**EÖTVÖS LORÁND TUDOMÁNYEGYETEM**



Informatikai Kar

Programozási Nyelvek és Fordítóprogramok Tanszék

**Közösségi zeneajánló platform**

Témavezető: Szerző:  
**Pataki Norbert Lakosi Bence**Docens, PhD Programtervező informatikus BSc.

Budapest, 2023

Tartalomjegyzék

[1. Bevezetés 4](#_Toc136207298)

[1.1 Témaválasztás 4](#_Toc136207299)

[1.2 Spotify 4](#_Toc136207300)

[2. Alapvető technológiák 6](#_Toc136207301)

[2.1 React 6](#_Toc136207302)

[2.2 Next.js 6](#_Toc136207303)

[2.3 Prisma 7](#_Toc136207304)

[2.4 tRPC 7](#_Toc136207305)

[2.5 NextAuth.js 7](#_Toc136207306)

[2.6 TailwindCSS 8](#_Toc136207307)

[2.7 Spotify Web Api 8](#_Toc136207308)

[3. Felhasználói dokumentáció 9](#_Toc136207309)

[3.1 Rövid ismertetés 9](#_Toc136207310)

[3.2 Célközönség 9](#_Toc136207311)

[3.3 Rendszerkövetelmények 9](#_Toc136207312)

[3.4 Használat 10](#_Toc136207313)

[3.4.1 Bejelentkezés 10](#_Toc136207314)

[3.4.2 Főoldal 12](#_Toc136207315)

[3.4.2.1 Bal oldali sáv 13](#_Toc136207316)

[3.4.2.2 Jobb oldali sáv 14](#_Toc136207317)

[3.4.2.3 Információs doboz 14](#_Toc136207318)

[3.4.2.4 Központi elem 16](#_Toc136207319)

[3.4.2.5 Lejátszási vezérlő 18](#_Toc136207320)

[4. Fejlesztői dokumentáció 21](#_Toc136207321)

[4.1 Tervezés és célkitűzés 21](#_Toc136207322)

[4.2 Az alkalmazás telepítése 21](#_Toc136207323)

[4.2.1 Node.js és npm 21](#_Toc136207324)

[4.2.2 Függőségek telepítse 22](#_Toc136207325)

[4.2.3 Spotify for Developers 24](#_Toc136207326)

[4.2.4 Adatbázis 26](#_Toc136207327)

[4.2.5 Környezeti változók 26](#_Toc136207328)

[4.3 Alkalmazás futtatása helyileg 27](#_Toc136207329)

[4.4 Alkalmazás publikálása 28](#_Toc136207330)

[4.4.1 GitHub 28](#_Toc136207331)

[4.4.2 Vercel 28](#_Toc136207332)

[4.5 A projekt felépítése 33](#_Toc136207333)

[4.5.1 Adatbázis réteg 34](#_Toc136207334)

[4.5.2 Szükséges függvények és konfigurációk 35](#_Toc136207335)

[4.5.2.1 Lib 37](#_Toc136207336)

[4.5.2.2 Hooks 37](#_Toc136207337)

[4.5.2.2 Atoms 38](#_Toc136207338)

[4.5.3 Oldalak 38](#_Toc136207339)

[4.5.3.1 login 38](#_Toc136207340)

[4.5.3.2 index 39](#_Toc136207341)

[4.5.4 Komponensek 39](#_Toc136207342)

[4.5.4.1 Sidebar 39](#_Toc136207343)

[4.5.4.2 FollowedPlaylists 40](#_Toc136207344)

[4.5.4.3 Center 40](#_Toc136207345)

[4.5.4.4 PlayerErrorBox 41](#_Toc136207346)

[4.5.4.5 FollowButton 41](#_Toc136207347)

[4.5.4.6 Songs 41](#_Toc136207348)

[4.5.4.7 Song 42](#_Toc136207349)

[4.5.4.8 Recommend 42](#_Toc136207350)

[4.5.4.9 Player 42](#_Toc136207351)

[4.6 Tesztelés 44](#_Toc136207352)

[4.6.1 Manuális tesztelés 44](#_Toc136207353)

[4.6.2 Felhasználói tesztelés 45](#_Toc136207354)

[5. Összefoglalás 46](#_Toc136207355)

[6. További fejlesztési lehetőségek 47](#_Toc136207356)

[6.1 Vizuális fejlesztés 47](#_Toc136207357)

[6.2 Nyelv 47](#_Toc136207358)

[6.3 Statisztikai oldal 47](#_Toc136207359)

[6.4 Értékelés 47](#_Toc136207360)

[6.5 Spotify API kérések 48](#_Toc136207361)

[7. Köszönetnyilvánítás 49](#_Toc136207362)

[Forrásjegyzék 50](#_Toc136207363)

# 1. Bevezetés

## 1.1 Témaválasztás

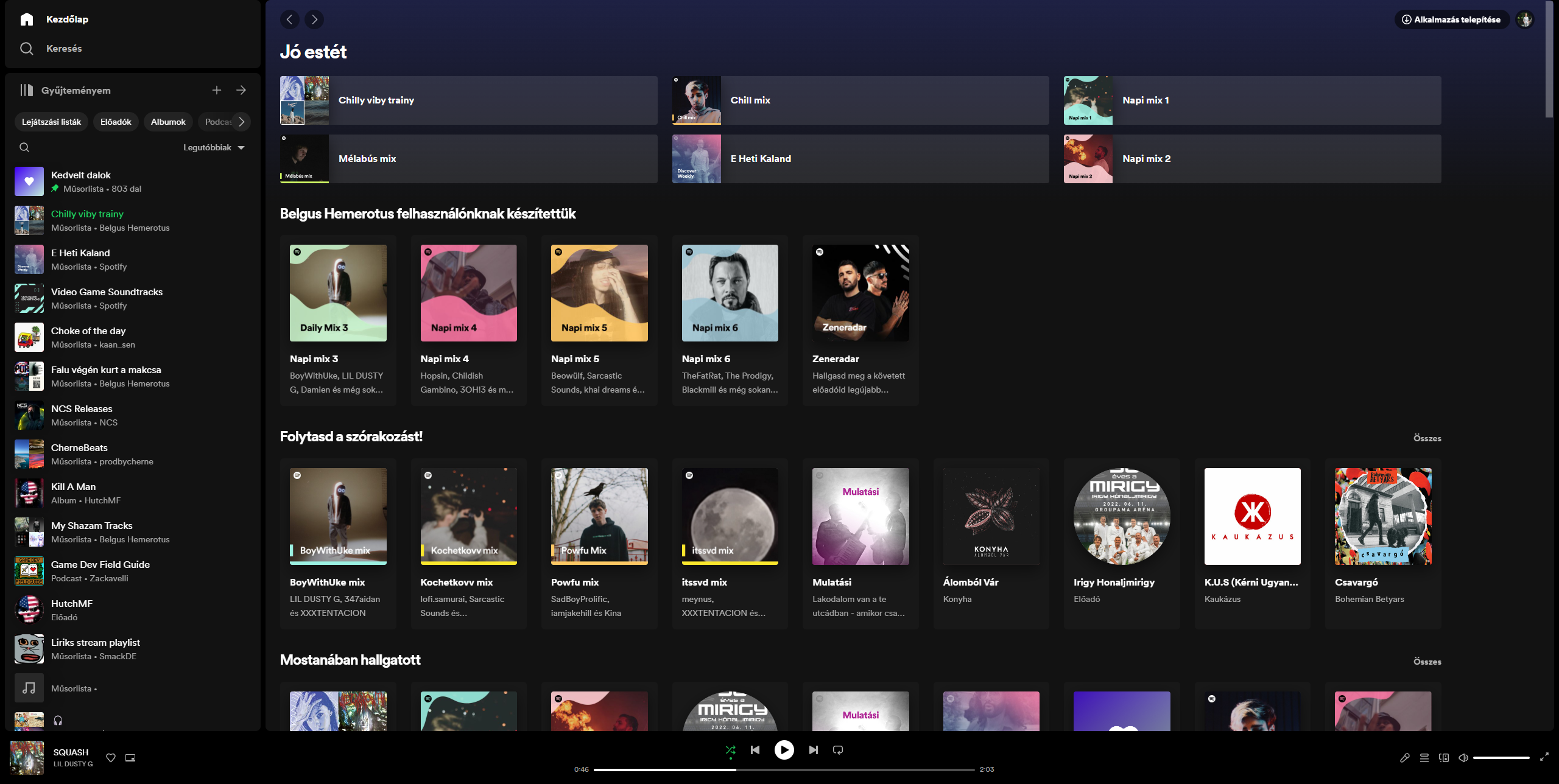
Az egyetemi éveim alatt mindig is nagy szerepet játszott a zene. Legyen az örömhírre vagy rossz hírre, tanulás alatt, közlekedés közben, sporthoz, és még sok más alkalom miatt. Több streaming szolgáltatót is kipróbálva, - többek között Tidal, YouTube Music, Amazon Music Unlimited, Apple Music, Spotify -, a Spotify-t találtam a leghasznosabbnak, mivel nem csak hivatalos zenéket találhatunk meg rajta és jobban elterjedt, mint konkurenciái, legalábbis Magyarországon.

Amit szerettem volna elérni a szakdolgozatommal, hogy egy kicsit átvariálva, kevesebb funkcióval, az én preferenciám szerint létrehozni egy úgymond ’’Spotify 2.0’’ – át. Amit átvettem nagyban tőle az a dizájn és magát a zenét is a Spotify-ból kérem le. Amit máshogy csinálok, hogy kizárólag a lejátszási listákon van a hangsúly. Ezeket tölti be az oldal, lehet kiválasztani őket és megnézni, milyen számok vannak rajta. Először a mi saját listáinkat tölti be az oldal, ezekkel nagyon interaktivitás nincs, csak eltudjuk indítani a lejátszást egy aktív eszközre, majd mutat ajánlott lejátszási listákat is. Ezek az ajánlások a Spotify által ajánlott listák, illetve más felhasználók által készített és ezen applikáció által beolvasott listák lehetnek. Ezeket tudjuk követni, így későbbre elmenteni, és szintén számokat lejátszani belőle egy aktív eszköz segítségével.

Dolgozatomat webes környezetben szerettem volna megvalósítani, mivel szakmai gyakorlatom alatt is webes technológiákkal foglalkoztam így, ezzel biztosítva a platform függetlenséget, és így a legkényelmesebb a felhasználóknak az elérés.

## 1.2 Spotify

A Spotify egy Svédországban kifejlesztett zenei streaming platform, ami ma már a világ sok más országában, területein elérhető. Tartalmai másolásvédettek, nagyobb zenei kiadóknak, illetve kisebb szabadúszó zenei alkotóknak is lehetőséget ad műveik feltöltésére és terjesztésére. A platform ingyen használható, ugyanakkor van egy prémium csomagja is. Ez a csomag többfajta módon is elérhető, más-más ár címkézéssel pl. Egyéni Prémium, Családi Prémium, Hallgatói Prémium. Többek között, amit átvettem zenék lejátszása funkciót, az ehhez a prémiumhoz kötött, így ezt csak olyan felhasználók tudják használni, akiknek van aktív prémium előfizetésük Spotify-ra.

  
**0. ábra** Spotify új webes dizájnja 2023.május

# 2. Alapvető technológiák

## 2.1 React

React[1], React.js, vagy ReactJS napjaink legelterjedtebb, ingyenes és nyílt forráskódú front-end JavaScript könyvtára. Fejlesztve a Facebook (most Meta) által, először 2013-ban került kiadásra. React-tel komponens alapú felhasználói felületeket lehet létrehozni, deklaratív és egységbe zárt módon. Ennek előnye, hogy komponenseinket könnyedén újra felhasználhatjuk. React azonban csak az állapotkezeléssel, és DOM (Document Object Model) manipulásával foglalkozik, így minden más tipikus front-end funkciók más csomagokon keresztül érhető el (pl. react router az útválasztáshoz).

React használható ’’egy oldalas’’, mobil, vagy szerveroldali betöltésű alkalmazások fejlesztésére más keretrendszerek által, amik ráépülnek, mint például a Next.js. Tipikusan JSX-et használva (vagy TSX, ha typescript-et használunk) fejlesztjük React alkalmazásunkat, ami a háttérben átkonvertál szimpla JavaScriptre, így, ha akarjuk JS fájlokkal is dolgozhatunk, de JSX használata sokkal egyszerűbb, és nagyon könnyedén tanulható, szintaxisa szinte megegyező. A JSX egy ’’superset”-e a JS-nek.

## 2.2 Next.js

A Next.js[2] egy nyílt forráskódú webfejlesztéshez használt keretrendszer. React-re épül és ad hozzá több funkcionalitást. Legnagyobb előnye, hogy egyben front-end és back-end-ként is használható, nem kell külön egy back-end könyvtárat keresni és használni. Node.js-et használ, mint fontosabb technológia, ami a REST-en alapul, mint kommunikáció a front-end és back-end részek között.

Fontosabb funkciói, amivel kiegészíti és megkönnyíti a React használatát:

* Oldal útválasztás – ’’Out-of-the-Box'’ működik fájlrendszer alapján
* Statikus oldali betöltés
* Szerveroldali betöltés
* Hibrid oldali betöltés
* Könnyű adatlekérés back-end-ről

## 2.3 Prisma

A Prisma[3] egy Node.js és TypeScript-en alapuló ORM (Object-relation mapping / Objektum-relációs leképzés). Ennek segítségével nem kell ’’nyers’’ SQL utasításokkal manipulálni adatbázisunkat (ennek egyik előnye az sql-injection támadások elleni védelem), így gyorsan és olvashatóan készíthetünk modelleket a sémánkban, könnyedén migrálhatunk adatokat, és GraphQL-hez hasonló módon végezhetünk ’’runtime’’ CRUD operációkat adatbázisunkban. Mindehhez társul egy ’’logger’’, amivel megtekinthetjük a háttérben lefutó SQL parancsokat. Jelenleg támogatott adatbázis fajtái a PostgreSQL, MySQL, SQLite, és MongoDB.

## 2.4 tRPC

A tRPC[4] egy End-to-end típus biztonságos API könyvtár. Legnagyobb előnye, hogy a fejlesztés során könnyebben el lehet kapni kisebb bug-okat, amik nagy fejtörést okoznának, ha egy elő production környezetben kerülnének elő. Más megemlítésre méltó előnye, hogy pl. VSCode-ban nagyban segít az autocomplete, és az egész könyvtár lightweight és nincs build lépés benne, így nem lesz generálva miatta új kód, tehát nem lesz miatta ’’runtime bloat’’.

## 2.5 NextAuth.js

A NextAuth.js[5] egy nyílt forráskódú autentikációs megoldás Next.js alkalmazásokhoz. Egyenesen a szerver nélküli elvet használó Next.js-hez volt fejlesztve. Ugyanakkor sok szerveroldali alkalmazáshoz is használható, tehát nagyon rugalmas. Lehet vele többek között OAuth 1.0, 1.0A, 2.0, és email/jelszó nélküli autentikációt végezni. Adapter támogatása is van, mi esetünkben ez a Prisma és ezt az opciót bekapcsolva egyszerűen az adatbázisunkba mentődnek el a felhasználók. Sok ’’Provider”-t támogat többek között, Google, Apple, Facebook, Spotify, stb.

## 2.6 TailwindCSS

A TailwindCSS[6] egy nyílt forráskódú CSS keretrendszer. A legnagyobb különbség más keretrendszerekkel ellentétben, mint például a Bootstrap, hogy nincsenek benne előre meghatározott klasszok, például gombok és táblák, hanem egyfajta ’’utility” klasszokat definiál. Ennek eredményeképp minden HTML elemre külön vagy csoportosítva lehet használni ezeket a kalsszokat akárhogyan keverve őket. Ugyanakkor akár saját új klasszt is létre lehet vele hozni. Plugin-okat is lehet használni, amik lényegében más fejlesztők által megírt új klasszok.

## 2.7 Spotify Web Api

A Spotify Web Api[7] REST-en alapuló, a Spotify adat katalógushoz kapcsolódó API. Használatához felhasználói szinten külön jogosultság kell. JSON meta adatokat ad vissza különböző kérésünkre. Ezek a kérések lehetnek zene előadókról, számokról, lejátszási listákról, vagy épp a lejátszás távoli vezérlése.

# 3. Felhasználói dokumentáció

## 3.1 Rövid ismertetés

Az alkalmazás lehetőséget ad a felhasználó Spotify regisztrációja/bejelentkezése után a lejátszási listái betöltésére, azok zenéinek megnézésére, ajánlott lejátszási listák megnézésére, azok elmentésére. Ha a felhasználónak van Spotify Prémium csomagja, akkor távirányító szerűen egy aktív Spotify eszközön le tudja játszani a látott zenéket.

## 3.2 Célközönség

Sokan, akik Spotify-t használnak egy idő után megunják saját lejátszási listájukat és új, rendszerezett zenéket keresnek. Célközönségem az ebben az esetben szenvedő Spotify felhasználók, akiknek van sok saját, illetve bekövettet lejátszási listája. Így az alkalmazásom által megoszthatja más felhasználóval ezeket, illetve ő is találhat új, mások által létrehozott listákat.

## 3.3 Rendszerkövetelmények

Azt szerettem volna, hogy a felhasználóknak ne keljen bajlódniuk az alkalmazás telepítésével, indításával, esetleges karbantartásával, és minél több felhasználónak elérhetővé tegyem, ezért webes környezetben hoztam létra az alkalmazást. Mivel nem időznek sokat a felhasználók az oldalon, csak böngésznek lejátszási listákat, amiket esetleg elindítanak az aktív eszközükön lejátszásra, ezért úgy döntöttem, hogy alkalmazásom publikálom, így elég csak felkeresni az oldalt és máris lehet használni (Ezért a telepítést és használatát local-ban, illetve hasonló deploy-olást a fejlesztői dokumentációban részletezném a 4.2, 4.3 és 4.4 pontban.)

Alkalmazás elérhetősége: <https://yfitops-2023.vercel.app/>

Használatához így csak egy böngészőre van szükség és internetkapcsolatra.

A következő böngészőkkel teszteltem, ezekben megfelelően működik minden funkcionalitás:

* Microsoft Edge (113.0.1774.35)
* Firefox (112.0.1)
* Safari (16.4)
* Google Chrome (113.0.5672.126)

## 3.4 Használat

### 3.4.1 Bejelentkezés

Az oldal első megnyitásánál automatikusan átirányít minket a bejelentkezési képernyőre (1. ábra). Itt láthatjuk az oldal logóját és az egyetlen módot ahogyan be tudunk jelentkezni az oldalra.

A képen képernyőkép, Grafika, rajzfilm, zöld látható

Automatikusan generált leírás

**1. ábra** Az alkalmazás bejelentkező oldala

A ”Log in with Spotify” gombra kattintva indíthatjuk el a bejelentkezés folyamatát. Ez átirányít minket a Spotify hivatalos bejelentkezési oldalára (2. ábra), hacsak a böngészőbe nem voltunk már bejelentkezve. Itt többfajta módon tudunk bejelentkezni Spotify fiókunkba, illetve közvetlen regisztrálni is tudunk. Ezt követően az alkalmazás engedélyt kér, hogy különböző funkciói működhessenek, ennek elfogadása után vissza irányit minket a fő oldalra.

A képen szöveg, képernyőkép, szoftver, tervezés látható

Automatikusan generált leírás

**2. ábra** Spotify bejelentkezési oldala

Az engedélyek elfogadását csak egyszer kell elfogadni, utána az alkalmazás, illetve maga a Spotify is megjegyzi, így később a ”Log in with Spotify” gomb egyből átirányíthat minket a főoldalra.

### 3.4.2 Főoldal

Sikeres bejelentkezést követően az alkalmazás főoldala tárul elénk. Ez a betöltést követően egy előre beállított alapértelmezett lejátszási listát tölt be elénk (későbbiekben ha ez a lejátszási lista törlésre kerül, akkor ez a központi elem sem fog betöltődni rendes adattal ). Ezen a központi elemen kívül egy bal és jobb oldalsó sáv is látható (nagyobb képernyőkön), illetve egy lejátszást vezérlő elem az oldal alján. Ennek a vezérlőnek a működésére Spotify Prémium felhasználónak kell lennie (3. ábra), így, ha olyan felhasználóval jelentkezünk be, ami ennek hiányába van, ez a lejátszási vezérlő nem jelenik meg (4. ábra).

A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

Automatikusan generált leírás

**3. ábra** Prémiummal rendelkező felhasználó

A képen képernyőkép, szöveg, szoftver, Multimédiás szoftver látható

Automatikusan generált leírás

**4.ábra** Prémium nélküli felhasználó

#### 3.4.2.1 Bal oldali sáv

A bal oldali sávba kerülnek bele a mi saját lejátszási listáink. Ennek a limitje jelenleg 50 lista. Ez azt jelenti, ha valakinek több mint 50 lejátszási listája van, az nem fog betöltődni az alkalmazásra.

Ide nem csak azok a lejátszási listák töltődnek be, amiket a felhasználó hozott létre, hanem amit bekövetett/kedvelt. Ezek között lehet kattintással válogatni. A kiválasztott lista a központi komponensben jelenik meg (erről majd később a 3.4.2.4 pontban).

A saját listáink bizonyos időközönként egy ellenőrzés is lefut, így dinamikusan az alkalmazás használata közben is, ha mentünk el lejátszási listát a Spotify-on, az alkalmazásban is meg fog ez jelenni.

Ezen kívül a mi lejátszási listáink alatt megtalálható, az ezen alkalmazásban bekövetett listák is. Ezek a listák nem kerülnek elmentésre a Spotify-ban, csak ebben az alkalmazásban, így nem kell félni, hogy véletlen káoszt csinálunk a Spotify platformján. Ide egyértelműen csak olyan listák kerülhetnek be, amiket nem tőlünk mentett el az alkalmazás.

Egy új felhasználónak nem lesznek bekövetett listái, de amint bekövet egy listát az alkalmazásban egyből megjelenik ennek a komponensek az alján. Ezek láthatóak visszamenőleg a 3. és 4. ábrán is.

#### 3.4.2.2 Jobb oldali sáv

A jobb oldali sávba generálódnak le az úgynevezett ”featured” lejátszási listák, amik egyszerre tartalmazzák a Spotify által létrehozott ajánlott listákat, illetve más felhasználók listáit is.

Ez a lista mindig dinamikusan változik. Naponta más a Spotify által ajánlott, és minden újratöltésnél más-más felhasználók által létrehozott lejátszási listákat láthatunk.

Ezekre kattintva a lista ugyanúgy betöltődik a központi komponensbe, mint a bal oldali sávnál, azzal az egy különbséggel, hogy látható lesz egy gomb is, amivel követhetjük, vagy megszüntethetjük a követését ezeknek az ajánlott lejátszási listáknak.

#### 3.4.2.3 Információs doboz

Ebben a dobozban kétfajta üzenet jelenhet meg, kétfajta hibára hivatkozva. Ezek a következőek:

* Nem Prémium felhasználó
* Nincs aktív lejátszási eszköz

Az első eset akkor következhet be, ha a központi oldalon rákattintunk egy dalra, ezzel elindítva a lejátszás folyamatát, de mivel nincs Prémiumunk egy a funkció nem érhető el, így erről tájékoztató üzenet jelenik meg az oldal tetején (5. ábra).

A képen szöveg, képernyőkép, szoftver, Számítógépes ikon látható

Automatikusan generált leírás

**5. ábra** ”Nincs Prémium” hibaüzenet

A második csak akkor fordulhat elő, ha az első hibája már nem áll fent, tehát olyan felhasználóval vagyunk bejelentkezve, akinek van Prémiuma. Ugyancsak akkor keletkezik az üzenet megjelenítése, amikor el akarunk indítani egy listán egy dalt a központi komponensből, vagy az oldal betöltésénél, ha már akkor sincs aktív eszközünk (az aktív eszköz, és lejátszási funkciót később, a vezérlő fejezetében taglalom tovább szám) (6. ábra).

A képen szöveg, képernyőkép, szoftver, Számítógépes ikon látható

Automatikusan generált leírás

**6.ábra** Nincs aktív lejátszási eszköz hiba

Mind a kettő hibaüzenet ott marad a helyén ahogyan görgetjük az oldalt, hogy mindig lássuk, ha valamilyen hiba áll fent. Az üzenet szimplán az ’X’-re kattintva tüntetjük el.

#### 3.4.2.4 Központi elem

A központi elem-re töltődnek be a lejátszási listák, innen lehet lejátszani őket, illetve böngészni milyen dalok vannak rajta. Ugyanakkor más elemek is helyezkednek el itt.

Jobb felső sarokba került a felhasználó profilképe, amit a Spotify-on használ, neve, és egy ikon ami mutatja, hogy azt megnyomva tud kijelentkezni az alkalmazásból. Ha nincs profil képe egy alapértelmezett ikon jelenik meg helyette (7. ábra).

A képen szöveg, képernyőkép, Betűtípus, embléma látható

Automatikusan generált leírás

**7/1.ábra** Alapértelmezett képű kijelentkezési doboz

A képen képernyőkép, Betűtípus, kör, embléma látható

Automatikusan generált leírás

**7/2.ábra** Spotify-ról betöltött képű kijelentkezési doboz

Ezen kívül az elem tetején láthatjuk még a lejátszási lista képét, ha van neki, ha nincs akkor egy alapértelmezett elemet, illetve a lista nevét és megtalálhatjuk mellette.

A név mellett helyezkedik el a követés gomb, ami csak olyan lejátszási listáknál jelenik meg, amiket nem mi hoztunk be az alkalmazásba. Követve a listákat azok elmentődnek a bal oldali sávba, így később is megtalálhatóak lesznek. Ha úgy döntöttünk, hogy ki akarjuk követni a listát, ennek a gomb helyén megjelenik egy másik gomb, ami erre szolgál (8. ábra). Ennek megnyomásával a bal oldali sávból is eltűnik a lista.

A képen képernyőkép, szöveg, ibolya látható

Automatikusan generált leírás

**8/1.ábra** Lista képe, neve, és követési gomb

A képen szöveg, képernyőkép, Betűtípus, embléma látható

Automatikusan generált leírás

**8/2.ábra** Lista képe, neve, és követési gomb

#### 3.4.2.5 Lejátszási vezérlő

A lejátszási vezérlő az oldal legalján helyezkedik el és onnan nem mozdul el görgetéssel sem. Ahhoz, hogy megjelenjen Spotify Prémium felhasználóval kell bejelentkeznünk, és megfelelő használatához kell egy aktív Spotify eszköz.

Aktív eszköz azt jelenti, hogy a telefonunkon vagy asztali gépünkön el kell indítanunk a Spotify hivatalos alkalmazását, és elindítani rajta egy számot, majd akár meg is lehet azt állítani. Erre azért van szükség, mert a lejátszási vezérlő amolyan távirányítóként tud csak működni. Ezek lépéseken végigérve indíthatunk el lejátszási listát és használhatjuk a vezérlő funkcióit.

A vezérlő bal oldalán helyezkedik el az éppen lejátszott szám, képe, neve és előadója. Ez lehet üres is, ha még nincs aktív eszközünk használva, így nem lehet lekérni róla információt (9. ábra).

A képen szöveg, képernyőkép, Betűtípus, Grafika látható

Automatikusan generált leírás

**9/1.ábra** Éppen játszott szám információk

A képen képernyőkép, Grafika, Betűtípus, embléma látható

Automatikusan generált leírás

**9/2.ábra** Éppen játszott szám aktív eszköz nélkül

Ezt követően a központi vezérlők láthatók, sorban:

* Keverés ki/be
* Előző szám
* Szünet/Lejátszás
* Következő szám
* Nincs ismétlés/Lista ismétlése/Jelenlegi szám ismétlése

Az előző és következő szám ikonjain kívül, attól függően milyen állapotban van a lejátszó, a gombok ikonjai változhatnak kinézetre (10.ábra).



**10/1.ábra** Vezérlők keverés, lejátszás közben, lista ismételve



**10/2.ábra** Vezérlők nincs keverés, szünetelve, szám ismételve

A vezérlő elem leg jobb oldalibb pontján található a hangszabályzó. Ez a vezérlő egy 0-100-as skálán engedi beállítani a hangerőt. Kétféle módon használható. egyik a 2 hang ikonokra kattintva a bal oldali -10-el csökkenti, a jobb oldali +10-el növeli a hangerőt. A másik mód a csúszka használata, ahol szabad kézzel szabható meg szemmértékkel a hangerő. Az oldal betöltésénél az alapértelmezett hangerőre lesz beállítva a lejátszó, ami 50.

# 4. Fejlesztői dokumentáció

## 4.1 Tervezés és célkitűzés

Szeretettem volna megvalósítani egy letisztult, könnyen fejleszthető és karbantartható webalkalmazást készíteni. Adatok tárolására adatbázist használni. Ezeket figyelembe véve olyan technológiákat választottam, amik könnyen tanulhatóak, egyszerűen használhatóak és egymással szorosan kompatibilisek.

## 4.2 Az alkalmazás telepítése

A fejlesztést Windows 11 (22H2) operációs rendszeren végeztem, telepítési útmutatót ezen rendszeren részletezem, angol nyelvű felületeken. Az összes felhasznált technológia platformfüggetlen, tehát más operációs rendszeren is lehet telepíteni, továbbfejleszteni, a szükséges függőségek telepítése után.

### 4.2.1 Node.js és npm

A függőségek telepítéséhez szükség van a következőkre:

* Node.js[8] (18.14.1 vagy frissebb)
* npm[9] (9.6.6 vagy frissebb)

Az alkalmazást Next.js keretrendszerrel valósítom meg, ami a Node.js-re épül. Az npm pedig a Node.js alapértelmezett csomagkezelője, így nem igényel egyéb telepítést, elég csak a Node.js-t telepíteni.

A telepítéshez szükséges fájlok a Node.js weboldalén érhetők el. Szükség esetén a Node.js GitHub oldalán[10] lehet találni részletes bemutatót a telepítésről. A telepítés után a a node -v, npm version, illetve npx -v parancsokkal ellenőrizhetjük a sikerességet (11.ábra).

A képen szöveg, képernyőkép, képernyő, szoftver látható

Automatikusan generált leírás

**11.ábra** Node.js és npm verziók ellenőrzése

### 4.2.2 Függőségek telepítse

Az alkalmazáshoz szükséges függőségek telepítése, a Node.js telepítése után, a projekt gyökér mappájába navigálva, egy konzolablakban az npm install parancsot kiadva lehetséges. Ez letölt és telepít minden függőséget. Parancs lefutásának időtartama internetkapcsolat és lemez író sebességétől függ.

A függőségeket és azok verzióit a projekt gyökér könyvtárában elhelyezkedő package.json fájlban találjuk (12.ábra).

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

**12/1.ábra** A projekt függőségei

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

**12/2.ábra** A projekt fejlesztéséhez használt függőségek

Ha rákeresük ezekre a csomagokra az npm weboldalán további információkat tudhatunk meg róluk.

### 4.2.3 Spotify for Developers

Mivel NextAuth-ot haználok autentikációhoz, és ott használom a Spotify provider-t, ezért létre kell hozni egy alkalmazást a Spotify for Developers oldalán. Innentől kezdve az itt kapott kliens azonosító és kliens kulcsot használva érhetjük el ezt a felületet. Ha már van Spotify felhasználónk, akkor nem szükséges új felhasználót regisztrálni, használhatjuk itt azt. Az alkalmazás létrehozásáról és konfigurációjáról a Spotify for Developers[11] részletes útmutatót biztosít.

Fontos konfiguráció, hogy miután létrehoztunk az alkalmazást, a Settings menüben, az Edit gomb megnyomásával beállítjuk a Redirect URIs-t (13. ábra).

A fejlesztéshez megadandó URI:

* <https://localhost:3000/api/auth/callback/spotify>

Deploy-olt alkalmazáshoz megadandó URI:

* https://<your-website-index-url>/api/auth/callback/spotify

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

**13.ábra** Spotify for Developers alkalmazás Redirect URIs

A két értéket, kliens azonosítót és kulcsot, is itt találjuk meg. A kulcsot a View client secret linkkel tudjuk elérni (14. ábra).

A képen szöveg, képernyőkép, sor, Betűtípus látható

Automatikusan generált leírás

**14.ábra** Spotify for Developers azonosító és kulcs

Fontos itt még megjegyezni, hogy az alkalmazás státusza Development mode-ban van, az Spotify írányelvei miatt csak azok a felhasználók tudják használni az alkalmazást, akik a User Management fülben hozzá lettek adva. Ez csak egy email-t jelent, ami a Spotify fiókhoz van használva, és egy nevet, amivel könnyebben azonosíthatóak a felhasználók. Ha olyan fiókkal akarunk bejelentkezni az alkalmazásba, ami nem lett hozzáadva itt, ők nem fognak tudni bejelentkezni az alkalmazásba.

Érdekes lehet még, hogy az oldal statisztikákkal is szolgál az alkalmazásról, hogy milyen kérések jöttek be különböző végpontokra, és például a felhasználók ország szerinti felbontását (15. ábra).

A képen szöveg, képernyőkép, diagram, Diagram látható

Automatikusan generált leírás

A képen képernyőkép, szöveg, térkép látható

Automatikusan generált leírás

**15. ábra** Spotify for Developers statisztikák

### 4.2.4 Adatbázis

Az alkalmazáshoz egy online host-olt PostgreSQL adatbázis készítettem. A host szerviz platformja a Supabase[12]. Ez az adatbázis cím teljesen megfelelő a mostani alkalmazás struktúrára és ezzel semmi teendő nincs, csak használni. A szakdolgozat írásának idejében az adatbázis címe a következő:

postgresql://postgres:yEe2NZ5&JMoJ@db.ljcujwrozwiflkyxitqw.supabase.co:5432/postgres

Ha bármilyen ok miatt nem lenne megfelelő ez az adatbázis és készíteni kellene egy újat, erről a Supabase oldalán részletes leírást lehet találni[13].

Fontos még, ha új adatbázist hozunk létre, annak bekonfigurálása után, és hozzáadása a .env fájlba, adjuk ki az npm run seed parancsot konzolba, amivel 2 alapvető adat, egy user és egy playlist, hozzá lesz adva az adatbázishoz.

### 4.2.5 Környezeti változók

Utolsó lépésként mielőtt elindítanánk az alkalmazásunkat, létre kell hoznunk a környezeti változókat. Ehhez a projektben megtalálható .env.example fájl tartalmát kell átmásolni egy .env nevezetű fájlba, szintén a projekt gyökér mappájába. A következő tartalommal kell kiegészíteni a fájlt:

DATABASE\_URL="postgresql://postgres:yEe2NZ5&JMoJ@db.ljcujwrozwiflkyxitqw.supabase.co:5432/postgres"

NEXTAUTH\_SECRET=<NEXTAUTH\_SECRET>

NEXTAUTH\_URL="http://localhost:3000"

NEXT\_PUBLIC\_SPOTIFY\_ID=<Client ID>

NEXT\_PUBLIC\_SPOTIFY\_SECRET=<Client Secret>

SECRET=<SECRET>

A <Client ID> és <Client Secret> a 4.2.3 pontban említett módon lehet megtalálni.

A <NEXTAUTH\_SECRET> és <SECRET> értékekhez a NetxAuth[14] javaslata szerint egy openssl rand -base64 32 parancsal lehet gerenálni. Míg ezt egy linux környezetben a konzolba ki lehet adni, addig Windows-on ezt nem lehet megtenni. Helyette online lehet generátorral lehet ezt elérni[15] (16.ábra).

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

**16. ábra** openssl parancs

Ha más adatbázist szeretnénk használni, akkor a DATABASE\_URL-t ki kell annak az elérési címére cserélni.

## 4.3 Alkalmazás futtatása helyileg

Miután a konfigurációval végeztem, a projekt gyökér mappájában egy parancssorban kiadva az npm run dev paranccsal indíthatjuk el az alkalmazást. Ez a <https://localhost:3000/> címen indítja el helyileg az alkalmazást (17.ábra).



**17. ábra** npm run dev parancs

Ez után készen állunk az alkalmazás további fejlesztésére. A Next.js által a módosítások, amiket végzünk a kódban egyből reflektálódnak a helyileg élőben indított oldalon, azaz nem kell újra indítani az alkalmazást.

## 4.4 Alkalmazás publikálása

A legegyszerűbben elérni és használni az alkalmazást úgy lehet, ha azt publikáljuk, így nem kell a sokszor hosszadalmas telepítésen átmenni. A következőkben a Vercel által biztosított publikációs folyamaton megyek végig.

### 4.4.1 GitHub

A verziókövető rendszerek használata elsősorban a több emberes projekteken a megszokottak, de a folyamatos deploy-oláshoz elengedhetetlen és szinte egyetlen módja.

Szükségünk lesz egy GitHub[16] fiókra. Majd az oldalon létre kell hozni egy új repository-t és ide feltölteni a projektünket. Ehhez útmutatóul a GitHub dokumentációja szolgál[17].

### 4.4.2 Vercel

Vercel a cég a Next.js mögött, és így az egyértelmű döntés a deploy-olás platformjára. A Next.js dokumentációban található útmutató[18] alapján gyorsan és könnyedén importálhatjuk projektünket GitHub-ról. Alap esetben ez az első import és deploy sikeres lenne, de a projektben használok környezeti változók ellenőrzését. Erről itt külön gondoskodnunk kell. Legegyszerűbb módja ha az egész .env fájlunk tartalmát kimásoljuk, és szimplán beillesztjük az Environment Variables részen a mezőbe (18.ábra).

A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

Automatikusan generált leírás

**18.ábra** Környezeti változók Vercel-en importálás közben

A változók megadásával már sikeresen véghez mehet az import és az első deploy is.

Miután végére ért a folyamat, kapni fogunk egy automatikusan kiosztott domain-t, amin az alkalmazás elérhető lesz. Ezt rögötön be is kell írni két helyre.

Először a Vercel-en meg kell változtatni a NEXTAUTH\_URL-t erre az URL-re. Ezt úgy tudjuk megtenni, hogy először a projektünk felső navigációs sávjában a settings menüre navigálunk (19. ábra).

A képen képernyőkép, szöveg, Betűtípus látható

Automatikusan generált leírás

**19. ábra** Vercel projekt navigációs sáv

Ezután a kapott oldalon a bal sávban kiválasztjuk az Environment Variables menüpontot (20.ábra)

A képen szöveg, képernyőkép, Betűtípus, tervezés látható

Automatikusan generált leírás

**20.ábra** Vercel settings opciók

Itt láthatjuk az összes jelenlegi környezeti változónkat és itt a NEXTAUTH\_URL utolsó ikonjára kattintva Edit-elhetjük és a value-höz elmenthetjük az új url-ünket (21.ábra).

A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

Automatikusan generált leírás

**21/1.ábra** Vercel környezeti változók listája

A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

Automatikusan generált leírás

**21/2.ábra** Vercel NEXTAUTH\_URL módosítása

Ezek után ne felejtsük el redeploy-olni alkalmazásunkat, amit a felső navigációs sáv Deplyoments fülén érhetünk el. Itt láthatjuk az össze deployment-ünket (22.ábra).

A képen szöveg, képernyőkép, szoftver, Betűtípus látható

Automatikusan generált leírás

**22.ábra** Vercel Deployments lista

Itt a legelső elemen, amin a ”(current)” jelző is szerepel, utolsó ikonra kattintva tudjuk redeploy-olni (23.ábra).

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

**23/1.ábra** Vercel redeploy menü

A képen szöveg, képernyőkép, szoftver, Betűtípus látható

Automatikusan generált leírás

**23/2. ábra** Redeploy megerősítő ablak

Fontos még a kapott URL-t kicserélni/hozzá adni a Spotify for Developers alkalmazásunkban a Redirect URIs-hoz. Ez a 4.2.3 pontban van említve és részletesen leírva.

Ezzel ténylegesen elkészült a weboldal publikálása. Továbbiakban ha a main ágra történik új commit a GitHub repo-ba, automatikusan indulni fog egy deploy az új frissítésekre, aminek az eredményéről email-ben is értesít minket a Vercel.

## 4.5 A projekt felépítése

A szakdolgozat során készített alkalmazás fő szerkezete az alábbi (24.) ábrán látható. A következőkben a fontosabb könyvtárakat, elemeket, fájlokat ebben a fejezetben részletezném.

A képen szöveg, képernyőkép, Betűtípus, szoftver látható

Automatikusan generált leírás

**24. ábra** A projekt felépítése

### 4.5.1 Adatbázis réteg

Az alkalmazás megvalósításához egy PostgreSQL adatbázist használok, ami egy nyílt forráskódú, ingyenesen használható relációs adatbázis. Az SQL utasításokat egy ORM kiegészítésen keresztül kommunikálom az adatbázissal. Az adatbázis sémájáért a prisma/schema.prisma fájl felel.

Az adatbázis egyed-kapcsolati modell diagramja az alábbi 25. ábrán látható.

A képen szöveg, képernyőkép, szoftver, szám látható

Automatikusan generált leírás

25.ábra Az adatbázis egyed-kapcsolat diagramja

### 4.5.2 Szükséges függvények és konfigurációk

Konfigurációs fájlok, amik a gyökér könyvtárban helyezkednek el, mind a saját dokumentációjából lehet beállítani, sokszor Out-of-the-Box kézhez kapjuk őket. Ezek a következőek:

* .eslintrc.cjs
* .gitignore
* next.config.mjs
* postcss.config.cjs
* prettier.config.cjs
* tailwind.config.ts
* tsconfig.json

Ezeknek a fájloknak a beállítását és részletezését egy projektben találjuk meg leghatékonyabban, amiből én is használtam a beállításokat. Ez a stack a T3 Stack[19] egy boilerplate-et generál nekünk és elvégzi a szükséges konfigurációkat, hogy egyből kezdhessünk kódolni és dolgozni a lényegi munkán.

További konfigurációs fájlok, aminek a kezdeti állapota ebből a boilerplate-ből való:

* src/env.mjs
* src/utils/api.ts
* src/styles/globals.css – átírt
* src/server/db.ts
* src/server/auth.ts – átírva a mi használatunkra
* src/server/api/trpc.ts
* src/server/api/roots.ts – új elemek adva hozzá
* src/server/api/routers/\* - példa kitörölve, új router-ek készítve
* src/pages/\_app.tsx – kiegészítve
* src/pages/index.tsx – átírva
* src/pages/api/\*

Ezekből a módosításokból a legfontosabb az a src/server/api/routers/\* mappa fájljai. Ide kerültek a tRPC által használt adatbázis kérések gyűjteménye, táblákra lebontva. A 26.táblában lehet őket látni.

|  |  |  |
| --- | --- | --- |
| Kérés neve | Tábla | Leírás |
| *addPlaylist* | playlist | Bemenetként kap egy userID-t, playlistIds tömböt, playlistNames, tömböt. Létrehoz ezekkel az adatokkal új playlisteket. |
| *getUserPlaylists* | playlist | Bemenetként kapott userId-re vissza adja a hozzá rendelt playlist-eket |
| *getOtherUsersPlaylists* | playlist | Vissza adja az összes playlist-et ami nem a bemenetként kapott userId-hoz, és nem az alap Spotify User-hez van kötve |
| *getUserPlaylistIDs* | user | Bemenetként kapott userId-re vissza adja a hozá rendelt birtokolt playlist-ek ID-ját |
| *getSpotifyUserID* | user | Vissza adja a Spotif User objektumot |
| *followPlaylist* | usersOnPlaylists | Kapcsolótáblába felvesz egy userId és playlistI által meghatározott kapcsolatot |
| *unFollowPlaylist* | usersOnPlaylists | Kapcsolattáblából töröl egy userId és playlistId által meghatározott kapcsolatot |
| *findPlaylist* | usersOnPlaylists | Bemenetként megadott userId és playlistId-re vissza adja a playlistId-ját (arra van haszálva hogy megnézzük van e ilyen record) |
| *getUserFollowedPlaylistDetails* | usersOnPlaylists | Egy userId-ra vissza adja az összes playlist-et ami hozzá tartozik follow szempontból |

**26.tábla** tRPC kérések

#### 4.5.2.1 Lib

Az src/lib könyvtárba helyezkednek el a segédfüggvények. Ezek a következőek funkcióikkal együtt:

* time.ts – átkonvertálja milisec-ből m:ss formátumba az időt
* reloadSession.ts – olyan event-et trigger-el, ami újra hívja a session lekérdező metódusokat
* spotify.ts – létrehozza a LOGIN\_URL-t amit a NextAuth használ, illetve a SpotifyApi objektumot (singleton) amit kliens oldalon használok a Spotify-hoz irányított kérésekre

#### 4.5.2.2 Hooks

Itt lettek definiálva az egyedi hook-jaim, amit a react kontextusban használok. Ezek a következőek:

* useSpotify.ts – létrehozza a SpotifyApi objektumot (singleton) amit kliens oldalon használok a Spotify-hoz irányított kérésekre
* useSongInfo.ts – lekéri az éppen játszott dal adatait – player komponensbe van felhasználva

#### 4.5.2.2 Atoms

Ebbe a könyvtárba helyezkednek el a recoil[20] state management könyvtár általt használt state-ek. Ennek segítségével ezek a state-ek az alkalmazás bármelyik pontján elérhetőek és módosíthatóak könnyen és egyszerűen.

### 4.5.3 Oldalak

Mivel az \_app.tsx főleg a konfigurálás szempontjából fontos így azt nem részletezném annál többel, hogy adjuk hozzá a NextAuth-ot, Recoil-t és tRPC-t az oldalunkhoz, hogy használni tudjuk őket.

Ezen kívül két oldal van az alkalmazásan. A login-on kívül lényegében ez egy Single Page Application. A következőkben ezt a kettő oldalt részletezem.

#### 4.5.3.1 login

A login oldalon tudunk bejelentkezni a NextAuth segítségével. Először is szerveroldalon megnézem a getServerSideProps segítségével, hogy be vagyunk-e jelentkezve, és ha igen, akkor átirányítjuk a felhasználót a főoldalra.

Ha nem vagyunk bejelentkezve, akkor lekérjük a provider-eket a NextAuth-tól, mi esetünkben ez 1 darab provider, és átadjuk a props változóban kliens oldalra.

Kliensoldalon TailwindCCS segítségével letisztult design lett létrehozva, amely a logóból és gombok oszlopából áll, amivel a különböző provider-ekkel be lehet jelentkezni. A mi esetünkben csak egy gomb látható, mert egy provider-ünk van, de a kód fel van készítve arra, hogy több provider-ünk lehessen.

A gomb megnyomásával a NextAuth signIn folyamata elindul a kijelölt provider-rel, majd miután az véghez ment, visszairányít minket a főoldalra, ahol már bejelentkezve elérhetjük az oldal tartalmait.

#### 4.5.3.2 index

Először is a login oldalhoz hasonlóan, a szerveroldalon megnézem, hogy be van-e jelentkezve a felhasználó, és ha nem átirányítjuk a login oldalra.

A kliensoldalon itt jelenítem meg a főkomponenseket sorban:

* Head – Next.js komponens a <head> html tag-re
* Sidebar – Bal oldali sáv komponense
* Center – Központi elem komponense
* Recommend – Jobb oldali sáv komponense
* Player – lejátszó vezérlő komponense

Fontos még megjegyezni, hogy a styling-on kívül, még egy extra logika van benne, ami a Recoil segítségével megnézi az isPremium state-et és ha igaz, akkor megjelenítem a Player komponenst.

### 4.5.4 Komponensek

A komponenseim az src/components mappában találhatóak meg. A következőkben az előző fejezetben említett fő komponenseken mennék végig sorban. Ahol kisebb részre kellett bontani egy elemet és új komponenst létrehozni neki, ott egy alfejezetet nyitva részletezem az új komponenst.

#### 4.5.4.1 Sidebar

A Sidebar-ban először is lefuttatok egy useEffect-et ami, minden playlist váltásra, és első betöltésre fog lefutni. Ebben a részben kérem le a felhasználó lejátszási listáit, amiből maximum 50-et mentek el. Miután kaptunk választ a kérésünkre a Spotify API-tól, úgy a sikeresen lekért playlist-eket elmentem egy Recoil state-be. Majd kiszedem a kapott válaszból a playlistId-kat és playlistName-eket és átküldöm szerveroldalra a tRPC által, ami elmenti őket megfelelően az adatbázisba.

Ez után az elmentett Recoil state segítségével betöltjük a kapott playlist-eket egy oszlopba, majd egy onClick event-et adunk hozzá, amivel, ha rákattintunk egy másik Recoil state-re beállítja a playlist ID-ját.

Még meghívom a FollowedPlaylists komponenst is, ami kicsit másfajta logikát használ.

#### 4.5.4.2 FollowedPlaylists

Ez a komponens egy tRPC query segítségével betölti az adatbázisból a felhasználó bekövetett lejátszási listáit, majd hasonló formátumba megjeleníti őket a Sidebar komponensen belül. Ha nincs még követett listája a felhasználónak, akkor egyszerűen nem jelenít itt meg semmit.

#### 4.5.4.3 Center

Itt a Sidebar-ban elmentett state segítségével kérek részletes adatot a kiválasztott lejátszási listáról, majd ezt is elmentem egy másik state-be. Emellett egy listából random kiválasztok egy színt, amivel a hátteret állítom be.

Betöltünk egy header html tag-et absolute pozícióba, ahol megjelenítjük a felhasználó profilképét, ha van, nevét és egy ikont, ami a kijelentkezésre szolgál. Ha az infoBoxVisible Recoil state-ünk igazra van állítva megjelenítünk még a PlayerErrorBox komponensts is (4.5.4.4).

Betöltjük szépen formázva a Playlist nevét és képét, ha van, majd meghívjuk a FollowButton komponenst (4.5.4.5).

Ezek után végül meghívjuk legalulra a Songs komponenst (4.5.4.6).

#### 4.5.4.4 PlayerErrorBox

Kettő fajta hiba léphet fel, amiről a felhasználót tájékoztatom. Ezt az isPremium state-tel ellenőrzöm és döntöm el melyiket mutatom.

A komponens egy szövegdobozt jelenít meg, amiben a szöveg vagy a nincs prémium hibára hivatkozik, vagy ha ez nem áll fent, akkor a nincs aktív eszköz hibára. A komponens jobb oldalán megjelenik egy ’X’ ikon, amire rákattintva eltűnik az üzenet.

#### 4.5.4.5 FollowButton

Először is lekérjük a felhasználó, hogy a felhasználó követi-e az éppen kiválasztott playlist-et és az összes ő általa birtokolt playlist ID-kat. Amíg ezek a kérések vissza nem jönnek az adatbázisból, addig egy töltő gif látszódik. Választ megkapva megnézzük, hogy ha követi az éppeni playlist-et akkor a gomb szövege Followed lesz, ha nem akkor Follow. A másik válasz eredményével megnézzük, hogy a kiválasztott playlist benne van-e ebben a tömbben, és ha nincs akkor nem jelenítjük meg a gombot, mert ez azt jelenti, hogy a playlist alapból az ő birtokában áll, így nincs értelme a követésnek.

Ezek után betöltjük a gombot a megfelelő szöveggel és ikonnal. Erre a gombra raktam egy onMouseEnter és onMouseLeave event-et amivel a hover effektet biztosítom, illetve egy onClick event-et ami az éppeni gomb szövegből kiindulva vagy törli vagy hozzáadja a playlist-et a követett listába az adatbázisba.

#### 4.5.4.6 Songs

Ez a komponens úgy viselkedik, mint a html tag-eknél a lista ul vagy ol eleme. Végigmegy az összes dalon, amit a playlistState state-ben van megadva és a létrehoz egy Song (4.5.4.7) komponenst minden egyes dalnak.

#### 4.5.4.7 Song

Ez a komponens úgy viselkedik, mint a html tag-eknél a lista li eleme. Szépen formázva létrehozza a dal sorokat feltöltve őket a dal képével, címével, előadójával, albumával, és időtartalmával.

Ha rákattintunk ezekre a sorokra, meghív egy függvényt, amivel beállítom a currentTrackId state-et az éppeni dalra, isPlaying state-et igazra, és küld a Spotify API-nak egy kérés, hogy kezdje el játszani a playlist-et ezzel a dallal.

Itt hibák léphetnek fel, amik a nincs prémium és nincs aktív eszköz lehetnek, ezeket lekezelve a catch ágban allítom be a state-eket ennek megfelelően.

#### 4.5.4.8 Recommend

Ez a komponens nagyon hasonló a Sidebar komponenshez. Itt is lekérem a playlisteket a Spotify-tól, amik nem a felhasználóhoz tartoznak, majd elmentem őket az adatbázisba, és megjelenítem őket a Sidebar-hoz hasonlóan.

Rájuk kattintva átállítom a playlistId state-et amivel betöltjük a Center komponensbe.

#### 4.5.4.9 Player

Ebben a komponensben helyezem el a lejátszó vezérlőt és mögöttes logikáját. Először is az oldal betöltésénél lefut egy useEffect (27. ábra), ami meghívja a fetchCurrentSong-ot és alapértelmezetten 50-re rakja a hangerőt.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

**27. ábra** Init useEffect

A fetchCurrentSong egyből megpróbál adatot lekérni a Spotify API-tól az éppen futó trackről és lekezeli, ha valami hiba lép fel, így egyből tudom, ha nincs prémiuma a felhasználónak, vagy nincs aktív lejátszója. Így meg tudom jeleníteni az információs dobozban a hibákat a state-ek használatával.

Ha nincs hiba, akkor beállítom a kapott adatokra a következő 28. ábrán lévő state-eket, ezzel az inicializációt elvégezve.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

**28. ábra** fetchCurrentSong sikeres ágának inicializációja

Ez után mivel a volume state-et megváltoztattuk, lefut az ezt a state-et figyelő useEffect-et (29. ábra).

A képen szöveg, Betűtípus, képernyőkép, Grafika látható

Automatikusan generált leírás

**29.ábra** volume state useEffect

Ez a useEffect meghívja a debounceAdjustVolume-ot, ami lényegében meghívja a Spotify API-t és beállítja a hangerőt a volume értékre, de csak 300 ms-ként hívható meg, hogy a Spotify ne dobjon limit exceeded error-t, mert spam-elve lett kérésekkel.

Ezen a ponton az összes inicializációs beállítást elvégeztem és alapértékeket beállítottam. Ha volt valami hiba, akkor azt lekezeltem és üzenetet jelenítettem meg arról. Ezek után kezdődhet az adatok megjelenítése.

A player bal oldalán az éppen játszott track-ről láthatunk információt. A tarck képét, címét és előadóját. Ezután a központi gombok helyezkednek el, sorban:

* Keverés ki/be
* Előző szám
* Szünet/Lejátszás
* Következő szám
* Nincs ismétlés/Lista ismétlése/Jelenlegi szám ismétlése

Mindegyikhez tartozik egy onClick event, ezekből az előző és következő számhoz inline lett megírva az API call és hibakezelés. A többihez külön létre lett hozva egy handleFunction. Sorban handleShuffle, handlePlayPause, handleRepeat.

A repeat és shuffle-höz létre lett hozva egy-egy useEffect ami, a state-jeiket figyeli, így ha meg lesz változtatva a state, egyből értesíti a Spotify API és beállítja azok értékeit.

A handlePlayPause egy if else ágban tér el, ami annyit tesz, ha isPlaying igaz volt, akkor a pause kérés megy, fordítva pedig a play kérés a Spotify API-hoz.

A player jobb oldalán pedig a hangerő szabályzók vannak. Itt biztosítom, hogy a hangerő nem lehet 0-nál kisebb és 100-nál nagyobb a hibák elkerülése végett. A két hangerő ikonnal +/- 10 értékkel lehet csökkenteni vagy növelni a hangerőt, a csúszkával (input – range) pedig szabad kezüleg lehet beállítani a hangerőt. Az init alatt említett useEffect megy itt is végbe, mivel a volume értékét figyeli és ezek a gombok ezt a state-et változtatják.

## 4.6 Tesztelés

Mivel az egész alkalmazás az integrációs adatoktól függ, ezért manuális teszteket tudtam véghez vinni. Ezeket a teszteket és menetét részletezném ebben a fejezetben.

### 4.6.1 Manuális tesztelés

Létrehoztam új Spotify felhasználókat különböző listákkal, prémium előfizetésekkel, adatokkal. Ezeket használva végig mentem az összes adatlekérésen, hogy milyen értékre állítják be a state-eket ezek az API kérések. Mivel az alkalmazás kinézetét is funkcióit nagyban a state-ek vezérlik az API hívásokon keresztül, ezért ezeket state értékeket keverve végig mentem az összes létező kombinációjukon ezzel végig tesztelve az alkalmazás különböző fázisait.

Ahol volt segédfüggvény használva ott annak eredményét kiíratva, ellenőrizve értékét.

Integrációs teszteknél végig tesztelve az összes válasz kódot és azoknak okait, adatait megnézve, hibákat lekezelve, ellenőrizve.

### 4.6.2 Felhasználói tesztelés

Ismerőseim egy kis csoportját felkértem alkalmazásom kipróbálására a nélkül, hogy sok információt árultam volna el róla. Ezzel biztosítva, hogy az alkalmazás megbízható és biztonságosan használható élő használat közben.

Ennek a tesztelésnek eredményéül tovább tudtam finomítani a felhasználói élményt, a weboldal megjelenését és kisebb felmerülő bugok/hibák orvoslását.

# 5. Összefoglalás

A célom az volt, hogy egy letisztult, Spotify-hoz hasonló alkalmazást készítsek a lejátszási listákra fókuszálva, ahol az új listákat kereső felhasználók könnyedén találhatnak új, érdekes listákat, amiket el tudnak menteni. Fontos volt az átláhatóság és egyszerűség.

Az alkalmazás elkészítéséhez Next.js React keretrendszert használtam, melynek megismerését hasznosnak tartottam. Más technológiákat, amiket használtam, mint például Prisma, Vercel, tRPC, NextAuth, is nagyon hasznos és praktikus tapasztalatokhoz vezettek.

Véleményem szerinte, amit kitűztem célul a szakdolgozat megírásának kezdetekor, sikerült elérnem, mint esztétikai, mint funkcionalitás szempontból.

# 6. További fejlesztési lehetőségek

A dolgozat írásás közben egyre több új ötletem lett hogyan optimalizálnám a megoldásaimat, tehetném szebbé, még jobban felhasználó barátabbá a felületet, új funkciók bevezetésére. Ezek közül említenék meg egy párat.

## 6.1 Vizuális fejlesztés

Lehetne többfajta témát készíteni, pl. világos mód, kontraszt mód, amit a felhasználók szabadon választhatnának ki egy beállítások menüben.

Ezen kívül lehetne finomítani és javítani a kisebb méretű eszközökön való megjelenésen, mint például a telefonokon, tableteken való nézeten.

## 6.2 Nyelv

Lehetne több nyelvet implementálni és egy beállítások menüben a felhasználó szabadon választhatna a felület megjelenítési nyelvei közül. Ezzel a hasznos funkcióval több felhasználót érne el az alkalmazás.

## 6.3 Statisztikai oldal

Egy új oldal, ahol különböző statisztikákat lehetne látni egy felhasználóról, lejátszási listáról, vagy akár az oldal adatbázisában lévő összes adatról csoportosítva. Ez egy érdekes funkció lenne bármely felhasználónak.

## 6.4 Értékelés

Lehetne értékelni a lejátszási listákat, amik értékeik megjelennének a listák mellett. Lehetne egy rangsorolt ajánlói felület, ahol értékelésre lehet szűrni listákat. Havi szinten a legjobb listákat egy dedikált helyen lehetne feltüntetni a felhasználók számára, így a legjobb listák könnyen elérhetővé válnának.

## 6.5 Spotify API kérések

A Spotify Web API kéréseket lehetne optimalizálni, hogy gyorsabb és időben érkező válaszokat kapjunk.

# 7. Köszönetnyilvánítás

Ezúton szeretném megköszönni Pataki Norbertnek, hogy a szakdolgozat írása közben rendszeres konzultációk során adott tanácsaival, ötleteivel, információival segítette a dolgozatom elkészítését.

# Forrásjegyzék

1. ”React – JavaScript library”

<https://react.dev/>

[Hozzáférés dátuma: 2023. 05. 28.]

1. ”Next.js by Vercel – React Framework”

<https://nextjs.org/>

[Hozzáférés dátuma: 2023. 05. 28.]

1. ”Prisma - Next-generation Node.js and TypeScript ORM for Databases”

<https://www.prisma.io/>

[Hozzáférés dátuma: 2023. 05. 28.]

1. ”tRPC - End-to-end typesafe APIs”

<https://trpc.io/>

[Hozzáférés dátuma: 2023. 05. 28.]

1. ”NextAuth – Authentication for Next.js”

<https://next-auth.js.org/>

[Hozzáférés dátuma: 2023. 05. 28.]

1. ”TailwindCSS – Utility class CSS”

<https://tailwindcss.com/>

[Hozzáférés dátuma: 2023. 05. 28.]

1. ”Spotify Web API – Retrieve metadata from Spotify content”

<https://developer.spotify.com/documentation/web-api>

[Hozzáférés dátuma: 2023. 05. 28.]

1. ”Node.js”

<https://nodejs.org/en>

[Hozzáférés dátuma: 2023. 05. 28.]

1. ”npm”

<https://www.npmjs.com/>

[Hozzáférés dátuma: 2023. 05. 28.]

1. “Installation · nodejs/help Wiki · GitHub”

<https://github.com/nodejs/help/wiki/Installation>

[Hozzáférés dátuma: 2023. 05. 28.]

1. ”App Settings - Spotify for Developers”

<https://developer.spotify.com/documentation/web-api/concepts/apps>

[Hozzáférés dátuma: 2023. 05. 28.]

1. ”Supabase – Build in a weekend”

<https://supabase.com/>

[Hozzáférés dátuma: 2023. 05. 28.]

1. ”Supabase – Docs”

<https://supabase.com/docs>

[Hozzáférés dátuma: 2023. 05. 28.]

1. ”Options - NextAuth.js”

<https://next-auth.js.org/configuration/options#secret>

[Hozzáférés dátuma: 2023. 05. 28.]

1. ”CryptoTool - Online”

<https://www.cryptool.org/en/cto/openssl>

[Hozzáférés dátuma: 2023. 05. 28.]

1. ”GitHub: Where the world builds software · GitHub”

<https://github.com/>

[Hozzáférés dátuma: 2023. 05. 28.]

1. ”Adding locally hosted code to GitHub - GitHub Docs”

<https://docs.github.com/en/get-started/importing-your-projects-to-github/importing-source-code-to-github/adding-locally-hosted-code-to-github>

[Hozzáférés dátuma: 2023. 05. 28.]

1. ”Deploy to Vercel - Deploying Your Next.js App | Learn Next.js”

<https://nextjs.org/learn/basics/deploying-nextjs-app/deploy>

[Hozzáférés dátuma: 2023. 05. 28.]

1. ”Folder Structure – Create T3 App”

<https://create.t3.gg/en/folder-structure>

[Hozzáférés dátuma: 2023. 05. 28.]

1. ”Recoil – State management library for React”

<https://recoiljs.org/>

[Hozzáférés dátuma: 2023. 05. 28.]