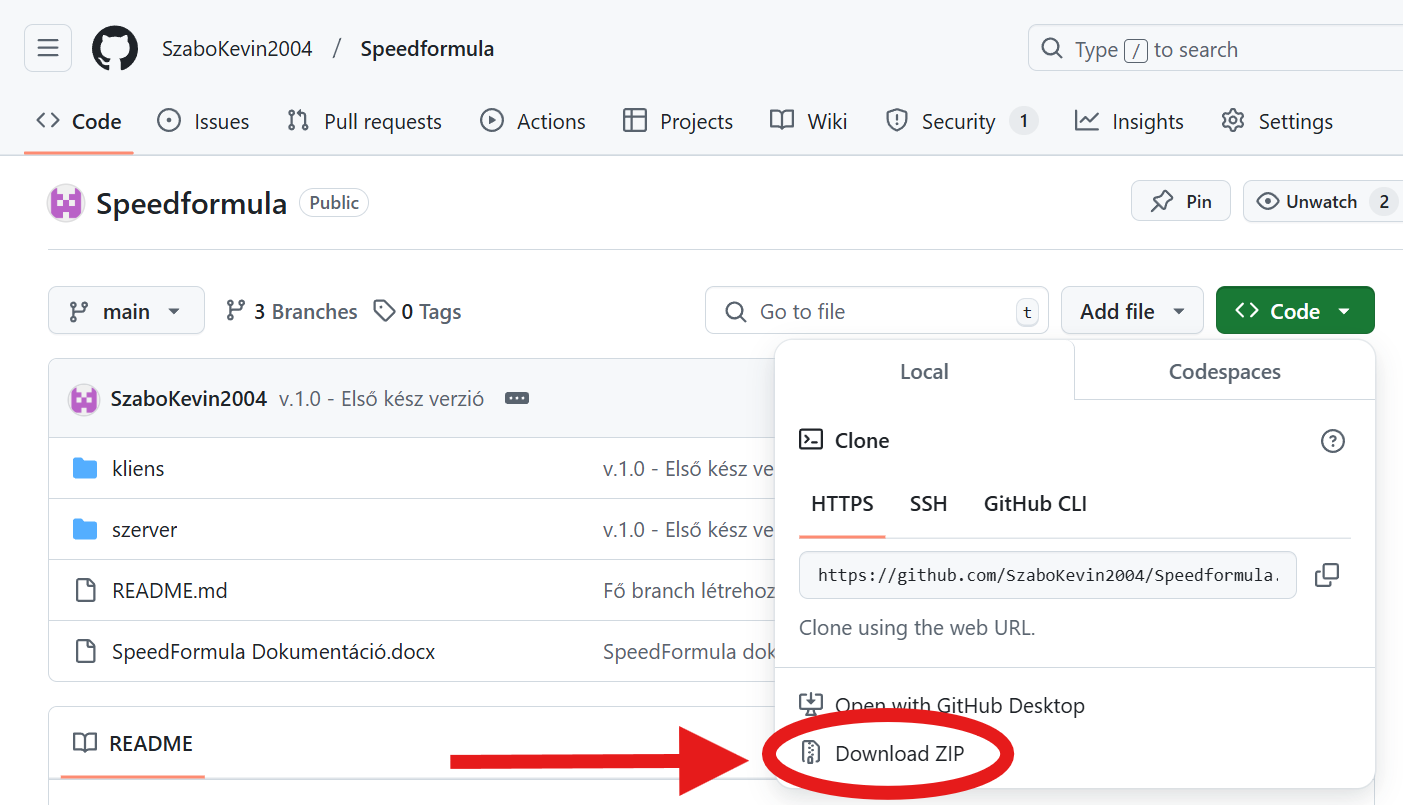
## Telepítési és futtatási útmutató

A weboldal fejlesztéséhez és futtatásához szükséges környezet előkészítése és futtatása.

### Telepítés menete

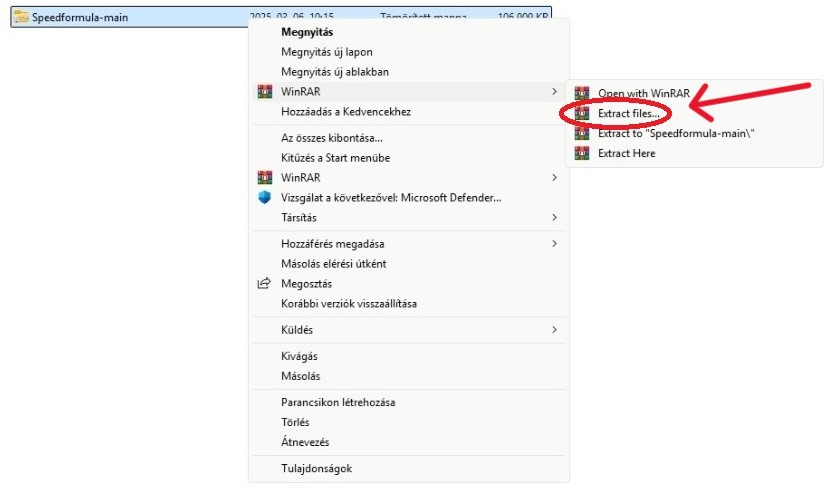
1. Lépjen fel a Github SpeedFormula projektünkre az alábbi linken:  
   <https://github.com/SzaboKevin2004/Speedformula>  
   Majd kattintson a < > Code gombra.

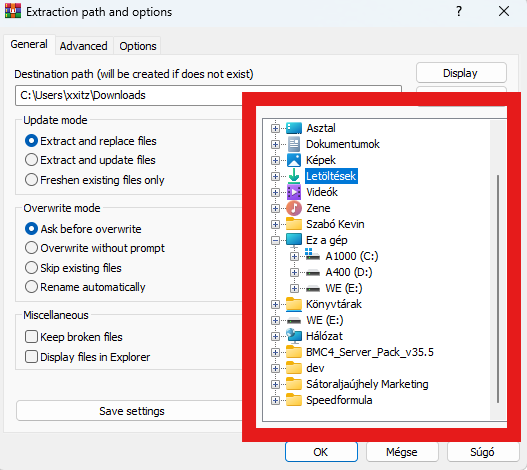
*1.kép*

1. Az így kapott legördülő sávból kattintson a Download ZIP-re.

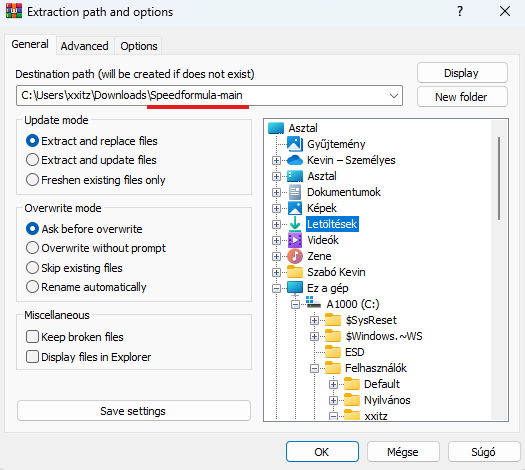
*2.kép*

1. A letöltött fájlt „SpeedFormula-main” néven találja.  
   A kicsomagoláshoz kattintson jobb-klikk-el rá, majd válassza ki a WinRAR-t a menüpontokból, azon belül pedig kattintson az Extract files… -ra / Fájlok kicsomagolása… -ra.

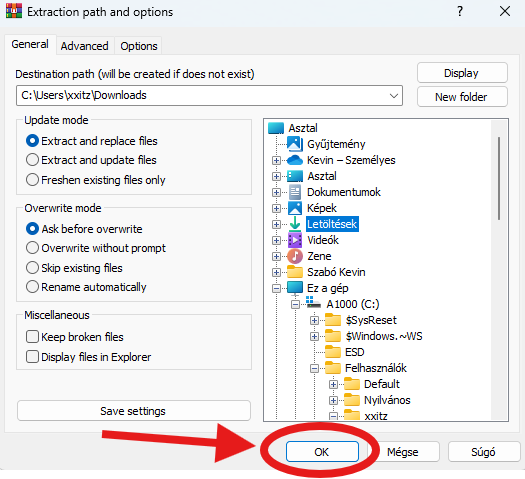
*3.kép*

1.  Alapértelmezetten oda csomagolja ki a tömörített mappát, ahova telepítette azt. A kicsomagolás helyét módosítani lehet a jobb oldalon megtalálható gördülő sávban.

*4.kép*

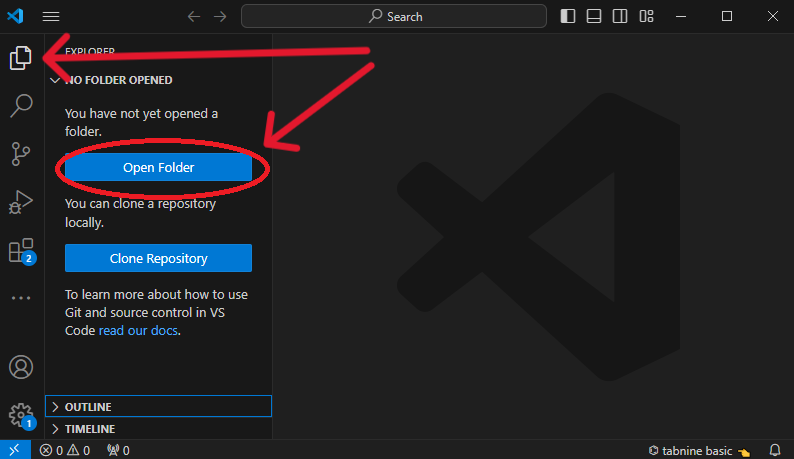
1. Ügyeljen arra, hogy a megjelenő elérési útvonal mezőben amennyiben tartalmazza a mappa nevét „Speedformula-main” például: C:\\Users\Felhasználó\Downloads\Speedformula-main  
   Akkor törölje ki belőle a mappa nevet! A példa esetében így már csak C:\\Users\Felhasználó\Downloads - kell, hogy maradjon! (Erre azért van szükség, mert, amit kicsomagol, az már tartalmaz egy Speedformula-main mappát)

*5.kép*

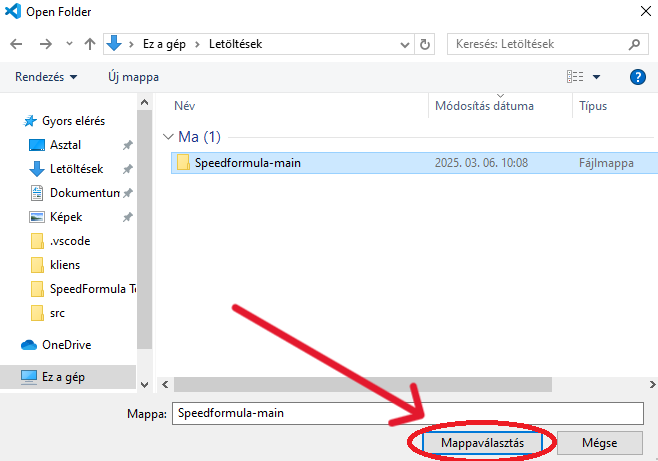
1. A kicsomagolás véglegesítéséhez kattintson az „OK” –gombra.

*6.kép*

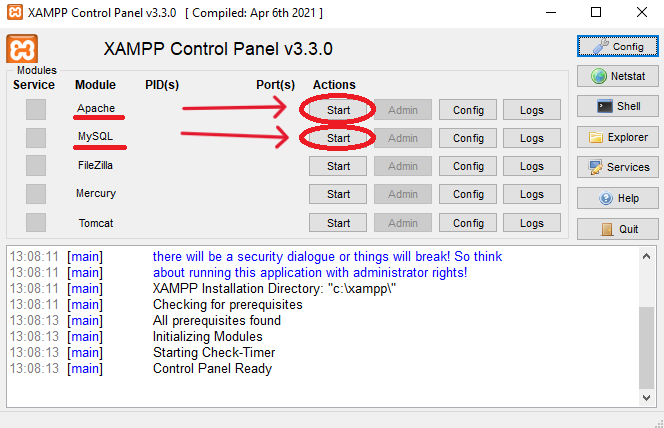
### Környezet előkészítés menete

1. Nyissa meg a Visual Studio Code alkalmazást. Menjen az Explorer fülre és kattintson az „Open Folder” gombra!

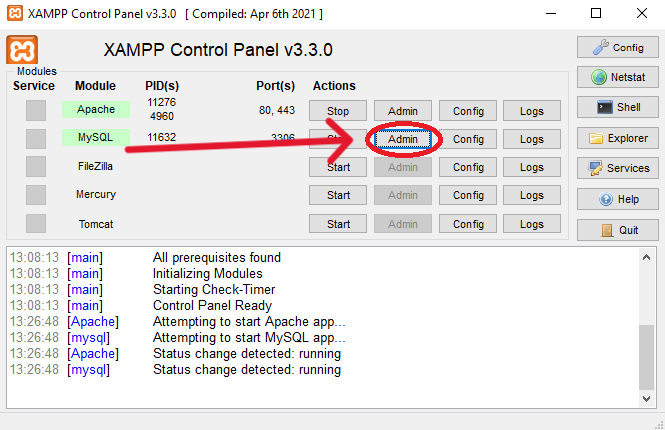
*7.kép*

1. Keresse meg a tallózási ablakban a kicsomagolt „Speedformula-main” mappát, majd kattintson a „Mappaválasztás” gombra!

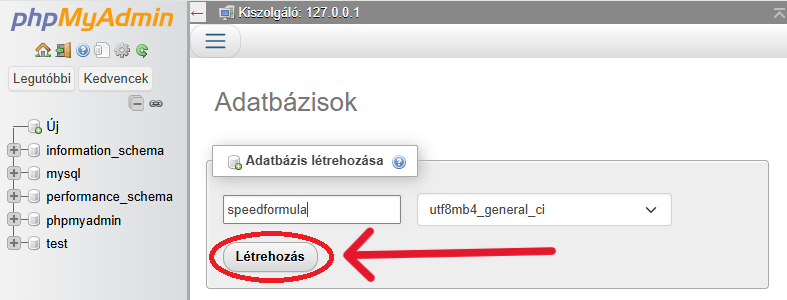
*8.kép*

1. Nyissa meg a XAMPP Control Panel-t, majd a „Start” - gombbal indítsa el a következő 2 modult: Apache, MySQL

*9.kép*

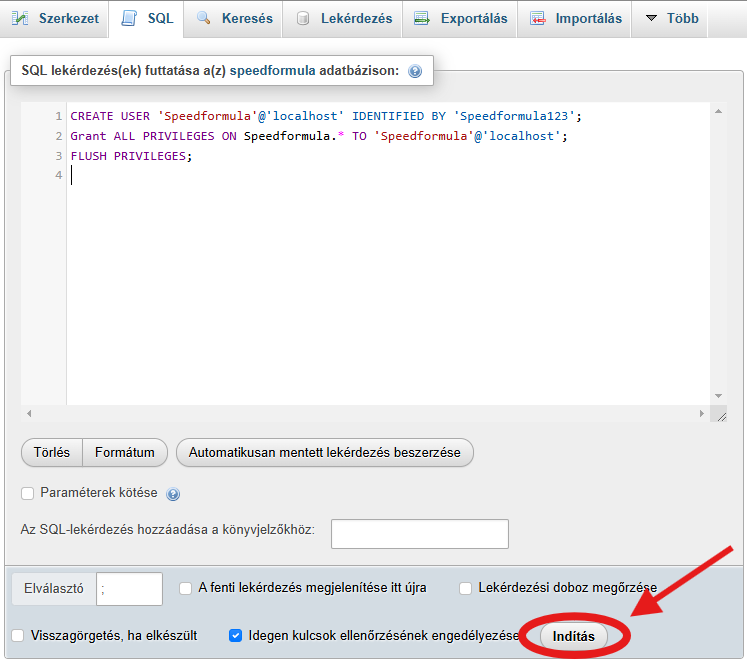
1. Kattintson a MySQL modul: „Admin” – gombra!

*10.kép*

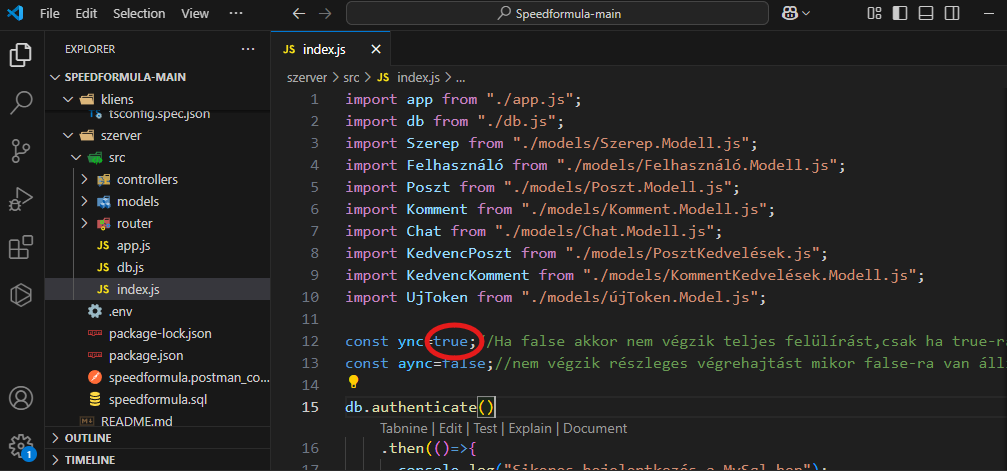
1. Hozzon létre új adatbázist bal oldali sávban az „Új” gombra kattintva, majd Adatbázis névnek adja meg „speedformula”, karakterkészletet hagyja alapértelmezetten (utf8mb4\_general\_ci), végezetül kattintson a „Létrehozás” gombra!

*11.kép*

1. Navigáljon a fenti sávban az „SQL” fülre, majd másolja át ide a következőeket:

CREATE USER 'Speedformula'@'localhost' IDENTIFIED BY 'Speedformula123';  
Grant ALL PRIVILEGES ON Speedformula.\* TO 'Speedformula'@'localhost';  
FLUSH PRIVILEGES;  
Ezt megtalálja a következő fájlban is: Speedformula-main\szerver\src\db.js  
Majd kattintson az „Indítás” gombra!

*12.kép*

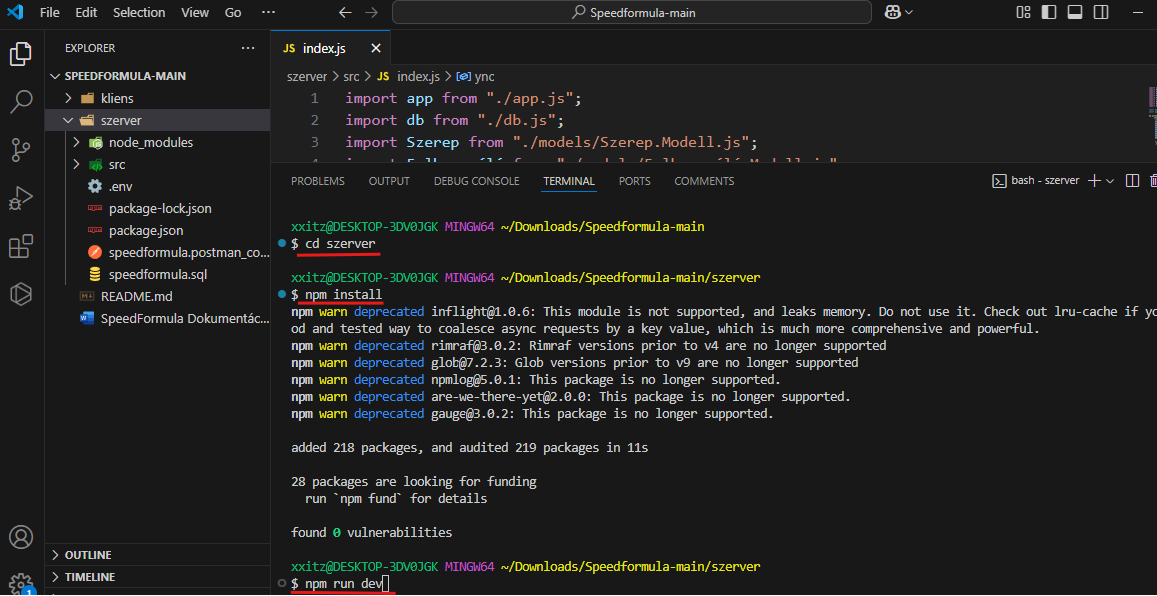
1. Visual Studio Code-ban nyissa meg a következő fájlt: Speedformula-main\szerver\src\index.js. A 12. sorban a const ync=false; false értékét írja át true-ra, majd mentse el. (ezzel az adatbázist feltölti a kellő táblákkal és kapcsolatokkal)

*13.kép*

1. A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

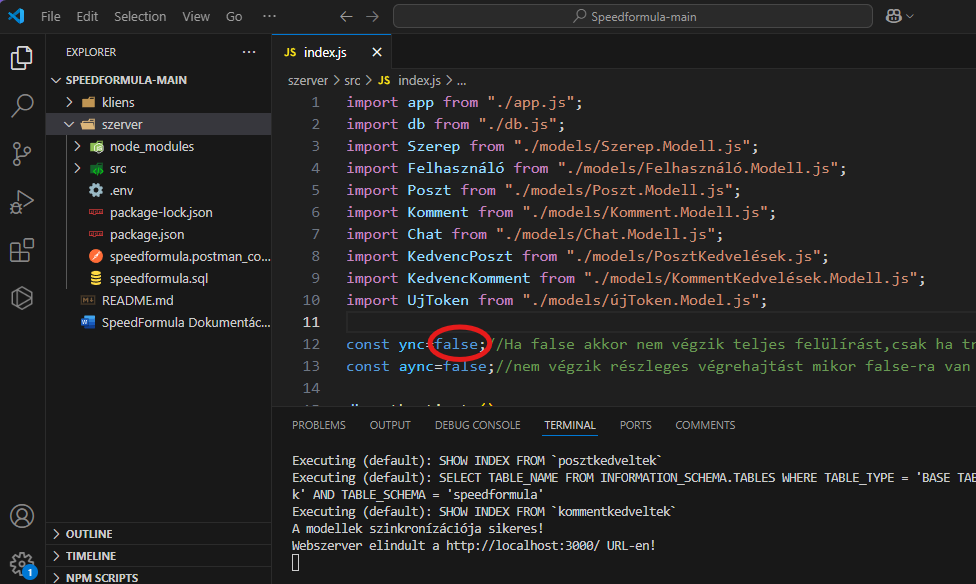
   AI-generated content may be incorrect.Válassza ki a felső sávban a 3 pöttyöt ( … ), majd vigye a kurzort a „Terminal” opcióra és kattintson a „New Terminal” -ra! / vagy nyomja meg a ctrl + shift + ö -t

*14.kép*

1. Az így kapott terminálban navigáljon a szerver mappába: cd szerver  
   majd telepítse a függőségeket: npm install  
   ezután indítsa el a szervert: npm run dev

*15.kép*

1. Amint elindult a szerver állítsa vissza a korábban átírt értéket „true”-ról „false”-ra és mentse el!

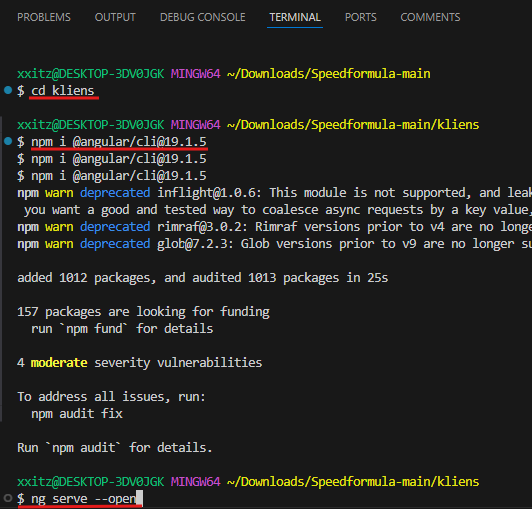
*16.kép*

1. A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

   AI-generated content may be incorrect.A korábban már említett módon nyisson egy új Terminált!

*17.kép*

1. navigáljon a kliens mappába: cd kliens  
   telepítse az angular 19-es csomagot: npm i @angular/cli@19.1.5  
   majd ha befejeződött a telepítés futtassa: ng serve –open

*18.kép*

Ha bármikor máskor futtatni és üzembe szeretné helyezni akkor az előbb felsorolt lépésekből a következőkre lesz szüksége:

Adatbázis elindítás: Xampp Control Panel-ben Apache és MySQL elindítása  
Szerver elindítás: Visual Studio Code Terminálban: cd szerver, npm start  
Kliens elindítás: Visual Studio Code Terminálban: cd kliens, ng serve --open

## 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 cserélését.

Lehetőség van téma módosításra bejelentkezett fiókkal.

Opció bemutatkozás írására (ami a fórum profilon jelenik meg).

1. Élő adás chat

Élő adáshoz chat szolgáltatás, amelyet bárki megtekinthet.

Bejelentkezett fiókkal lehetőség van üzenetküldésre.

1. Fórum

Poszt létrehozás.

Poszt kedvelés/komment kedvelés.  
Komment létrehozás posztra/kommentre.

## 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.

## 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)

## Frontend komponens dokumentációja

A dokumentáció a frontend alkalmazás részeit tartalmazza, ami a megjelenésért és funkció kezelésekért felelős. Ezen elemek, fájlok összessége megtalálható az: **Speedformula-main/kliens/src** mappán belül.

### Routing működése, frontend végpontok felépítése

A frontend végpontok kezelésének megvalósítása az **app/app.routes.ts** fájlban történik és az @angular/router modul segítségével konfiguráljuk.

Itt megtalálható az export const routes: Routes objektum, amely az alkalmazás különböző komponenseihez vezető útvonalakat definiálja.

path: Ez mondja meg hogy milyen végponton legyen elérhető frontenden az adott komponens.

component: Azt a komponens-t tartalmazza, ami a végponthoz fűződik.

Egyes útvonalakhoz hozzá van rendelve az authGuard, amely biztosítja, hogy csak hitelesített, azaz csak bejelentkezett felhasználó legyen képes elérni bizonyos útvonalakat, illetve bizonyos útvonalakat ne tudjon elérni a bejelentkezett felhasználó.

Alap útvonalak:

**’’ (gyökér útvonal)** – A kezdőlap (KezdolapComponent), az alkalmazás induló oldala.

**regisztracio** – A regisztrációs oldal (RegisztracioComponent), itt lehet új fiókot létrehozni.

**bejelentkezes** – A bejelentkezési oldal (BejelentkezesComponent), itt lehet bejelentkezni

**beallitasok** – Felhasználói beállítások oldal (BeallitasokComponent)

**visszajelzes** – A felhasználók visszajelzések küldését biztosító oldal (VisszajelzesComponent)

**rolunk** – Az oldal készítőiről szóló oldal (RolunkComponent)

**gyik** – Gyakran ismételt kérdések oldal (GyikComponent)

**adatvedelem** – Az adatvédelmi nyilatkozatot tartalmazó oldal (AdatvedelemComponent)

**elo** – Élő közvetítést megjelenítő és chatet biztosító oldal (EloComponent)

**hirek** – Hírek és friss információk oldala (HirekComponent)

**versenyzok** – Versenyzők listázása API-ból (VersenyzokComponent) (fejlesztés alatt)

**versenyzo** – Kiválasztott versenyzőről szóló adatok (VersenyzoComponent) (fejlesztés alatt)

**csapatok** – Csapatok listázása API-ból (CsapatokComponent) (fejlesztés alatt)

**csapat** – Kiválasztott csapatról szóló adatok (CsapatComponent) (fejlesztés alatt)

**tortenet** – Forma1 történeti áttekintő oldal (TortenetComponent)

**forum** – Fórum kezdőlap (ForumComponent)

**forum-poszt-letrehozas** – Fórum poszt létrehozását szolgáló oldal (ForumPosztComponent)

**forum/:id** – Egy adott fórum bejegyzés megtekintése (ForumPosztReszletComponent)

**forum/profil/:felhasznalonev** – A felhasználó profiljának megtekintése (ProfilComponent)

**\*\*** - Hibakezelő oldal (HibaComponent), amely akkor jelenik meg, ha a felhasználó egy nem létező útvonalra navigál.

A felsoroltak közül az alábbiak védett útvonalak:

beallitasok, forum, forum-poszt-letrehozas, forum/:id, forum/profil/:felhasznalonev

A regisztracio és bejelentkezes oldalaknál a felhasználó már be van jelentkezve, ezért automatikusan átirányítja őt a kezdőlapra.

Az authGuard megtalálható az **app/auth.guard.ts** fájlban. Ez egy CanActivateFn típusú függvény, mely az útvonalak elérhetőségét szabályozza.

A működési logikája:

const hitelesitettE = authService.hitelesitettE();

Az authGuard az AuthService hitelesitettE() metódusát használja hogy ellenőrizni tudja, hogy a felhasználó be van-e jelentkezve, vagy sem.

  if (hitelesitettE && (state.url === '/regisztracio' || state.url === '/bejelentkezes')) {

    router.navigate(['/']);

    return false;

  }

Korábban említettekben:   
ha a felhasználó bejelentkezve próbál a regisztracio vagy a bejelentkezes oldalra menni, akkor átirányítja őt automatikusan a kezdőlapra.

  const vedettUtvonalak = ['/beallitasok', '/forum', '/forum-poszt-letrehozas'];

  if (!hitelesitettE && (vedettUtvonalak.includes(state.url) || state.url.startsWith('/forum/'))) {

    router.navigate(['/bejelentkezes']);

    return false;

  }

Ha a felhasználó nincs bejelentkezve és úgy próbál meg egy védett oldalra lépni, akkor automatikusan a rendszer átirányítja őt a bejelentkezési oldalra.

### Hitelesítés, API kommunikáció és Adatkezelés

Az alkalmazásban a backend kapcsolatok, API hívások és logikájuk az Angular szolgáltatásfájlokban (service-ekben) kerülnek kezelésre. Ezek a fájlok kezeskednek az adatok eléréséért, feldolgozásáért és tárolásukért, illetve külső API-kkal való kommunikációért. Ezek a service osztályok .service.ts fájlokban lettek megvalósítva, amik a **app/services** mappában találhatóak a könnyebb átláthatóság érdekében. Működésük biztosítják, hogy az alkalmazás különböző részei könnyen hozzáférhessenek az adatokhoz és funkcionalitásokhoz.

#### auth.service.ts:

Az AuthService felelős a felhasználók hitelesítéséért és jogosultságkezeléséért. Az alábbi funkciókat biztosítja:

* Regisztráció és bejelentkezés
* Token kezelés és automatikus frissítés
* Profiladatok kezelése (név, profilkép, téma, szerepkör)
* Kijelentkezés és felhasználói adatok törlése

Használt eszközei és könyvtárai:

@Injectable({ providedIn: 'root'}) – Globálisan elérhetővé teszi a szolgáltatást.

HttpClient – http kérések küldését valósítja meg a backend API felé.

Router – Angolar routert használat navigáláshoz.

BehaviorSubject – Relatív állapotkezeléshez használtam a felhasználói adatok követésére.

tap() – Adatfolyam módosítás bejelentkezéskor.

switchMap() – Token automatikus frissítése időzített ciklusban.

timer() – Token frissítés időzítéshez.

interface AuthResponse {

token: string;

pfp: string;

username: string;

tema: number;

szerep: number;

}

AuthResponse interfész tartalmazza a felhasználó hitelesítési válaszát, amennyiben érkezik egy sikeres bejelentkezés során.

private url = 'http://localhost:3000';

Backend API végpontját tárolja az url.

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

  felhBejelentkezettE$ = this.felhBejelentkezettE.asObservable();

A felhasználó bejelentkezési állapotát tárolja. Kezdetben ez hamis érték.

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

  randomKep$ = this.randomKep.asObservable();

Felhasználó profilkép útvonalának tárolása.

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

  felhasznaloNev$ = this.felhasznaloNev.asObservable();

A bejelentkezett felhasználó nevét tárolja.

  private szamSzin = new BehaviorSubject<number>(2);

  szamSzin$ = this.szamSzin.asObservable();

A felhasználó által választott témaazonosító.  
1 = sötét, 2 = világos, 3 = vörös. Kezdetben

  private szerep = new BehaviorSubject<number>(2);

  szerep$ = this.szerep.asObservable();

A felhasználó szerepkörének azonosítóját tárolja.  
1 = admin, 2 = normál felhasználó. Kezdetben normál felhasználói értéke van.

private tokenFrissitoSub: Subscription | null = null;

Az automatikus token frissítés időzítés kezeléséért felelős.

  constructor(private http: HttpClient, private router: Router) {

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

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

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

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

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

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

      if (token) {

        this.felhBejelentkezettE.next(true);

      }

      if (pfp) {

        this.randomKep.next(pfp);

      }

      if (username) {

        this.felhasznaloNev.next(username);

      }

      if (tema) {

        this.szamSzin.next(parseInt(tema));

      }

      if (szerep) {

        this.szerep.next(parseInt(szerep));

      }

    }

    this.autoTokenFrissites();

  }

A konstruktor inicializálja az állapotokat és ellenőrzi, hogy van-e mentett bejelentkezési adat, amit a localStorage-ból kérdez le a localStorage.getItem-el.

* Ha van elmentett token, akkor a felhasználó bejelentkezett állapotba kerül.
* Beállítja a felhasználó profilképét, nevét, témáját és szerepkörét.
* Meghívja a token automatikus frissítését.

setToken(token: string) {

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

localStorage.setItem('token', token);

}

}

Ez és ehhez hasonló setter metódusok beállítják a felhasználók adatait, mely lementődik localStorage.setItem segítségével a böngésző helyi tárolójába a megadott kulcsszóval. Ebben a kódban a hitelesítési tokent állítja be, amennyiben létezik, azaz bejelentkezett a felhasználó. Ellenőrzi hogy a window és a localStorage elérhetőek-e (ami azt jelenti hogy böngésző környezetben futtatják)-e, ha igen csak azután menti helyi tárolóba a bejelentkezett felhasználó adatot. Értelemszerűen több helyen is felhasználtam ezt a kódrészletet, ami ellenőrzi, hogy böngésző környezetben van-e, emiatt nem írom ki mindenhez a magyarázatát.

  regisztracio(regisztracioData: {

    felhasznalonev: string,

    email: string,password: string,

    confirm\_password: string,

  }) {

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

      catchError((error) => {

        return throwError(() => error);

      })

    );

  }

A regisztracio metódus egy HTTP POST kérést küld a /regist végpontra a backendnek, mely tartalmazza a regisztrációs adatokat melyet a felhasználótól kap meg a regisztracio komponensből regisztráció esetén. Mivel observable-t ad vissza, ezért pipe() segítségével elvégezhetünk rajta műveleteket. catchError() operátorral kezelni tudjuk a felmerülő hibákat pl hálózati hiba, szerver válasza hibaüzenetet tartalmaz stb. a catchError úgy mondd „elkapja” ezt a hibát. A throwError() segítségével pedig a hibát újra kiadatja, így az alkalmazás más részein is elérhetővé válik ez a hiba és lehetőség lesz rá reagálni. Ezt értelemszerűen minden HTTP kérésnél használtam.

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

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

      tap((response) => {

        this.setTheme(response.tema);

        this.setToken(response.token);

        this.setPfpId(response.pfp);

        this.setFelhasznaloNev(response.username);

        this.setSzerep(response.szerep)

        this.setBejelentkezettE(true);

      }),

      catchError((error) => {

        console.error('Hiba történt a bejelentkezés során:', error);

        return throwError(() => error);

      })

    );

  }

HTTP Post kérést küld a /login végponton backend felé, hogy a felhasználó bejelentkezzen.

Stringként, azaz szövegként küldi a következőket: felhasznalonev, email, password. Ezek azok az adatok, amit majd megvizsgál a backend, hogy egyezik-e az adatbázisban a tároltakkal. (Az adatokat a bejelentkezes komponens továbbítja a felhasználótól az auth servicebe). tap() operátor, ami akkor fut le ha a válasz sikeres, jelen esetben akkor ha a bejelentkezés sikeres. A response tartalmazza a választ, ami visszaérkezik a backendtől és a meghívott setter metódusokba elhelyezve állítjuk be az adatokat. Pl. this.setTheme(response.tema) ez a válaszból érkezett téma szám értékét állítja be. Értelemszerűen a tap() operátort több helyen is felhasználtam, nem csak ennél a HTTP kérésnél.

  autoTokenFrissites() {

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

      if (!localStorage.getItem('token')) {

        console.warn('Nincs token, nem indul az automatikus frissítés.');

        return;

      }

      this.tokenFrissitoSub = timer(0, 12 \* 60 \* 60 \* 1000)

        .pipe(

          switchMap(() => this.http.post<{ accessToken: string }>(`${this.url}/ujToken`, {}, {

            headers: { 'Authorization': `Bearer ${localStorage.getItem('token')}` }

          }).pipe(

            catchError(err => {

              console.error('Token frissítés sikertelen:', err);

              this.kijelentkezes();

              return throwError(() => err);

            })

          ))

        )

        .subscribe({

          next: (response) => {

            this.setToken(response.accessToken);

            console.log('🔄 Token automatikusan frissítve.');

          }

        });

    }

  }

Ez a metódus felelős azért, hogy automatikusan új hozzáférési tokent kapjon/kérjen a frontend a backendtől. Mivel a token 12 óránként lejár, ezért szükséges az új token igénylése, ahhoz, hogy ne legyen kiléptetve a felhasználó mindig 12 óra lejárta után.

Megvizsgálja, hogy a token létezik-e, ha nem kiír egy figyelmeztetést konzolra és nem hajtja végre a token frissítést, ha létezik akkor tovább fut. Ha a token létezik akkor a timer operátorral ütemezi a frissítést: timer(0, 12 \* 60 \* 60 \* 1000), Az első paraméter (0) azt jelzi hogy azonnal elindítsa az első frissítést. Egyrészt ezáltal jobb ütemet biztosít a frissítés idejének, másrészt nélkülönözhetetlen egy hiba elkerülésére. A hiba abból adódik, hogy minden egyes oldal frissítésnél újraindul ez a számoló, tehát ha folyamat frissítené a felhasználó az oldalt akkor soha nem jutna el a token frissítésig és lejárna (mindig újból kezdené a 12 órás számlálást), viszont ezzel minden oldal frissítésnél lefut 1x és új tokent kap. A második paraméter biztosítja, hogy 12 óránként frissítse. (Azért kell szorozni 1000-rel, mert az operátor milliszekundumban vizsgál és ez a váltószám hozzá)

switchMap használata garantálja azt, hogy folyamatosan ismétlődjön az új hitelesítési token generálás kérése. Minden alkalommal, mikor a timer egy új értéket kiad (minden 12 órában) a switchMap operátor újra lefut és egy új HTTP Post kerül küldésre /ujToken végpontra. A HTTP Post kérésnél Authorization fejlécben a már meglevő token-t küldi el Bearer típusú hitelesítéssel, amit a böngésző helyi tárolójából olvas ki a localStorage.getItem(’token’) -el. Ez a művelet azért is jó mert ezzel megtudja mondani, hogy melyik felhasználónál kell kicserélni a token-t a meglevő token-re hivatkozva a backendnél, illetve azt is biztosítja, hogy csak bejelentkezett felhasználók tudják az adott műveleteket megtenni. Ezt a megoldást több kódrészletben is használom. Végsősoron a HTTP kérés válaszát subscribe() metódus segítségével vizsgáljuk meg, majd ha küld adatokat akkor a next: callback függvény fut le és itt adódnak át a backendtől kapott adatok. Ebben a kódban az új token érkezik meg accessToken néven. (Ezt a kódrészt is használtam több helyen).

  profilLekeres() {

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

    if (!token) {

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

    }

    return this.http.get<any>(`${this.url}/profil`, {

      headers: {

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

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

      }

    }).pipe(

      tap((response) => {

        if (response.felhasználó.kep) {

          this.randomKep.next(response.felhasználó.kep);

          localStorage.setItem('pfp', response.felhasználó.kep);

        }

      }),

      catchError((error) => throwError(() => error))

    );

  }

Profil adatok lekérését végezzük HTTP GET kéréssel és Authorization Bearer típusú hitelesítéssel lehetővé tesszük hogy azonosítsuk a felhasználót küldött token alapján.

Ha a válaszban található profilkép URL akkor hívással frissítjük a BehaviorSubject-et, ami a profilképet tárolja, majd elmentjük a profilképet helyi tárolóba.

  profilModositas(

    felhasznalonev: string | undefined = undefined,

    email: string | undefined = undefined,

    password: string | undefined = undefined,

    tema\_id: number | undefined = undefined,

    kep: number | undefined = undefined,

    magamrol: string | 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, kep, magamrol },

      {

        headers: {

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

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

        }

      }).pipe(

        catchError((error) => {

          return throwError(() => error);

        })

      );

  }

Profil adatok módosítását végezzük HTTP PATCH kérés küldésével az API / profil végpontjára a megadott adatokkal a beallitasok komponensből. Az adatokat csak akkor küldi, ha ténylegesen megadta azt a felhasználó, különben undefined értéket küld. Az adatok, amik küldésre kerülnek:

* felhasznalonev: Új felhasználónév
* email: Új email-cím
* password: Új jelszó
* tema\_id: Módosított téma id (ha másik témát akar beállítani, használni a felhasználó)
* kep: Új EGYEDILEG feltöltött profilkép (Fejlesztés alatt)
* magamrol: A felhasználó új bemutatkozó szövege

kepModositas() – HTTP PATCH kérést küld a profil/profilkep végpontra, majd módosítja a profilképet egy random kiválasztott profilképre az alapértelmezett profilképek közül.

profilTorles() – HTTP DELETE kérést küld a /profil végpontra, ami ide a beallitasok komponensből továbbítódik. Authorization Bearer típussal eltárolt tokennel. A backend törli a felhasználót a megadott token-nel.

  profilMasikTorles(felhasznalonev: string){

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

    if (!token) {

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

    }

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

      headers: {

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

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

      }

    }).pipe(

      catchError((error) => {

        return throwError(() => error);

      })

    );

  }

HTTP DELETE kérést küld a /profil/${felhasznalonev} végpontra Authorization Bearer típussal eltárolt tokennel. A Felhasználónévhez a kapott felhasználónevet behelyettesíti és azt küldi el a backendnek paraméterben, majd a backend törli a felhasználót a megadott felhasználónévvel. Ez segít abban hogy admin felhasználóval képesek lehessünk törölni más felhasználókat is.

  kijelentkezes() {

    if (this.tokenFrissitoSub) {

      this.tokenFrissitoSub.unsubscribe();

      this.tokenFrissitoSub = null;

    }

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

      localStorage.clear();

    }

    this.felhBejelentkezettE.next(false);

    this.felhasznaloNev.next('');

    this.szamSzin.next(2);

    this.szerep.next(2);

    this.randomKep.next('');

    this.router.navigate(['/']).then(() => {

      window.location.reload();

    });

  }

A kijelentkezes() metódus felelős a felhasználó kijelentkeztetéséért és több olyan műveletet is végrehajt, amivel törli a bejelentkezett állapotot és adatokat.

Ha a token automatikus frissítése aktív, akkor a metódus leiratkozik az időzített tokenfrissítésről unsubscribe() hívással. Ezzel oldjuk meg azt a problémát, hogy a rendszer kijelentkezés után is próbálja frissíteni a token-t. Majd nullázza a tokenFrissitoSub -ot hogy jelezze hogy nincs aktív frissítés.

localStorage.clear() törli az összes helyi tárolóban eltárolt adatot.

Beállítja a bejelentkezési állapotot false-ra, ezzel tudja majd az alkalmazás, hogy a felhasználó kijelentkezett. Felhasználónevet töröl. Alapértelmezettre visszaállítja a témát és a szerepkört. Azaz a témát 2 = világosra, a szerepet 2 = normál felhasználóra (ha adminként lett volna belépve akkor 1-ről, 2-re váltana), majd törli a profilképet is, ez biztosítja hogy kijelentkezés után nem marad meg a régi kép.

Végezetül a főoldalra (/) átirányítja a felhasználót ( this.router.navigate[’’]) ), majd miután a navigáció megtörtént( .then(() =); ), az oldal teljesen újratöltődik ( window.location.reload() ) Ez azért is fontos, mert az újratöltés lehetővé teszi, hogy minden állapot teljesen frissüljön és az esetlegesen memóriában maradt adatok is eltűnjenek.

  hitelesitettE(): boolean {

    return !!localStorage.getItem('token');

  }

Ennek segítségével dönti el az alkalmazás, hogy a felhasználó be van-e jelentkezve.

Két féle lehetőséget biztosít, annak esetében, hogy true vagy false értéket ad vissza:  
1. UI vezérlés: true: Megjelennek a bejelentkezett felh. szánt elemek, ellenkező esetben nem. Pl. a bejelentkezés, regisztráció gomb ne jelenjen meg bejelentkezett felhasználóknak.  
2. Oldalak védelme: Megakadályozza, hogy bizonyos oldalakat megnyithasson a felhasználó, aki nem jelentkezett be/aki bejelentkezett. Ez korábban leírásra került a „Routing működése, frontend végpontok felépítése” pontban.

#### forum.service.ts

Ebben a service-ben kerülnek kezelésre a fórummal kapcsolatos API-hívások, amik a forum komponensben találhatóak.

Az alábbi fő funkciókat biztosítja:

* Posztok kezelése: lekérése, létrehozása, törlése, kedvelése
* Kommentek kezelése: lekérése, hozzáadása, törlése, kedvelése
* Felhasználói profil információk lekérése

Ami egységes és minden kérésre elmondható:

Az itt megtalálható összes kérés a forum komponensből kerül átadásra.

Mindegyik kéréshez a helyi tárolóból kapott tokent hozzáadja:  
const token = localStorage.getItem('token');

A tokent Authorization fejlécben, Bearer típusként küldi, ami backenden azt határozza meg hogy a felhasználó hitelesítve van, és jogosult az adatok lekérésére:  
const headers = new HttpHeaders().set('Authorization', `Bearer ${token}`);

  private url = 'http://localhost:3000/forum';

  private alapUrl = 'http://localhost:3000';

Backend végpontok meghatározása.  
Url: Kizárólagosan a fórum oldalhoz tartozó adatok végpontja  
alapUrl: Az alapértelmezett végpontok url-je. (Felhasználó profil adatok lekérése)

Posztok kezelése:

getPosts() – A szervertől az összes posztot lekéri /forum végponton, ha a felhasználó be van jelentkezve token-t küld a fejlécben.

getPostsId() – Egy adott felhasználó összes posztját lekéri /forum/${felhasznalonev} végponton. Ez a fórum profil oldal miatt szükséges, ahol minden felhasználó saját posztjai jelenítődnek meg.

createPost() – HTTP PUT kérés a szerver felé a /forum/cikk végponton új poszt létrehozására.

createImagePost() – Ha a poszt csak multimédiát (kép, videó) tartalmaz, akkor ez a metódus hajtódik végre. A kérés a /forum/kepes végponton történik. (Fejlesztés alatt, jelenleg nem elérhető).

deletePost() – HTTP DELETE kérés a szerver felé a /forum/poszttorles végponton egy adott poszt törlésére.

Kommentek kezelése:

addCommentToPost() – Egy adott poszthoz kommentet fűz a kérés a /forum/komment/poszt végponton történik.

AddCommentToComment() – Már létező kommentre komment fűzés a kérés a /forum/komment/komment végponton zajlik.

getComments() – Poszthoz az összes komment lekérdezése a /forum/komment/${posztid} végponton.

deleteComments() – HTTP DELETE kérés a /forum/komenttorles végponton, amellyel egy adott komment törölhető.

Kedvelések kezelése:

likePost() – Egy adott poszt kedvelése a kérés a /forum/posztkedveles/${postId} végponton történik.

dislikePost() – Egy adott poszt kikedvelése a kérés a /forum/posztkikedveles/${postId} végponton történik.

likeComment() – Egy adott komment kedvelése a kérés a /forum/posztkedveles/${commentId} végponton történik.

dislikeComment() – Egy adott komment kikedvelése a kérés a /forum/kommentkikedveles/${commentId} végponton történik.

Profilkezelés:

profilLekeresMasik() – Egy adott felhasználó profilját lekéri a szervertől, az adott lekérés a /profil/${felhasznalonev} végponton zajlik.

#### elo.service.ts

Ebben a service-ben hajtódnak végre az elo komponens-től származó kérések továbbítása a szerver felé, amely élő YouTube közvetítésekkel és csevegési üzenetekkel foglalkozik.

A következőket valósítja meg:

* Élő YouTube közvetítés adatainak lekérése
* Chatüzenetek küldése és lekérése
* Üzenetek törlése

getEloApi(): Observable<any> {

return this.http.get(`${this.url}/elo`).pipe(

tap((response: any) => {

this.apiKey = response.apikey;

this.channelId = response.channelId;

})

);

}

A szerverről lekéri a YouTube API kulcsot és a csatorna azonosítóját.

Az apiKey és channelId változókba menti az értékeket.

  getElo(): Observable<any> {

    if (!this.apiKey || !this.channelId) {

      return this.getEloApi().pipe(

        switchMap(() => {

          const url = `https://www.googleapis.com/youtube/v3/search?part=snippet&channelId=${this.channelId}&type=video&eventType=live&key=${this.apiKey}`;

          return this.http.get(url);

        })

      );

    } else {

      const url = `https://www.googleapis.com/youtube/v3/search?part=snippet&channelId=${this.channelId}&type=video&eventType=live&key=${this.apiKey}`;

      return this.http.get(url);

    }

  }

Ha nincs API kulcs vagy csatorna ID, akkor először lekéri a getEloApi() segítségével.

switchMap: Eredményt továbbviszi a YouTube API hívásra.

Végül elküldi a YouTube API-hoz a keresési kérést, amely az élő videókat adja vissza.

  getVideoUrl(videoId: string): string {

    return `https://www.youtube.com/embed/${videoId}?autoplay=1&modestbranding=1&controls=1&showinfo=0&rel=0&iv\_load\_policy=3&fs=1`;

  }

Beágyazott YouTube lejátszó URL-t generál a kapott videoId alapján.

Automatikusan elindul és minimalizált „brandinget” használ, irányítókat megjeleníti mint pl kinagyítás, hangszabályozók stb.

sendChatMessage(uzenet: string) – üzenet küldés

Hitelesítés szükséges. Csak bejelentkezett felhasználó küldhet üzenetet (Bearer token)

HTTP POST kérés küldés a szerver chat végpontjára az üzenettel.

Hiba esetén továbbra is catchError-t használva feldob egy hibát.

Sikeres küldés után automatikusan frissíti az üzenetek listáját tap() => getMessages()

getMessages() – üzenetek lekérése

Lekéri az összes üzenetet a szerverről.

Ha van token, hitelesítési fejlécet is küld.

Ha sikeres a válasz tap(), frissíti a BehaviorSubject-et, így az uzenetek$ Observable frissül.

deleteMessages() – üzenetek törlése

Hitelesített felhasználók törölhetik az üzeneteket.

HTTP DELETE kérést küld a szerver chat végpontjára.

Sikeres törlés után a tap(), kiüríti a BehaviorSubject-et, így az üzenetek eltűnnek a UI-ból.

#### api.service.ts

Ebben a service-ben kerülnek kezelésre azok az API hívások, amelyek külső forrásból érkeznek a backend felé.

Kizárólag HTTP GET kéréseket tartalmaz ez a service.

getNews() – Hírek lekérése a szervertől a /hirek végponton. Ezeket a kéréseket a hirek komponensben kerülnek felhasználásra.

getTeams() – Csapatokról szóló adatok lekérése a szervertől a /csapatok végponton. (Fejlesztés alatt, jelenleg nincs funkciója).

#### visszajelzes.service.ts

Ebben a service-ben a visszajelzes komponens-től származó kérés kerül továbbításra a szerver felé.

visszajelzes() – HTTP POST kérés a /visszajelzes végpontra, melyben továbbítódnak a backend felé az alábbiak:

* Felhasználó email-címe
* Felhasználó által írt üzenet

ezen adatok feldolgozásra kerülnek a backendben, majd a speedformula email-címre továbbítódnak.

### Elkészített komponensek és működésük

## Backend komponens dokumentációja

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.

# Felhasználói dokumentáció

## Rendszerkövetelmény

Minimum:

* Op. rendszer: Windows 7
* Processzor: Intel Core i3-6100 / AMD Ryzen 3 1200
* 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

## Használati útmutató

# Tesztelés

## Frontend

## Backend

## Terheléses teszt

# Továbbfejlesztési lehetőségek

Az alábbiakban összegyűjtöttük a weboldal jövőbeli fejlesztési lehetőségeit, amik segíthetik a felhasználói élményt és teljesítmény növelést.

1. Funkcionalitás bővítése

* Kép, videó feltöltés: Poszt készítésnél tudjon képeket vagy videót is feltölteni a felhasználó.
* Kategória hozzáadás: Poszt készítésnél tudjon kategóriákat megadni, amik a poszthoz köthetőek.
* Szűrés és keresés opció: Képes legyen a felhasználó szűrni, illetve keresni a posztok között megadott kategóriák alapján.
* Követés/feliratkozás: A felhasználók képesek legyenek más felhasználókra feliratkozni/követni.
* Mentés funkció: Posztok saját felhasználójába mentése / mentés törlése
* Felhasználó kommentjeinek mutatása: Fórum profil oldalán minden felhasználónak lehessen megtekinteni a kommentjeit.
* Egyéni profilkép: Tudjon saját profilképet is feltölteni, ne csak a megadottakból kapni egyet.

1. Teljesítmény és biztonság növelése

* Gyorsítótárazás és képtömörítés: A betöltési sebesség javítása miatt.
* HTTPS-re való átállás: A biztonságos kapcsolat biztosítása és a keresőoptimalizálás érdekében.
* WebSocket támogatás: A valós idejű adatátvitel és interaktív funkciók hatékonyabb kiszolgálása érdekében.
* Támogatás a legújabb böngészőtechnológiákhoz: Az új szabványok és funkciók implementálása.

1. További integrációs lehetőségek

* Versenyzők: Versenyző adatokat tartalmazó API külső forrásból.
* Csapatok: Csapat adatokat tartalmazó API külső forrásból.
* Telemetria: Telemetria adatokat tartalmazó API külső forrásból

# Irodalomjegyzék

1. Angular hivatalos dokumentáció

url: <https://angular.io/docs>

Leírás: Az angular keretrendszer hivatalos dokumentációja

felhasználva: Frontend

idő: 2024.11.25 11:55

1. YouTube (videó letöltés)

url: <https://www.youtube.com/watch?v=KMzB_7xM2_c>

Leírás: Kezdőlap videója

felhasználva: kezdolap komponens

idő: 2024.11.26 15:13

1. RedBull (f1 gyakori kérdések)

url: <https://www.redbull.com/hu-hu/f1-gyakori-kerdesek-amiket-nem-mersz-feltenni>

Leírás: Forma 1-gyel kapcsolatos gyakori kérdések

felhasználva: gyik komponensben

idő: 2024.12.18 14:45

1. Stack Overflow

url: <https://stackoverflow.com/>

Leírás: Fejlesztési problémák megoldásához használt közösségi fórum

felhasználva: Frontend, Backend

idő: 2024.12.19 8:53

1. Wikipédia Formula-1

url: <https://hu.wikipedia.org/wiki/Formula%E2%80%931>

Leírás: Wikipédia Forma-1-ről adott leírása.

felhasználva: tortenet komponensben

idő: 2025.01.05 17:32

1. NewsAPI

url: <https://newsapi.org/>

Leírás: Hírszerző API hírforrásokból és blogokból

felhasználva: hir komponens

idő: 2025.01.10 17:38

1. Google Cloud APIs

url: <https://cloud.google.com/apis?hl=hu>

Leírás: Google api szolgáltatás (Youtube api v3 felhasználva)

felhasználva: elo komponens

idő: 2025.01.30 9:48