Eötvös Loránd Tudomány Egyetem

Informatikai kar

Programozási nyelvek és Fordítóprogramok tanszék

# Zenék megosztását támogató platform

*Témavezető:*
Pataki Norbert
Docens

*Szerző:*
Vass Márk Benjámin
Programtervező informatikus, Bsc

*Budapest, 2021*

**EÖTVÖS LORÁND TUDOMÁNYEGYETEM**

**INFORMATIKAI KAR**

# SZAKDOLGOZAT-TÉMA BEJELENTŐ

Név: **Vass Márk Benjámin** Neptun kód: **I8WZ8L**

Tagozat: **nappali** Szak: **programtervező informatikus BSc**

Témavezető neve: **Pataki Norbert**

munkahelyének neve és címe: **ELTE IK, Prog. Nyelvek és Fordprog. Tanszék, 1117. Bp. Pázmány Péter sétány 1/C**

beosztása és iskolai végzettsége: **Adjunktus, PhD**

A dolgozat címe: **Zenék megosztását támogató platform**

A dolgozat témája: A dolgozat célja egy webes környezetben futó alkalmazás, melynek segítségével a zenészek eljuttathatják zenéjüket az érdeklődő közönséghez. A felhasználók több csoportja létezik: a zenészek, akik feltöltik és megosztják saját zenéjüket, valamint a rajongók, akik meghallgathatják a feltöltött zenéket. A dalok tag-ekkel vannak ellátva, amit feltöltéskor adhat meg a zenész. Ezen tag-ek felhasználásával a szoftver ajánlásokat tesz további zenék meghallgatására, így ajánló rendszerként funkcionál. A felhasználó lejátszási listákat állíthat össze, az így elkészített listát a saját eszközén lejátszhatja. A szoftver használata regisztrációhoz kötött.

A szoftver két részből áll: a backend logika Python nyelv segítségével, Flask keretrendszerben, a frontend VUE.JS keretrendszer alkalmazásával készül. A szoftver perzisztens módon kezeli a lényeges adatokat. A rendszer cross-platform megvalósítással készül, konténerizált futtatásra felkészített.

# Tartalomjegyzék

**1. Bevezetés** **3**

1.1. Témaválasztás indoklása . . . . . 3

1.2. Megoldandó feladat leirása . . . . . 4

**2. Felhasználói dokumentáció** **5**

2.1. Rendszerkövetelmények . . . . . 5

2.1.1. Szerver oldali követelmények . . . . . 5

2.1.2. Kliens oldali követelmények . . . . . 7

2.2. Telepités . . . . . 8

2.2.1. Konténerizált futtatás . . . . . 8

2.2.2. Nem konténerizált futtatás . . . . . 11

2.3. Felhasználói felület bemutatása . . . . . 12

2.3.1. Alkalmazás felépitése . . . . . 12

2.3.2. Üdvözlő nézet . . . . . 13

2.3.3. Bejelentkezési nézet . . . . . 13

2.3.4. Regisztrációs nézet . . . . . 14

2.3.5. Albumok szerkesztése nézet . . . . . 14

2.3.6. Zeneszámok szerkesztése . . . . . 15

2.3.7. Albumok nézet . . . . . 16

2.3.8. Előadók nézet . . . . . 17

2.3.9. Számok nézet . . . . . 18

2.3.10. Lejátszási nézet . . . . . 18

2.3.11. Lejátszási lista nézet . . . . . 18

2.3.12. Felfedezési nézet . . . . . 19

**3. Fejlesztői dokumentáció** **20**

3.1. Fejlesztői környezet . . . . . 20
3.2. Szerver oldal . . . . . 21
3.2.1. Specifikáció . . . . . 21
3.2.2. Használt módszerek és könyvtárak . . . . . 21
3.2.3. Adatbázis . . . . . 22
3.2.4. Adatbázis sémák . . . . . 23
3.2.5. Project struktúra . . . . . 30
3.2.6. RESTful interfész . . . . . 43
3.2.7. Applikáció modul . . . . . 44
3.3. Kliens oldal . . . . . 46
3.3.1. Specifikáció . . . . . 46
3.3.2. Használt könyvtárak . . . . . 46
3.3.3. Felépités . . . . . 46
3.3.4. Kliens oldali adatok tárolása . . . . . 49
3.3.5. Felhasználó authentikálásának folyamata . . . . . 49
3.3.6. Kliens oldali routing . . . . . 55
3.4. Tesztelés . . . . . 59
3.4.1. Szerver oldali unit teszt . . . . . 59
3.4.2. Szerver oldali integrációs teszt . . . . . 63
3.4.3. Kliens oldali integrációs teszt . . . . . 67
3.5. További fejlesztési lehetőségek . . . . . 67

**4. Összegzés** **68**

**5. Köszönetnyilvánítás** **69**

**Irodalomjegyzék** **70**

**Forráskódjegyzék** **71**

# 1. fejezet

# Bevezetés

## 1.1. Témaválasztás indoklása

Napjainkban egyre jobban elterjedtek az ajánlórendszereken alapuló multimédia szolgáltatások, mint a YouTube, Spotify, Soundcloud, Netllfix stb. Az emlitett szolgáltatások nagy része megpróbálja elemezni az alkotók által feltöltött tartalmakat, majd az abból kinyert metaadatokat dolgozza fel és próbál ajánlani újabb tartalmakat a felhasználónak az addig elfogyasztott tartalom után. Mivel ezek a szolgáltatások maguk elemzik a tartalmakat, igy nem adják meg a szerzőknek a szabadságot hogy saját leiró modellt készitsenek a saját tartalmukhoz ami pontosabb lenne mint amit egy mai modern algoritmus meg tud határozni. Ha megadnánk ezt a szabadságot a tartalom készitőinek, akkor ideális esetben az ajánlórendszerek sokkal személyreszabottabb és pontosabb ajánlást tudnának adni a végfelhasználóknak. Ez inspirálta a témaválasztásomat, hogy létrehozzak egy olyan rendszert ami megadja a készitőknek ezt a szabadságot, zenei szolgáltatásban. Igy egy olyan platformot nyújt a zenészeknek, ahol le tudják irni egy modellel a saját szerzeményüket, igy a hallgatóknak jobb elményt tud nyújtani a platform ajánlások szempontjából.

## 1.2. Megoldandó feladat leirása

A megoldandó feladat két részre bontható: egy kliens oldali webapplikációra és egy ezt kiszolgáló szerver oldali szolgáltatásra.

- Kliens oldali feladatok:
  - Felhasználók regisztrálása, azonositása
  - A tartalomfogyasztó userek és a tartalomkészitők megkülönböztetése
  - A tartalomkészitőknek lehetőséget biztositani a tartalom feltöltésére
  - A tartalomfogyasztóknak lehetőséget biztositani hogy meg tudják hallgatni a tartalmakat
  - A tartalomfogyasztóknak lehetőséget biztositani hogy megjelölják a tartalmakat amiket kedvelnek
- Szerver oldali feladatok:
  - Kiszolgálni a frontend alkalmazást RESTful[1] architectúrán és HTTP protokollon keresztül
  - A számokhoz tartozó tagek alapján meghatározni a hasonló számokat
  - Kapcsolatot létesiteni az adatbázissal
  - CRUD[2] műveletek biztositása

[1]REST - Representational State Transfer, állapotmentes kliens-szerver alapú szoftverarchitektúra típus

[2]Create - Read - Update - Delete

# 2. fejezet

# Felhasználói dokumentáció

A felhasználók sikeres bejelentkezés után az oldal főoldalával találják szembe magukat. Innen el tudnak indulni több irányba is. Ha a Track fülre kattint a navigációs sávban, akkor láthatja az összes zeneszámot, ami elérhető. Ha az Albumsra kattint, akkor láthatja az összes albumot, ha az Artistsra, akkor az összes előadót. Ezeken a menüpontokon belül lépegethet, játszhat le zeneszámot, adhat zeneszámot a lejátszási listájához. A lejátszási listáját megtekintheti, ha a Playlist menüpontot választja. Ha a lejátszási listájához készült ajánlásokat akarja megtekinteni, akkor pedig azokat elérheti a Discover fül alatt. Jobb oldalt a saját nevére kattintva a kinyiló menüből a Logoutot tudja kiválsztani, valamint a saját profilját, amit szerkeszthet.

## 2.1. Rendszerkövetelmények

### 2.1.1. Szerver oldali követelmények

A szerver oldali alkalmazás fejlesztése és tesztelése Windows környezetben, Javascript nyelven iródott Node futtatási környezetben. A konténerizált megoldásnak köszönhetően docker és docker-compose segitségével bármilyen operációs rendszeren futtatható, amely olyan hardveren fut, ami támogat virtualizációt.

Hardver:

1. Processzor: Bármilyen Intel vagy AMD processzor, ami támogatja a virtualizációt
2. Ram: 4GB vagy több
3. Háttértár: 40Gb vagy több

Operációs rendszer:

1. Windows 10
2. Linux
3. MacOS

Adatbázis:

1. MongoDB

Web szerver:

1. Webpack

Szükséges programok:

1. Node 8 vagy nagyobb LTS verzió
2. npm
3. Ha containerben akarjuk futtatni:
   (a) Docker
   (b) Docker-compose
   (c) Windows esetén: Windows Linux Subsystem

Szükséges javascript libraryk:

1. aws-sdk: 2.726.0
2. bcryptjs: 2.4.3
3. body-parser: 1.19.0
4. bson: 4.1.0
5. cors: 2.8.5
6. dotenv: 8.2.0
7. express: 4.17.1
8. express-fileupload: 1.1.6
9. express-oas-generator: 1.0.23
10. jsonwebtoken: 8.5.1
11. lodash: 4.17.15

12. mongoose: 5.9.4
13. mongoose-to-swagger: 1.1.1
14. morgan: 1.9.1
15. node-tfidf: 0.0.2
16. nodemailer: 6.4.4
17. nodemon: 2.0.0
18. querystring: 0.2.0
19. vector-object: 1.3.0

### 2.1.2. Kliens oldali követelmények

A kliens oldali alkalmazés fejlesztése és tesztelése Windowson történt Javascript nyelvben Vue.js keretrendszerben [1]. Google Chrome és Firefox böngészőre van optimalizálva.

Hardver:

1. Processzor: Bármilyen Intel vagy AMD processzor ami támogatja a virtualizációt
2. Ram: 4GB vagy több
3. Háttértár: 40Gb vagy több

Operációs rendszer:

1. Windows 10
2. Linux
3. MacOS

Támogatott böngészők:

1. Google Chrome
2. Firefox

Szükséges programok:

1. Node 8 vagy nagyobb LTS verzió
2. npm
3. Ha containerben akarjuk futtatni:
   (a) Docker
   (b) Docker-compose
   (c) Windows esetén: Windows Linux Subsystem

Szükséges javascript library-k:

1. axios: 0.19.2
2. bootstrap: 4.4.1
3. bootstrap-vue: 2.7.0
4. dotenv: 8.2.0
5. vue: 2.5.2
6. vue-router: 3.0.1

## 2.2. Telepités

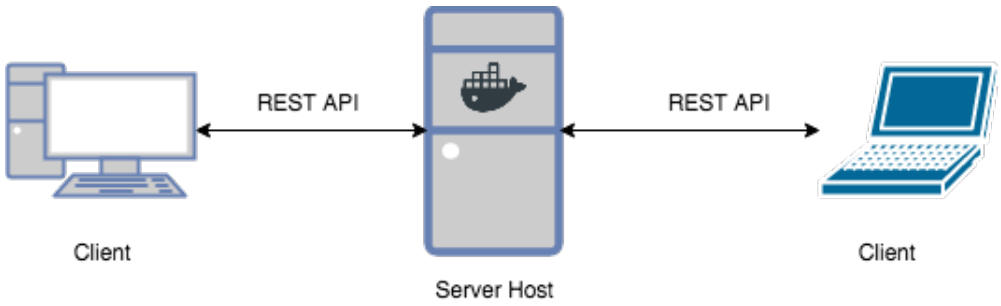
A kliens alkalmazás és a szerver oldali alkalmazás futtatása megoldható ugyanarról a gépről, egyetlen parancs kiadásával, ha konténerizálva szeretnénk futtatni. Ha több szerveres infrastruktúrát megfelelő előismeretekkel könnyedén ki tudjuk építeni.

### 2.2.1. Konténerizált futtatás

#### Docker

A Docker platform képes becsomagolni az alkalmazásunkat egy gyengén izolált környezetbe, amit konténernek hívunk [2]. Ezt a konténert könnyen el tudjuk szállítani a velünk dolgozó mérnököknek, vagy akár éles környezetbe is felküldhetjük annak a veszélye nélkül, hogy az alkalmazás környezete megváltozik és ezáltal nem funkcionál rendesen.

Az alapelemek Dockerben az image-ek és a konténerek. Az image-ek olyanok, mint egy objektum orientált nyelvben az osztályok. Leírják, hogy milyen egy konténer, tehát egy image példánya a konténer. De akár el tudnánk képzelni úgy is az image-eket, mint receptek, amik alapján a Docker képes konténereket létrehozni. Dockerfile-ba írjuk le az image felépítését, amiből az image-ek létrehozhatók. Az analógiát követve a Dockerfile leírás arról, hogy pontosan hogy kell megírni egy konkrét receptet. A snippet folytatásában, példákon keresztül jobban megérthetjük majd ezt.




Ahhoz, hogy a Docker működjön, szükségünk van egy hoszt gépre, amin egy docker daemon nevű program fut. Ennek a processznek köszönhetően futtathatóak a konténerek. Ezt a hoszt gépet tudjuk irányítani parancssoros klienssel, akár a hoszt gép termináljáról is és ezáltal fogjuk tudni kezelni a konténereket és az image-eket.

### Futtatás konténerizált környezetben

Első lépésként bekell konfigurálnunk a frontendet és a backendet.

Ezt a frontenden ugy tudjuk megtenni hogy a `frontend/config.js` fáljben beállitjuk a backend elérhetőségét. Ez konténerizált környezetben a localhost lesz.

A backenden a `backend/.env` fileban kell a következő parametereket megadnunk:

1. `PORT` - Ezt a portot fogja használni az backend szerverünk
2. `EMAIL_USER` - Az email szolgáltatás használatában lévő gmail-es email cim ahol engedelyezve vannak a kevésbé biztonságos alkalmazások
3. `EMAIL_PWD` - Az email szolgáltatás használatában levő gmail-es email cim jelszava
4. `SECRET` - Titkositó algoritmushoz használt secret
5. `DATABASE_URL` - Az adatbazis elérhetősége, ez konténerizált környezetben a `mongodb://database.db:27017/thesis` lesz
6. `FRONTEND_VERIFY_LINK` - A frontend host cime. Konténerizált esetben ez `http://localhost:8080` lesz

Példa `.env` file backenden:

```
PORT=3000
EMAIL_USER=nodejsemailbot123@gmail.com
EMAIL_PWD=Almafa123456
DATABASE_URL=mongodb://database.db:27017/thesis
FRONTEND_VERIFY_LINK=http://localhost:8080
SECRET=nagyontitkos1234
```

Majd ha ezekkel készen vagyunk utána kiadjuk a program gyökér könyvtárában a (ahol a `docker-compose.yml` fájl található) `docker-compose up --build` parancsot. Ez a következőket fogja csinálni:

1. Letölti a mongodb docker imaget
2. Átmásolja a backend alkalmazás forráskódját a docker containerbe
3. Feltelepiti a backendhez szükséges javascript könyvtárakat
4. Átmásolja a frontend alkalmazás forráskódját a docker containerbe
5. Feltelepiti a frontendhez szükséges javascript könyvtárakat
6. Elinditja az adatbázis, backend, frontend containereket

Amikor a terminálunkban meglátjuk azt hogy milyen cimen érhető el a frontend alkalmazásunk, onnan tudjuk hogy végzett a telepitesekkel és az inditással.

```
database_1  | {"t":{"$date":"2020-12-05T01:44:12.458+00:00"},"s":"I",  "c":"NETW
database_1  | {"t":{"$date":"2020-12-05T01:44:12.458+00:00"},"s":"I",  "c":"NETW
database_1  | {"t":{"$date":"2020-12-05T01:44:12.458+00:00"},"s":"I",  "c":"NETW
backend_1   |
backend_1   | > university-thesis-backend@1.0.0 start /usr/src/app
backend_1   | > nodemon -L index.js
backend_1   |
backend_1   | [nodemon] 2.0.4
backend_1   | [nodemon] to restart at any time, enter `rs`
backend_1   | [nodemon] watching path(s): *.*
backend_1   | [nodemon] watching extensions: js,mjs,json
backend_1   | [nodemon] starting `node index.js`
backend_1   | Server is listening on 3000
database_1  | {"t":{"$date":"2020-12-05T01:44:25.948+00:00"},"s":"I",  "c":"NETW
database_1  | {"t":{"$date":"2020-12-05T01:44:25.955+00:00"},"s":"I",  "c":"NETW
.1, LE (unified)","version":"3.6.0|5.10.2"}}}
database_1  | {"t":{"$date":"2020-12-05T01:44:25.966+00:00"},"s":"I",  "c":"NETW
backend_1   | Connected to the database at mongodb://database.db:27017/thesis
database_1  | {"t":{"$date":"2020-12-05T01:44:25.967+00:00"},"s":"I",  "c":"NETW
database_1  | {"t":{"$date":"2020-12-05T01:44:25.972+00:00"},"s":"I",  "c":"NETW
.1, LE (unified)","version":"3.6.0|5.10.2"}}}
database_1  | {"t":{"$date":"2020-12-05T01:44:25.977+00:00"},"s":"I",  "c":"NETW
.1, LE (unified)","version":"3.6.0|5.10.2"}}}
database_1  | {"t":{"$date":"2020-12-05T01:44:25.985+00:00"},"s":"I",  "c":"NETW
database_1  | {"t":{"$date":"2020-12-05T01:44:25.987+00:00"},"s":"I",  "c":"NETW
.1, LE (unified)","version":"3.6.0|5.10.2"}}}
database_1  | {"t":{"$date":"2020-12-05T01:44:36.470+00:00"},"s":"I",  "c":"NETW
database_1  | {"t":{"$date":"2020-12-05T01:44:36.471+00:00"},"s":"I",  "c":"NETW
.1, LE (unified)","version":"3.6.0|5.10.2"}}}
frontend_1  |  21% building modules 94/100 modules 6 active ...rc/app/src/compone
 95% emitting DONE  Compiled successfully in 41138ms1:45:15 AM
frontend_1  |
frontend_1  |  I  Your application is running here: http://localhost:8080
```

Itt már láthatjuk hogy a `http://localhost:8080`-on érjük el az alkalmazásunkat.

### 2.2.2. Nem konténerizált futtatás

Ha úgy döntünk hogy nem akarjuk a docker-compose segitségével futtatni, esetleg egyáltalán nem akarunk dockert használni akkor a következőkőkre lesz szükségünk:

1. Adatbázis szerverre
2. Backend alkalmazás szerverre
3. Frontend alkalmazás szerverre

#### Adatbázis szerver

A mongodb-t tudjuk saját magunk is telepiteni illetve bérelhetünk is dedikált clustereket különböző szolgáltatóktól (Pl MongoAtlas). Ha ugy döntünk saját magunk rakjuk össze az adatbázis szerverünket akkor a következőképen kell telepiteni

Windows:

1. Letöltjük a mongodb telepitőjét
2. Feltelepitjük és konfiguráljuk
3. Majd elinditjuk a hozzá tartozó windows servicet
4. Bővebb infó itt [3].

Linux:

1. Letöltjük a mongodb packaget a megfelő csomegkezelővel és telepitjük
2. Konfiguráljuk
3. Elinditjuk a hozzá tartozó servicet
4. Bővebb infó itt [4].

MacOS:

1. Letöltjük a mongodb packaget a megfelő csomegkezelővel és telepitjük
2. Konfiguráljuk
3. Elinditjuk a hozzá tartozó servicet
4. Bővebb infó itt [5].

[3] https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/
[4] https://docs.mongodb.com/manual/administration/install-on-linux/
[5] https://docs.mongodb.com/manual/tutorial/install-mongodb-on-os-x/

#### Szerver oldali alkalmazás

A szerver oldali alkalmazást bármilyen platformon eltudjuk inditani ha teljesülnek a következő feltételek:

1. Van feltelepitve node futtató környezet
2. Van feltelepitett npm csomagkezelőnk

A lépések a következőek:

1. Az `npm install` parancsal telepitjük a szükséges könyvtárakat
2. Bekonfiguráljuk a fentebb irt módon
3. Az `npm start` parancsal elinditjuk

#### Kliens oldali alkalmazás

A kliens oldali alkalmazást bármilyen platformon el tudjuk inditani, ha teljesülnek a következő feltételek:

1. Van feltelepitve node futtató környezet
2. Van feltelepitett npm csomagkezelőnk

A lépések a következőek:

1. Az `npm install` parancsal telepitjük a szükséges könyvtárakat
2. Bekonfiguráljuk a fentebb irt módon
3. Az `npm build` paranccsal leforditjuk
4. Valamilyen web server programmal hostoljuk (pl: webpack, apache) vagy feltöltjük a kapott dist-et valamilyen felhőszolgáltatásba (pl: amazon cloudfront)

## 2.3. Felhasználói felület bemutatása

### 2.3.1. Alkalmazás felépitése

1. Üdvözlő nézet
2. Bejelentkezési nézet
3. Regisztrációs nézet
4. Előadóként
   (a) Albumok szerkeztése nézet
   (b) Zeneszámok szerkeztése nézet

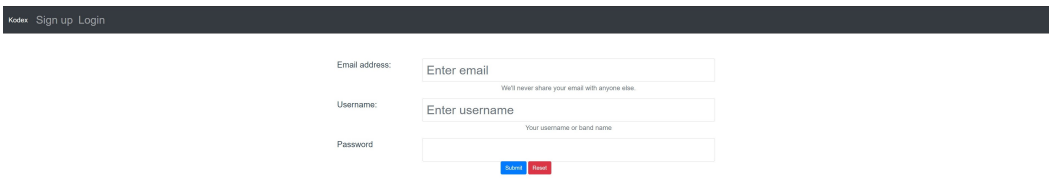
5. Felhasználóként
   (a) Albumok nézet
   (b) Előadók nézet
   (c) Számok nézet
   (d) Lejátszási nézet
   (e) Lejátszási lista nézet
   (f) Felfedezési nézet

### 2.3.2. Üdvözlő nézet

Az üdvőzlő nézetben csak néhány kedvcsináló képet látunk teljes képernyőben, a navigációs felső sávot és azon a Login és Sign up opciót.

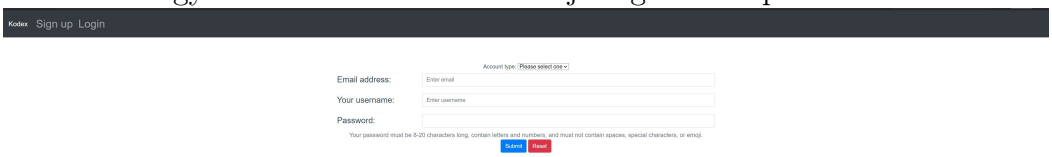
### 2.3.3. Bejelentkezési nézet

A bejelentkezési nézetben tudunk bejelentkezni amihez szükség van az email címünkre, felhasználó nevünkre és jelszavunkra. A `Submit` gombra kattintva tudjuk elküldeni a bejelentkezési kérelmünket a szerver fele, aminek sikerességét egy zöld toast üzenet jelzi, sikertelenségét pedig egy piros toast üzenet. A `Reset` gombbal tudjuk a mezőket alaphelyzetbe állitani. Bejelentkezés után a navigációs sávon megjelennek az elérhető menüsávok, attól függően milyen fiókkal léptünk be. Előadói fiókkal az `Albums` menüpont jelenik meg, felhasználói fiókkal pedig az `Artist`, `Albums`, `Tracks` menüpontok jelennek meg.



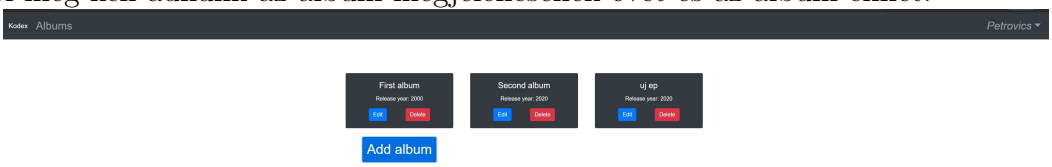

### 2.3.4. Regisztrációs nézet

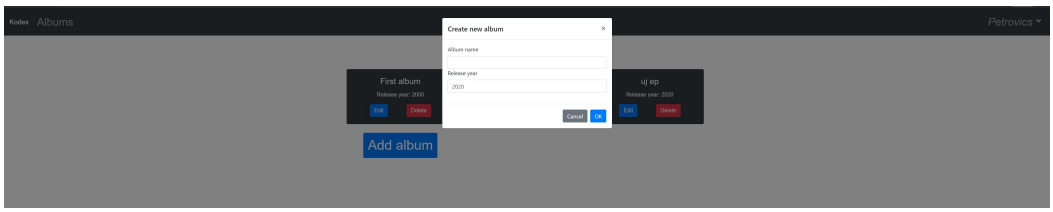
A regisztrációs nézetben tudunk regisztrálni az oldalra. Ehhez szükséges egy felhasználónév, egy email cim és egy jelszó, valamint kiválasztani, hogy felhasználói vagy előadói fiókot akarunk-e létrehozni. Az utóbbi esetben a felhasználónevünk megegyezik az előadói nevünkkel. A `Submit` gombra kattintva tudjuk elküldeni a regisztrációs kérelmünket a szerver fele, aminek sikerességét egy zöld toast üzenet jelzi, sikertelenségét pedig egy piros toast üzenet. A `Reset` gombbal tudjuk a mezőket alaphelyzetbe állitani. Sikeres regisztráció után egy megerősitő emailt fogunk kapni, ami tartalmaz egy linket amire kattintva tudjuk igazolni a profilunkat.




### 2.3.5. Albumok szerkesztése nézet

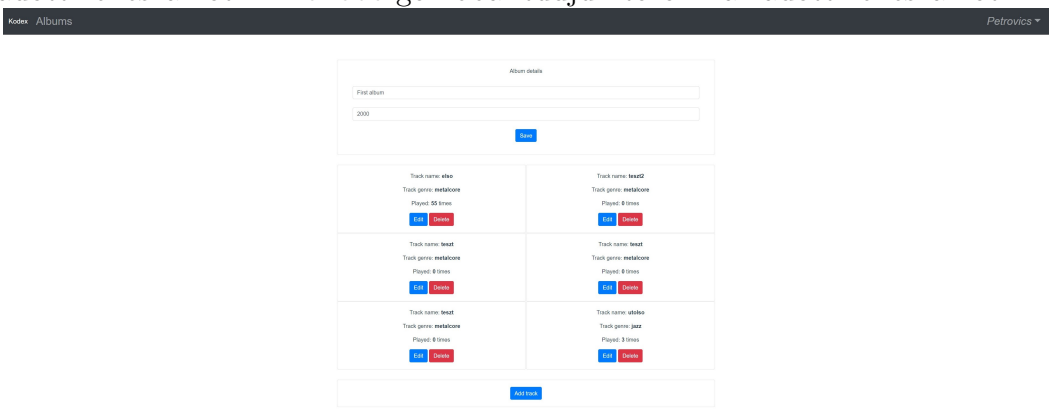
Itt tudja az előadó szerkeszteni és törölni meglévő albumjait, valamint újakat hozzáadni. Az `Edit` gombra kattintva tudjuk megnyitni az album szerkesztése nézetet. A `Delete` gombra kattintva tudjuk törölni az albumot. Az `Add album` gombra kattintva tudunk új albumot létrehozni. Erre rákattintva egy popup jelenik meg, ahol meg kell adnunk az album megjelenésének évét és az album címét.

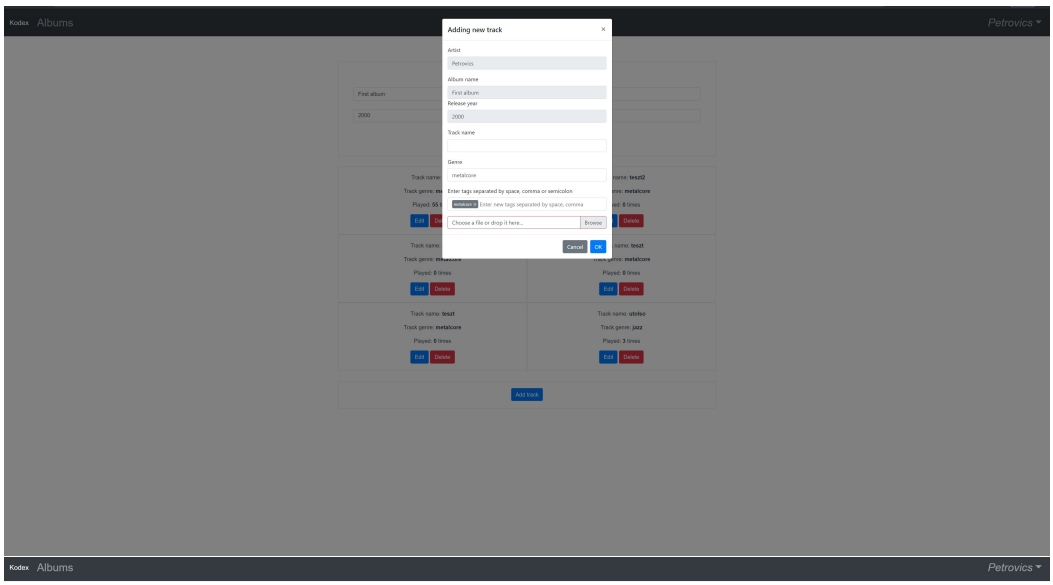


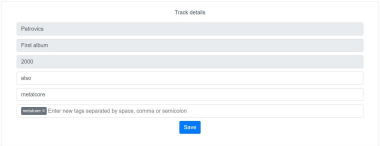



### 2.3.6. Zeneszámok szerkesztése

Itt tudja az előadó szerkeszteni és törölni a meglévő zeneszámjait, valamint újakat hozzáadni. Itt tudja szerkeszteni az album nevét és kiadási dátumat is. A `Save` gombra kattintva elküldjük a szerver felé a kérésünket arról hogy változtatni szeretnénk az albumon. Ennek sikeressét zöld toast jelzi, sikertelenségét piros. Tudunk valamint új zeneszámot felvenni. Itt meg kell adnunk a szám nevét, műfaját és a hozzá tartozó tageket, valamint a feltöltendő fáljt. A fájlokat biztonsági okokból nem a szerveroldalon tároljuk, hanem egy Amazon S3 bucketben, aminek segitségével kiküszöbölhető az, hogy esetlegesen futtassanak valamit a szerverünkön. Az `Ok` gombra kattintva elküldjük a kerésünket a szerver felé, sikeres kérés esetén megjelenik a legújabb feltöltött szám a listánkban. Az `Edit` gombra kattintva tudjuk szerkeszteni az adott zeneszámot. A `Delete` gombbal tudjuk törölni az adott zeneszámot.

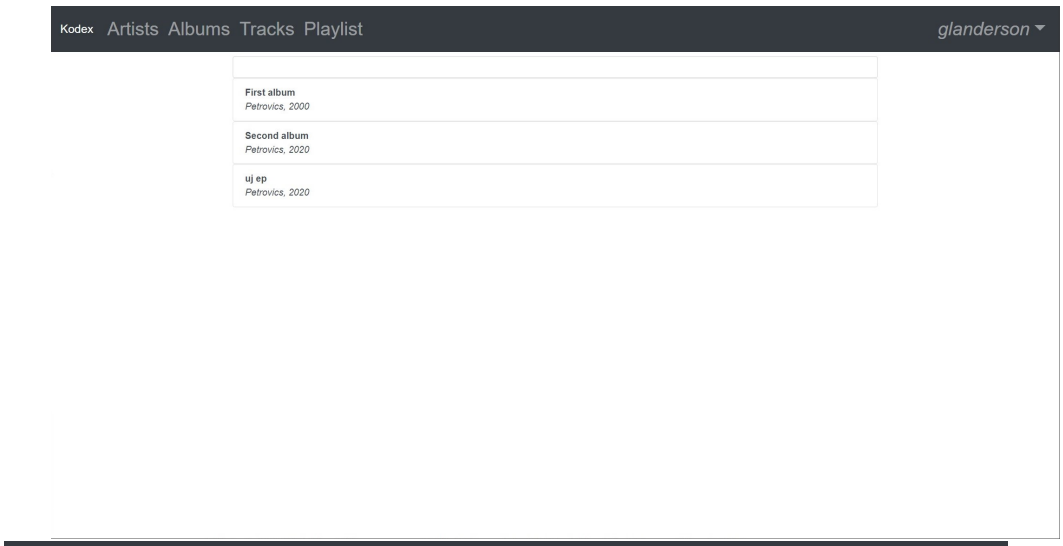


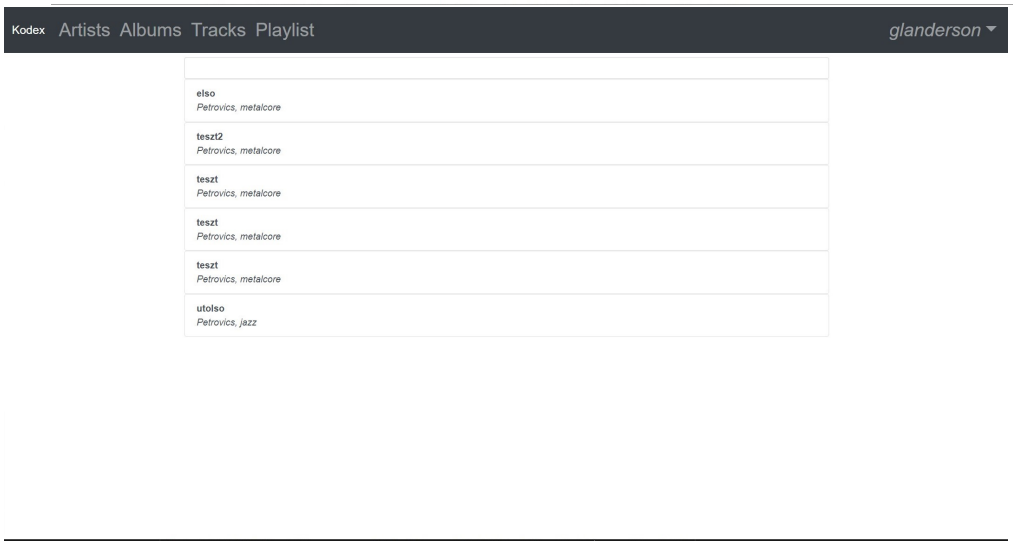



## 2.3.7. Albumok nézet

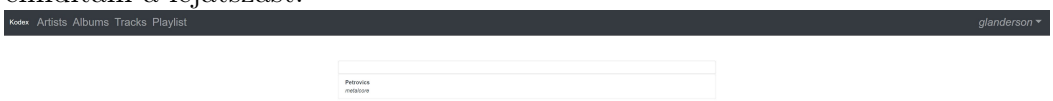
Az albumok nézetben egy listát látunk az összes albumról. A listában láthatjuk az album címét, előadóját és a kiadás dátumát. A felső részen találhatunk egy kereső inputot, ahova begépelve a keresendő nevet, egyből visszaadja az eredményeket. Csak az album címében és az előadó nevében keres. Ha rákattintunk egy albumra, akkor a következő oldalon az albumhoz tartozó számokat tudjuk megtekinteni. A számra kattintva tudjuk elindítani a lejátszást.

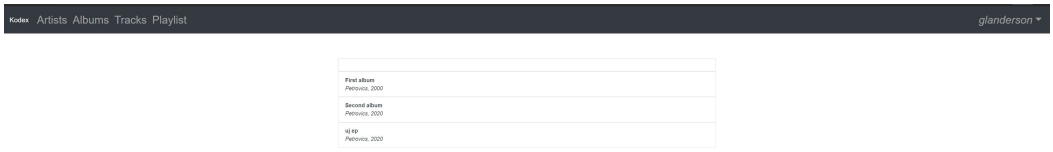


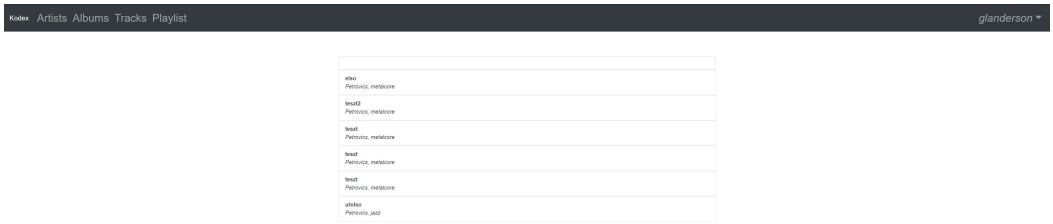




### 2.3.8. Előadók nézet

Az előadók nézetben egy listát kapunk az összes előadóról. A listában látjuk az előadók nevét és a stílust, amit képviselnek. A kereső sávban tudunk az előadó nevére keresni. Az előadóra rákattintva láthatjuk az előadóhoz tartozó albumokat, az albumra kattintva az albumhoz tartozó számokat. A számra kattintva tudjuk elindítani a lejátszást.






### 2.3.9. Számok nézet

A számok nézetben az összes feltöltött számot látjuk. A listában láthatjuk a szám nevét, előadóját és stilusát. A kereső sávban tudunk a szám nevére keresni. A számra rakattintva elindithatjuk a lejátszást.

### 2.3.10. Lejátszási nézet

A lejátszási listában a tudjuk elinditani az adott számot, tudunk beletekerni, hangerőt állítani, hozzáadni a lejátszási listához, kivenni a lejátszási listából. Az oldal bal oldalán láthatjuk az ajánlatokat egyezési sorrendben. Ami legfelül van az egyezik legjobban a hallgatott számmal. Ha a szám benne van a lejátszási listánkban, akkor piros `Remove` gombot látunk, ha nincs, akkor kék `Add` gombot látunk.

### 2.3.11. Lejátszási lista nézet

Itt láthatjuk azokat a számokat, amiket beraktunk a lejátszási listába, a felfedezési nézetben az ezekhez tartozó ajánláso-

kat fogjuk látni. A számra kattintva elindíthatjuk a lejátszást.

### 2.3.12. Felfedezési nézet

A lejátszási listából generált ajánlásokat láthatjuk itt. A számra kattintva elindíthatjuk a lejátszást.

# 3. fejezet

# Fejlesztői dokumentáció

## 3.1. Fejlesztői környezet

A fejlesztést és a tesztelést teljes egészében Windows 10 operációs rendszeren végeztem. A kód irásához Visual Studio Code szövegszerkesztő ajánlott a következő pluginokkal:

1. ESLint
2. Docker
3. JavaScript (ES6) code snippets
4. npm
5. npm Intellisense
6. Vetur
7. Visual Studio IntelliCode
8. Beautify

A fejlesztéshez szükségesek még a következő programok:

1. Node 8 vagy nagyobb LTS verzió
2. npm
3. Ha containerben akarjuk futtatni:
   (a) Docker
   (b) Docker-compose
   (c) Windows esetén: Windows Linux Subsystem

## 3.2. Szerver oldal

### 3.2.1. Specifikáció

Szerver oldali tervezésnél fő szempont volt az egyszerű bővíthetőség lehetősége, és könnyű kezelés lehetősége, ezzel elősegítve annak a lehetőségét, hogy a jövőben akár más is sikeresen tudja módosítani a szerver működését, illetve azt további modulokkal lehessen bővíteni. Az alkalmazás készítése továbbá a következő elvárások figyelembe vételével készült:

1. A szerver HTTP protokollon keresztül szolgálja ki a hivásokat
2. Képes legyen kiszolgálni RESTful interfacen való kommunikációt a klienstől
3. Kommunikáció JSON formátumban kell, hogy történjen
4. A szerveren található perzisztens réteg könnyen frissithető és kicserélhető legyen
5. Az adatbázis réteg legyen elrejtve a REST interface elől
6. Az adatbázis nem tárolhat semmilyen állapot információt a kliens oldaláról két lekérés között
7. A szerver alkalmazzon authentikációt a felhasználók authentikálására
8. Nyújtson lehetőséget új felhasználók regisztrálására és azok authentikálására
9. Nyújtson lehetőséget a felhasználóknak fájl feltöltésre
10. Valamilyen authentikációs standarddel védje a végpontokat
11. A program külső adminisztráció nélkül, magától működjön
12. A program a logolást a standard outputra küldje, a logok adminisztrációja nem a program feladata
13. A RESTful API végpontjainak kezelését és a hozzájuk tartozó logikát különböző szoftverrétegekben kell megvalósítani

### 3.2.2. Használt módszerek és könyvtárak

A szerver oldali programot teljes egészében JavaScript nyelven írtam, node futtatási környezetben. Használt JavaScript library-k:

1. aws-sdk 2.726.0: Ahhoz használjuk hogy a feltöltött fileokat AWS S3 bucketban tároljuk. Ennek segitségével tudjuk kikerülni a fájl feltöltésen keresztül támadókat.

2. bcryptjs 2.4.3: Jelszó titkositáshoz használjuk.
3. body-parser 1.19.0: Request body adattagját tudjuk vele JSON-né parsolni
4. bson 4.1.0: MongoDB schemák leirása
5. cors 2.8.5:
6. dotenv 8.2.0: .env file parsoláshoz használjuk.
7. express 4.17.1: A backend framework, ami a kérések kezelését végzi.
8. express-fileupload 1.1.6: Az expresshez szükséges modul ami fájl feltöltésért felelős.
9. express-oas-generator 1.0.23: Swagger generálásához használt modul
10. jsonwebtoken 8.5.1: Authentikációhoz használt generátor modul
11. lodash 4.17.15: Array műveletekhez használt modul
12. mongoose 5.9.4: Adatbázis műveletekért felelős modul
13. mongoose-to-swagger 1.1.1: Schemak leirása
14. morgan 1.9.1: Beérkező requestek logolása
15. node-tfidf 0.0.2: Az ajánlórendszerhez használt library
16. nodemailer 6.4.4: Emailekhez használt library
17. nodemon 2.0.0: Futtatáshoz használt library
18. querystring 0.2.0: Beérkező HTTP query-k parsolására használt library
19. vector-object 1.3.0: Az ajánlórendszerhez használt library

### 3.2.3. Adatbázis

Az adatok tárolására MongoDB adatbázist használtam. Az adatbázis kapcsolatot a programból a mongoose library segítségével oldottam meg. A MongoDB egy NoSQL adatbázis, azért esett erre a választás mivel:

1. Könnyű javascript objektumokká parsolni
2. Dinamikus schemaval rendelkezik
3. Egyszerű query-kre van szükségünk

### 3.2.4. Adatbázis sémák

Az SQL adatbázisokban az adatok elrendezését ahhoz lehetne hasonlítani, mintha lenne egy vectornyi tuple-ünk, ahol a tuple-öknek fix hossza van. Ebből adódóan nem tudunk új adattagot belerakni. A NoSQL-ben ez úgy van megvalósítva mintha egy vectornyi jsonünk lenne. Ezek a collectionök.

**Felhasználók collection**

| Users | |
|---|---|
| _id | ObjectId |
| username | String |
| password | String |
| email | String |
| userType | userType |
| genre | String |
| isVerified | Boolean |
| createdAt | Date |

A felhasználó collection schemája tartalmazza a felhasználók adatait. Egy felhasználó a következő adattagokkal rendelkezik:

1. `_id` : Egy ObjectId, a MongoDB magától generálja és állitja be
2. `username` : A felhasználó felhasználóneve, előadónak a banda neve
3. `password` : A felhasználó jelszava, ami titkositva kerül az adatbázisba
4. `email` : A felhasználó email cime. Az email cim formai helyessége regexel van ellenőrizve.
5. `userType` : Ez lehet artist, vagy user. Attól függetlenül hogy a felhasználó csak tartalomfogyasztó vagy készitő
6. `genre` : Ha a felhasználó artist, akkor a stilus, amit képvisel, egyébként üres
7. `isVerified` : Ezzel ellenőrizzük hogy a felhasználó aktiválta-e már a fiókját

8. `createdAt` : A dátum, ami azt jelöli, mikor jött létre a felhasználó

Ugyanez a kódban:

```

'use strict';

const mongoose = require('mongoose');
var Schema = mongoose.Schema;

var UserSchema = new Schema({
    username: {
        type: String,
        unique: true,
        required: 'Username is required'
    },
    password: {
        type: String,
        required: 'Password is required'
    },
    email: {
        type: String,
        unique: true,
        trim: true,
        match: /^\w+([.-]?\w+)*@\w+([.-]?\w+)*(\.\w{2,3})+$/,
        required: 'Email is required'
    },
    userType: {
        type: String,
        enum: ['user', 'artist'],
        required: 'User type is required'
    },
    genre: {
        type: String,
        default: 'None'
    },
    isVerified: {
        type: Boolean,
        default: false
    },
```

```
    createdAt: {
        type: Date,
        default: Date.now()
    }
});

module.exports = mongoose.model('Users', UserSchema);
```

3.1. forráskód. Users schema

### Albumok collection

| Albums | |
|---|---|
| _id | ObjectId |
| artist | String |
| name | String |
| releaseDate | Integer |

Az album collection schemája tartalmazza az albumok adatait. Egy album a következő adattagokkal rendelkezik:

1. `_id` : Egy ObjectId, a MongoDB magától generálja és állítja be
2. `artist` : Az előadó neve
3. `name` : Az album neve
4. `releaseYear` : A kiadás dátuma

Ugyanez a kódban:

```

'use strict';

const mongoose = require('mongoose');
var Schema = mongoose.Schema;

var AlbumSchema = new Schema({
    artist: {
        type: String,
```

```
10          required: 'Artist required!',
11      },
12      name: {
13          type: String,
14          required: 'Album name required!',
15      },
16      releaseYear: {
17          type: Number,
18          required: 'Year is required!',
19      },
20  });
21
22  module.exports = mongoose.model('Albums', AlbumSchema);
```

3.2. forráskód. Albums schema

**Zeneszám collection**

| Tracks | |
|---|---|
| _id | ObjectId |
| artist | String |
| album | String |
| releaseYear | Integer |
| genre | String |
| tags | [String] |
| trackName | String |
| trackPath | String |
| played | Integer |

A zeneszám collection schemája tartalmazza a zeneszámok adatait. Egy zeneszám a következő adattagokkal rendelkezik:

1. `_id` : Egy ObjectId, a MongoDB magától generálja és állítja be
2. `artist` : Az előadó neve
3. `name` : Az album neve
4. `releaseYear` : A kiadás dátuma
5. `genre` : A szám stilusa
6. `tags` : A számhoz használt tagek
7. `trackName` : A szám neve
8. `trackPath` : A szám elérési útja az AWS S3 Bucketen belül
9. `played` : Az, hogy hányszor játszották le a számot

Ugyanez a kódban:

```

'use strict';

const mongoose = require('mongoose');
var Schema = mongoose.Schema;

var TrackSchema = new Schema({
    artist: {
        type: String,
        required: 'Artist required'
    },
    album: {
        type: String,
        required: 'Album required'
    },
    releaseYear: {
        type: Number,
        required: 'Release year required'
    },
    genre: {
        type: String,
        required: 'Genre required!'
    },
    tags: {
        type: [String],
        required: 'Tags required!',
    },
```

```
    trackName: {
        type: String,
        required: 'Track name required'
    },
    trackPath: {
        type: String,
        default: ""
    },
    played: {
        type: Number,
        default: 0
    }
});

module.exports = mongoose.model('Tracks', TrackSchema);
```

3.3. forráskód. Tracks schema

### Lejátszási lista collection

A zeneszám collection schemája tartalmazza a zeneszámok adatait. Egy zeneszám a következő adattagokkal rendelkezik:

1. `_id` : Egy ObjectId, a MongoDB magától generálja és állítja be
2. `userId` : A felhasználó `_id`-ja
3. `items` : Song objectumokat tárol

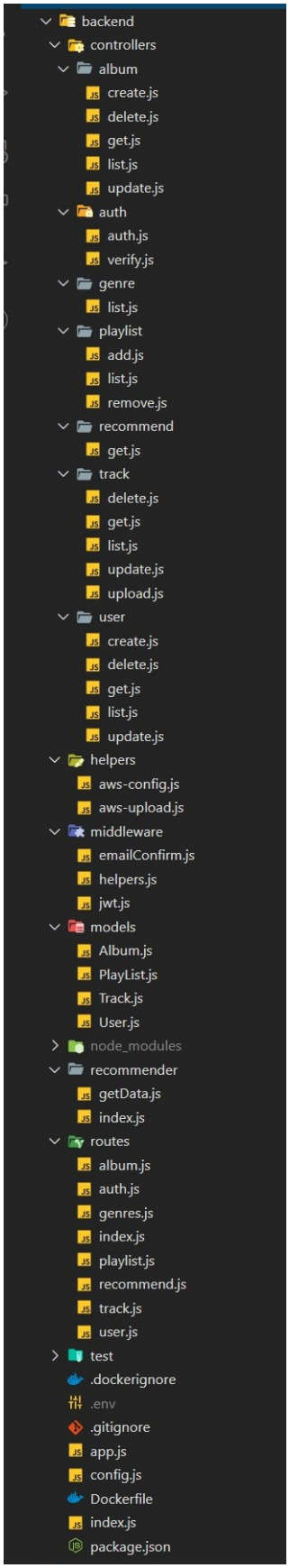
Ugyanez a kódban:

```

'use strict';

```

```
4 const mongoose = require('mongoose');
5 var Schema = mongoose.Schema;
6
7 var PlayListSchema = new Schema({
8     userId: {
9         type: String,
10        unique: true,
11        required: 'UserId is required!'
12    },
13    items: {
14        type: [{}]
15    }
16 });
17
18 module.exports = mongoose.model('Playlists', PlayListSchema);
```

3.4. forráskód. Playlists schema

### 3.2.5. Project struktúra



A project gyökerében találhatóak a docker és gitignore fájlok, amik azért felelnek, hogy ne töltsünk fel nem kivánt dolgokat a Docker image-be, vagy a git verziókezelőbe. Itt található továbba a `config.js` és a `.env` fájlok, amik a beállitások felparsolásáért felel. A `package.json`-ben találhatóak az inditó scriptek és a szükséges package-ek.

### Controllerek

A controllerek a program azon részei, amik a bejövő requestet kezelik és összerakják a választ a kliensnek. Ezek a következőképpen épülnek fel:

1. Egy függvényt, aminek egyik paramétere a request, egy masik a response
2. Itt valósulnak meg az adatbázis műveletek
3. Itt épül fel a response body

Példa egy controllerre:

```

'use strict';

const emailConfirm = require('../../middleware/emailConfirm');
const config = require('../../config');
const bcrypt = require('bcryptjs');
var User = require('../../models/User');
var PlayList = require('../../models/PlayList');

function createUser(req, res) {
    var new_user = new User(req.body);
    var salt = bcrypt.genSaltSync(5);
    var hash = bcrypt.hashSync(req.body.password, salt);
    new_user.password = hash;

    new_user.save((err, user) => {
        if(err) {
            console.log(err);
        }
        emailConfirm.sendConfirmationMail(user, config.
            frontendVerifyLink);
    });

```

```
    var playlist = new PlayList({userId: new_user._id});
    playlist.save();

    res.json({registration: 'success'});
};

module.exports = createUser;
```

3.5. forráskód. User create controller

Példa async controllerre:

```

'use strict';

var Album = require('../../models/Album');
var Track = require('../../models/Track');

async function updateAlbum(req, res) {
    var album = await Album.findOne({
        _id: req.params.id
    });

    const tracks = await Track.updateMany({
        album: album.name
    }, {
        "$set": {
            album: req.body.name
        }
    });

    album = await Album.findOneAndUpdate({
        _id: req.params.id
    }, req.body);

    res.json(album)
};

module.exports = updateAlbum;
```

3.6. forráskód. Album update controller

### Helperek

A projectben egyetlen egy helper található, az `emailConfirm.js`, ami arra szolgál, hogy létrehoz egy nodemailer instance-t, amin keresztül kiküldi a megerősitő emailt. Az email egy linket tartalmaz a frontendre ami meghivja a verification endpointot.

### Middleware

A middlewarebe tartozik a `aws-config.js`, `aws-upload.js` és `jwt.js`

Az `aws-config.js`-ben található az aws instance es a hozzá tartozó kulcsok. Az `aws-upload.js`-ben található az a függvény, ami azért felel, hogy feltöltse a fájlt az AWS S3 bucketba. Azért lett az AWS S3 saját szerver helyett, mivel igy ki tudjuk küszöbölni a lehetséges támadásokat ami fájlfeltöltés léphet fel. Ezt az AWS már megcsinálta helyettünk. Így feltöltött fájl nem kerül tárolásra a szerveren, mivel a requestből szedi ki a fájlt és adja tovább az awsnek. Itt menti el a trackPath-et, ami az aws linkjére mutat. Az egyező fájlneveket úgy tudjuk elkerülni, hogy az aktuális dátum unix timestampjét hozzáfűzzük az eredeti fájl nevéhez.

```

const awsConfig = require('./aws-config');
var Track = require('../models/Track');

const uploadFile = (req, res) => {
    console.log(req.files.track)
    const params = {
        Bucket: 'kodex-data',
        Key: Date.now() + req.files.track.name,
        Body: req.files.track.data,
        ACL: 'public-read',
    };


    awsConfig.upload(params, function(err, data) {
        if (err) {
            throw err;
        }
```

```
19          console.log(‘File uploaded successfully. ${data.Location}‘)
                ;
20          console.log(req.body.trackData)
21          var track = new Track(JSON.parse(req.body.trackData));
22          track.trackPath = data.Location;
23
24          track.save((err, track) => {
25              if(err) {
26                  console.log(’Error’);
27              }
28              console.log(track);
29          });
30
31          res.json({
32              success: true,
33              message: ‘File uploaded successfully!‘,
34              track_id: track._id,
35          });
36      });
37
38  };
39
40  module.exports = uploadFile
```

3.7. forráskód. AWS config

A `jwt.js`-ben hoztuk létre a `checkToken` és `generateToken` függvényeket. Ezek felelnek azért hogy generáljunk JWT-t és ellenőrizzük azt. A JWT egy egyszerű tokenizációs egyszerű arra használunk hogy az endpointokat csak authentikált felhasználók tudják megtekinteni. A json web tokennek tudunk beállitani szavatosságot, egyedi aláirással tudjuk ellátni.

```
1
2  const jwt = require(’jsonwebtoken’);
3  const config = require(’../config’);
4
5  exports.checkToken = (req, res, next) => {
6      let token = req.headers[’x-access-token’] || req.headers[’
           authorization’];
7      if(token != undefined && token.startsWith(’Bearer ’)) {
```

```
8           token = token.slice(7, token.length);
9       }
10
11      if(token) {
12          jwt.verify(token, config.secret, (err, decoded) => {
13              if(err) {
14                  return res.json({
15                      success: false,
16                      message: 'invalid token'
17                  });
18              }
19
20              req.decoded = decoded;
21              next();
22          });
23      } else {
24          console.log('Auth not supplied');
25      }
26  }
27
28  exports.generateToken = (payload) => {
29      return jwt.sign(
30          { payload },
31          config.secret,
32          { expiresIn: '168h' }
33      )
34  }
35
36  module.exports = uploadFile
```

3.8. forráskód. JWT middleware

### Models

A modelleket a `mongoose` schémákkal irjuk le, majd definiáljuk az adatbázisban az API segitségével. Az objektumokat ezek után egyszerűen javascript objektumként tudjuk kezelni. Példa egy schémára:

```
1
2  'use strict';
```

```
3
4 const mongoose = require('mongoose');
5 var Schema = mongoose.Schema;
6
7 var TrackSchema = new Schema({
8     artist: {
9         type: String,
10        required: 'Artist required'
11    },
12    album: {
13        type: String,
14        required: 'Album required'
15    },
16    releaseYear: {
17        type: Number,
18        required: 'Release year required'
19    },
20    genre: {
21        type: String,
22        required: 'Genre required!'
23    },
24    tags: {
25        type: [String],
26        required: 'Tags required!',
27    },
28    trackName: {
29        type: String,
30        required: 'Track name required'
31    },
32    trackPath: {
33        type: String,
34        default: ""
35    },
36    played: {
37        type: Number,
38        default: 0
39    }
40 });
41
```

```
module.exports = mongoose.model('Tracks', TrackSchema);
```

3.9. forráskód. Tracks schema

### Recommender

A recommender modulban három fő függvény található. Első a `getData.js` amiben található a `getData` függvény. Ez a függvény ad vissza egy arrayt amiben benne vannak a számok `ObjectId`-ja, és a hozzá tartozó tagek egy sztringként.

```

'use strict';

const Track = require('../models/Track');

async function getData() {
    var tracks = await Track.find();
    const data = tracks.map(track => ({
        id: track._id,
        tags: track.tags.join(" ")
    }))

    return tracks;
}

module.exports = getData;
```

3.10. forráskód. getData.js

A `index.js`-ben található a másik két fő függvény. A `getRecommendations` egy `trackObject`-hez adja vissza a hozzáillő ajánlásokat, mig a `getRecommendationsByPlaylist` egy lejátszási listához adja vissza az összes ajánlást. Ez a függvény hívja az előzőt minden egyes `trackObject`-re. A `getRecommendations`függvényben inicializáljuk a `tfidf`-et, beadjuk neki az inputot, majd kivesszük a duplikált objectet.

```
1
2 async function getRecommendations(trackObject) {
3     const tfidf = new TfIdf();
4     const ids = [];
5     let recommendations = [];
6     const data = await getData();
7
8     for(let i = 0; i < data.length; ++i) {
9         tfidf.addDocument(data[i].tags);
10        ids.push(data[i].id)
11    }
12
13    tfidf.tfidfs(trackObject[0].tags.join(" "), function(i, score)
         {
14        recommendations.push({id: i, score: score})
15    });
16
17    for(let i = 0; i < recommendations.length; ++i) {
18        recommendations[i].id = ids[i];
19    }
20
21    var resp = [];
22    for(let i = 0; i < data.length; ++i) {
23        resp.push(
24            { song: data[i], score: recommendations[i].score }
25        );
26    }
27
28    resp = resp.filter((item) => {
29        return (String(trackObject[0]._id) !== String(item.song._id
             ));
30    })
31    return resp.sort((a, b) => b.score - a.score);
32 }
```

3.11. forráskód. getRecommendations függvény

A `getRecommendationsByPlaylist` függvényben használjuk az előzőleg megírt függvényt majd kivesszük belőle a duplikátumokat és a lejátszási listához tartozó számokat.

```

async function getRecommendationsByPlaylist(tracks) {
    var recommendations = [];
    console.log(Array.from(tracks))
    for(let track of tracks) {
        recommendations.push(...await getRecommendations([track]));
    }

    // Remove duplicates
    recommendations = Array
        .from(
            new Set(recommendations.map(a => String(a.song._id))))
        .map(_id => {
            return recommendations
                .find(a => String(a.song._id) === _id)
        });

    // Remove elements which is in the playlist
    for(let i = 0; i < recommendations.length; ++i) {
        for(let j = 0; j < tracks.length; ++j) {
            if(recommendations[i] && (String(recommendations[i].
               song._id) === String(tracks[j]._id))) {
                recommendations.splice(i, 1);
            }
        }
    }

    return recommendations
}
```

3.12. forráskód. getRecommendationsByPlaylist függvény

### TF-IDF

Az ajánláshoz a `TF-IDF` algoritmust használom. Teljes nevén: **term frequency–inverse document frequency** ami magyarra fordítva **kifejezés gyakoriság – fordított dokumentum gyakoriság**ot jelenti. Ez egy olyan keresőoptimalizálási technológiát takar, amikor más szövegek elemzésével ismerjük meg azokat a kifejezéseket, melyeket nekünk is használnunk kell a saját tartalmainkban. 2015-ben a szöveg alapú ajánlórendszerek 83%-a a `TF-IDF` algoritmust használta [3].

### TF-IDF matematikai háttere

A `TF-IDF` két statisztika, a frekvencia kifejezés és az inverz dokumentum gyakoriságának szorzata.

**Frekvencia kifejezés**

A frekvencia kifejezés esetében $tf(t,d)$, a legegyszerübb az ha használjuk a *raw count*-ját a kifejezésnek a dokumentumban. Ha a $t$ kifejezés előfordulása a dokumentumban $d$, ha a *raw count*-ot $f_{t,d}$-vel jelöljük. Alkalmazzuk a $tf(t,d) = f_{t,d}$ sémát.

$$tf(t,d) = 0.5 + 0.5 \cdot \frac{f_{t,d}}{max\{f_{t',d} : t' \in d\}}$$

**Inverz dokumentumok gyakorisága**

Az *inverz dokumentumok gyakorisága* a mértéke annak hogy mennyi információt biztosítani nekünk a szavak. Ez logaritmikusan méretezett inverz töredékét adja vissza a dokumentumnak ami tartalmazza a szót.

$$idf(t,D) = log\frac{N}{|\{d \in D : t \in d\}|}$$

ahol

1. $N$ a korpuszban szereplő összes dokumentum $N = |D|$
2. $|\{d \in D : t \in d\}|$ : a dokumentumok száma ahol a $t$ kifejezés megjelenik (tegyük fel $tf(t,d) \neq 0$). Ha a kifejezés nincs a korpuszban, az nullával való osztáshoz vezethet. Ezért az általános nevezőt a következőhoz igazítjuk: $1 + |\{d \in D : t \in d\}|$

#### Kifejezés gyakoriság – Fordított dokumentum gyakoriság

Ezután a TF-IDF-et a következőképpen számoljuk ki:

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D)$$

### Routes

A route-ok mondják meg, hogy az adott request urljére melyik controller hivódjon meg, milyen paramétereket adódjanak át az URLből. A routes mappában található az összes routingot megvalósító fájl. A route-okat az `index.js`-ben található.

```

'use strict';

const auth = require('./auth');
const track = require('./track');
const user = require('./user');
const album = require('./album');
const genres = require('./genres');
const playlist = require('./playlist');
const recommend = require('./recommend');

const elements = {
  auth,
  track,
  user,
  album,
  genres,
  playlist,
  recommend
};

function init(app, express) {
  const router = express.Router();

  Object.getOwnPropertyNames(elements).forEach(route => {
    router.use(`/${route}`, elements[route](app, express));
  });
```

```

  app.use('/', router);
}

module.exports = init;
```

3.13. forráskód. routes/index.js

Miután require-rel include-olom a megfelelő route-okat, utána hozzáadom azt az elements listához. Utaná az init függvényben létrehoz egy express router objectumot ahol hozzáfűzi a route-okat az objectumhoz. Az endpointok megegyeznek a route-ok változóneveivel. Később az `app.js`-ben inicializálom a route-jaimat.

Példa egy routera:

```

'use strict';

const jwt = require('../middleware/jwt');

module.exports = (app, express) => {
    const router = express.Router();

    router.get('/', jwt.checkToken, require('../controllers/user/
        list'));
    router.post('/', require('../controllers/user/create'));

    router.get('/:id', jwt.checkToken, require('../controllers/user
        /get'));
    router.put('/:id', jwt.checkToken, require('../controllers/user
        /update'));
    router.delete('/:id', jwt.checkToken, require('../controllers/
        user/delete'));

    return router;
};
```

3.14. forráskód. routes/user.js

Itt úgyszint létrehozunk egy express router objectumot, ahol beállitjuk az endpointokat és az endpointok tipusait is. Itt tudjuk beállitani a jwt `checkToken`

midleware-jét is. Ahol ezt beállítjuk azt nem tudjuk elérni érvényes JWT token nélkül.

### 3.2.6. RESTful interfész

| HTTP metódus | URL | Paraméter | Body | Válasz | JWT |
|---|---|---|---|---|---|
| POST | /auth | - | User object | Bearer token | Nem |
| GET | /auth/verify | id | - | Result message | Nem |
| GET | /track | - | - | All tracks obj. | Igen |
| GET | /track | - | id | Track object | Igen |
| POST | /track | - | Track object | Upload result | Igen |
| PUT | /track | id | Track object | Result message | Igen |
| DELETE | /track | id | - | Result message | Igen |
| GET | /user | - | - | All user obj. | Igen |
| POST | /user | - | User objet | Result message | Nem |
| GET | /user | id | - | User object | Igen |
| PUT | /user | id | User object | Result message | Igen |
| DELETE | /user | id | - | Result message | Igen |
| GET | /album | - | - | All album obj. | Igen |
| POST | /album | - | Album object | Result message | Igen |
| GET | /album | id | - | Album object | Igen |
| PUT | /album | id | Album object | Result message | Igen |
| DELETE | /album | id | - | Result message | Igen |
| GET | /genres | - | - | Genre list | Igen |
| GET | /playlist | id | - | Playlist object | Igen |
| PUT | /playlist | id | User object | Result message | Igen |
| DELETE | /playlist | id | - | Result message | Igen |
| GET | /rec | id | - | Recommendations | Igen |
| GET | /rec/pl | id | - | Recommendations | Igen |
| GET | / | - | - | Status message | Nem |

### 3.2.7. Applikáció modul

Az applikáció modul az `app.js`-ben található. Itt konfiguráljuk be az alkalmazásunk alapvető dolgait, mint például a route-ok, adatbázis kapcsolat. Az applikációs modulunkban a következő komponenseket regisztráljuk, inicializáljuk:

1. Létrehozzuk az express appot
2. Inicalizáljuk a swaggert
3. Létrehozzuk és beállitjuk az adatbázis kapcsolatot
4. Regisztráljuk a fájl upload komponenst az expresshez
5. Regisztráljuk a body parseoló, morgen, CORS komponenst az expresshez
6. Regisztráljuk a routejainkat az expresshez
7. Regisztráljuk a különálló status routeot az expresshez

```

const config = require('./config');
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const fileUpload = require('express-fileupload');
const morgan = require('morgan');
const cors = require('cors');
const expressOasGenerator = require('express-oas-generator');

const routes = require('./routes');

const app = express();
expressOasGenerator.init(app, {});

mongoose.Promise = global.Promise;

mongoose.connect(config.databaseUrl, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
    useCreateIndex: true
}).then(() => {
    console.log(`Connected to the database at ${config.databaseUrl
        }`);
}).catch((err) => {
```

```
25     console.log('Can not connect to the database! ' + err);
26 });
27
28 app.use(fileUpload({
29     createParentPath: true
30 }));
31
32 app.use(bodyParser.urlencoded({
33     extended: true
34 }));
35
36 app.use(bodyParser.json());
37 app.use(morgan('dev'));
38 app.use(cors())
39
40 routes(app, express);
41
42 app.route('/')
43     .get((req, res) => {
44         res.json({ status: 'Ok'});
45     });
46
47 module.exports = app;
```

3.15. forráskód. app.js

Ezek után exportáljuk az app modulet és az `index.js`-ben elinditjuk a szerverünket a bekonfigurált porton.

```
1
2 'use strict'
3
4 const app = require('./app');
5 const config = require('./config');
6
7 app.listen(config.port, () => {
8     console.log(`Server is listening on ${config.port}`);
9 });
```

3.16. forráskód. app.js

## 3.3. Kliens oldal

### 3.3.1. Specifikáció

Kliens oldali tervezésnél fő szempont volt az egyszerű bővíthetőség lehetősége,és könnyű kezelés lehetősége, ezzel elősegítve annak a lehetőségét, hogy a jövőben akár más is sikeresen tudja módosítani a kliens működését, illetve azt további modulokkal lehessen bővíteni. Az alkalmazás készítése továbbá a következő elvárások figyelembe vételével készült:

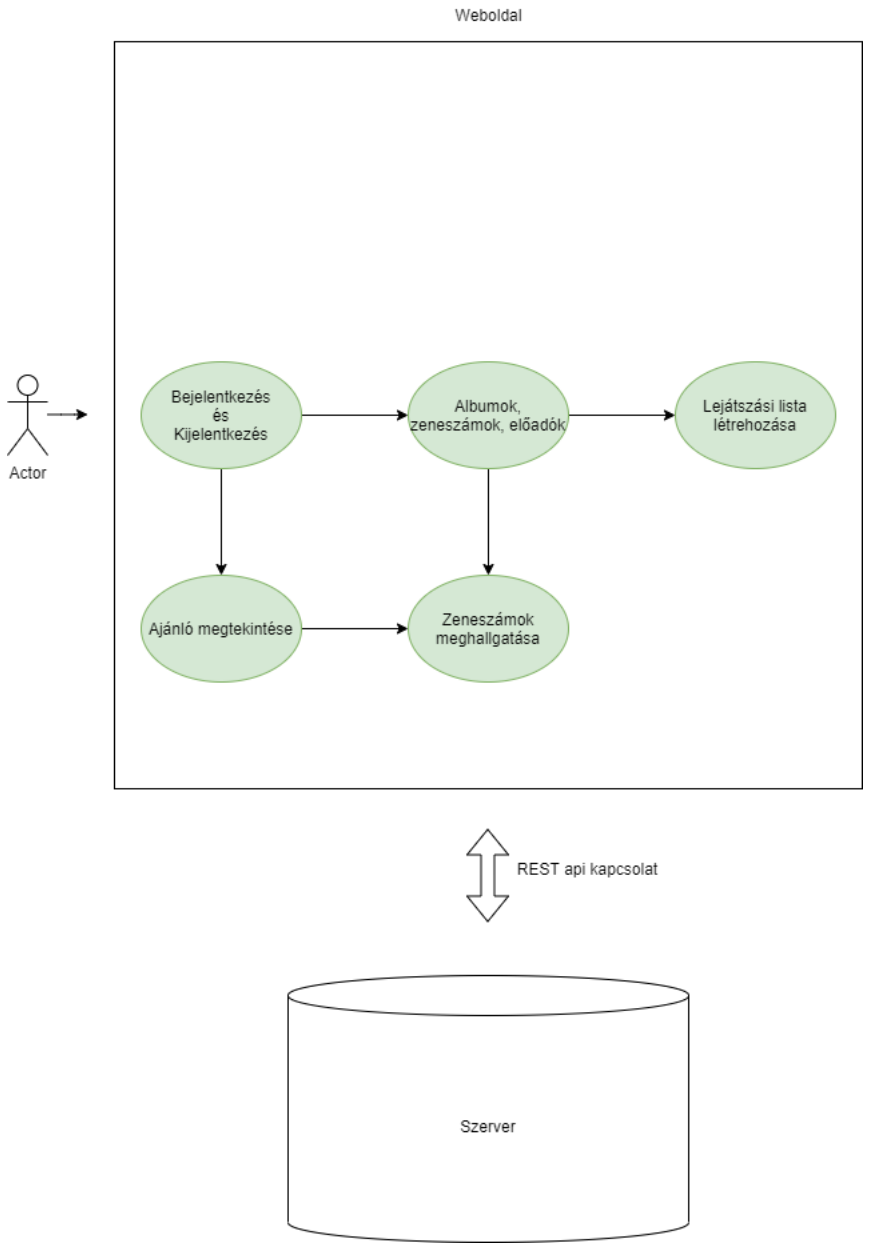
1. Minden felhasználói felületet érintő logika a kliensen fusson
2. Legyen képes a szerver oldali RESTful webszolgáltatással HTTP protokollon kommunikálni
3. Legalább Firefox és Google Chrome böngészők támogatása
4. Lehetőség felhasználó létrehozására
5. Lehetőség felhasználó authentikálására
6. Zeneszámok keresése
7. Zeneszámok meghallgatása
8. Lejátszási listák összeállítása
9. Lejátszási listák megtekintése
10. Ajánlások megtekintése
11. Zeneszámok, albumok feltöltése, törlése, változtatása előadók számára

### 3.3.2. Használt könyvtárak

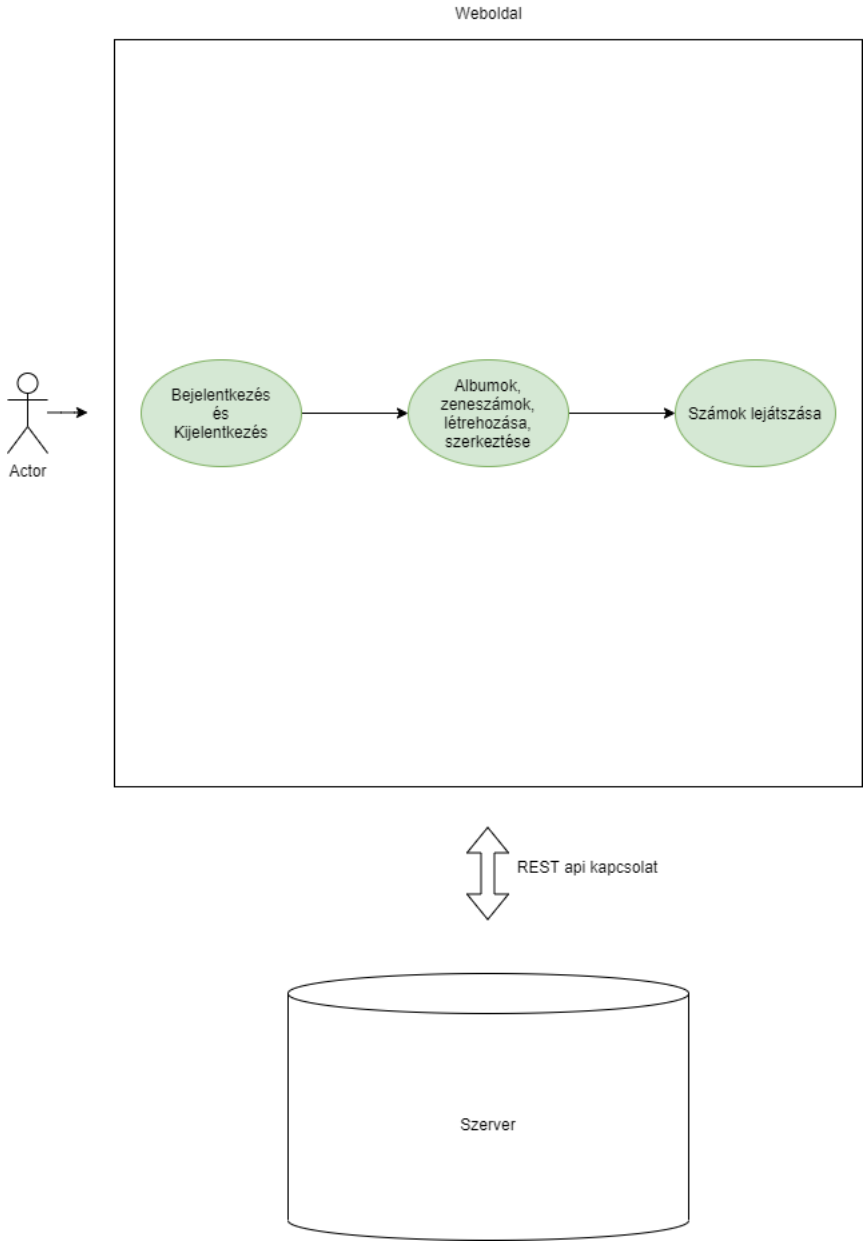
1. axios 0.19.2: HTTP Request library
2. bootstrap 4.4.1: Dizájn library
3. bootstrap-vue 2.7.0: Boostrap Vue bindingok
4. dotenv 8.2.0: .env fájl parsolához használt library
5. vue 2.5.2: Kliens oldali keretrendszer
6. vue-router 3.0.1: Kliens oldai routing library

### 3.3.3. Felépités

`Listener` felhasználói fiókkal


Weboldal
Actor
Bejelentkezés és Kijelentkezés
Albumok, zeneszámok, előadók
Lejátszási lista létrehozása
Ajánló megtekintése
Zeneszámok meghallgatása
REST api kapcsolat
Szerver

`Előadó fiókkal`



### 3.3.4. Kliens oldali adatok tárolása

Szükségünk van a nehány funkció megkönnyítéséhez eltárolnunk adatokat a böngészőben is. Ehhez a minden böngészőben elérhető `localStorage`-t használtam. A nativ javascript apiból elérhető és kulcs-érték párokként tudjuk eltárolni. Hasonló a `cookie`-khoz, de az ebben tárolt értékek nem kerülnek bele a request headerbe. Kapacitása böngészőtől függően 5Mbtől 10Mbig terjed. Élettartama addig tart, amíg be nem zárjuk a böngészőt.

Példa a használatra:

```
localStorage.setItem('key', 'value'); // Set a { key: value } pair
    into the localStorage

const val = localStorage.getItem('key') // Get the value by key,
    returns 'value'
```

3.17. forráskód. localStorage usage

A kliens oldali alkalmazásban a következő adatok kerülnek be a localStorage-be:

1. `username` : A felhasználó neve, azért hogy a navbarban folyamatosan megjelenithető legyen.
2. `userId` : A felhasználó egy egyedi azonosítója, amely szükséges bizonyos requestek sikerességéhez.
3. `token` : A szerver végpointjainak eléréséhez szükséges JWT alapú authentikációs token
4. `userType` : A felhasználó tipusa, megkönnyiti a DOM elemek kirajzolását

### 3.3.5. Felhasználó authentikálásának folyamata

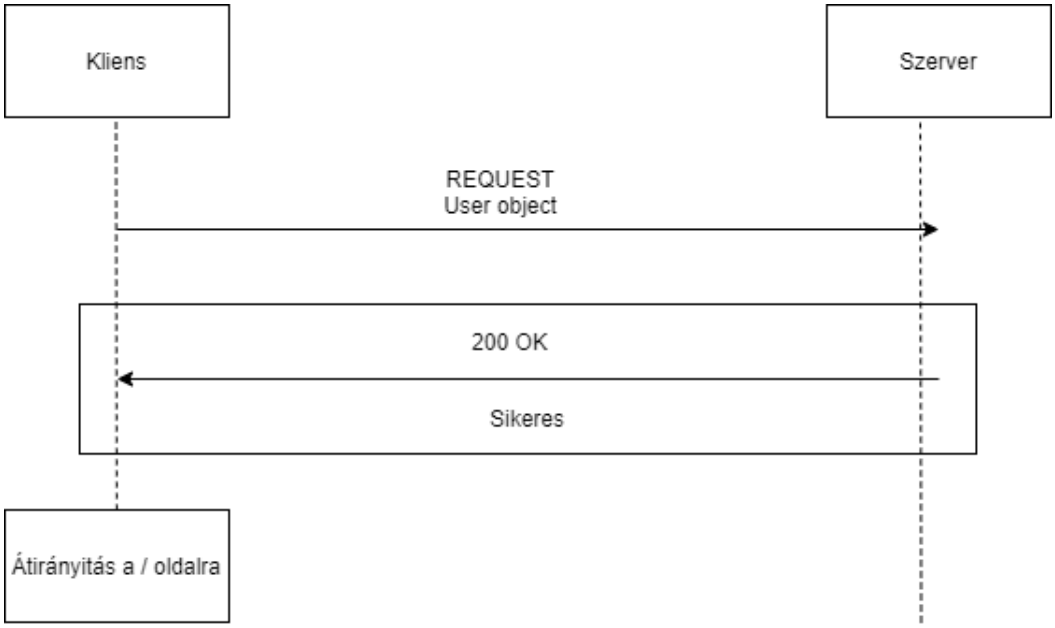
A felhasználók a `/registration` oldalon keresztül tudják magukat beregisztrálni. A művelet sikerességéről toast notificationként kapunk értesítést a kliens oldalon. A regisztrációs requestet az axios library segitségével HTTP POST metódussal küldjük el a szervernek, JSON formátumban. Az axios egy promise-based megoldásával a then részbe érünk ha sikeresen végbement a request és kaptunk választ a szervertől, a catch ágba kerülünk hogy ha valami probléma történt, például nem értük el a szervert. Ha sikeresen regisztráltunk, utána a főoldalra redirecteljük a felhasználót és kiküljük az fiók megerősitő emailt. Miután megerősitettük fiókunkat,

utána betudunk lépni.

```
    axios
        .post(config.userEndpointUrl, this.form, {
          headers: {
            'Content-Type': 'application/json'
          }
        })
        .then(response => {
          console.log(response)
          if (response.data.registration === 'success') {
            this.$bvToast.toast('You will be redirected! Check your
                email to verify your registration', {
              title: 'Registration successfull!',
              toaster: 'b-toaster-top-full',
              variant: 'success',
              solid: true
            })
            setTimeout(() => {
              console.log('timeout')
              router.push('/')
            }, 3000)
          } else {
            this.$bvToast.toast('Username or email is exists in our
                system', {
              title: 'Registration failed!',
              toaster: 'b-toaster-top-full',
              variant: 'danger',
              solid: true
            })
          }

          this.form.email = ''
          this.form.username = ''
          this.form.password = ''
        })
        .catch(err => {
          this.$bvToast.toast(err, {
```

```
            title: 'ERROR',
            toaster: 'b-toaster-top-full',
            variant: 'danger',
            solid: true
          })
        })
```

3.18. forráskód. app.js



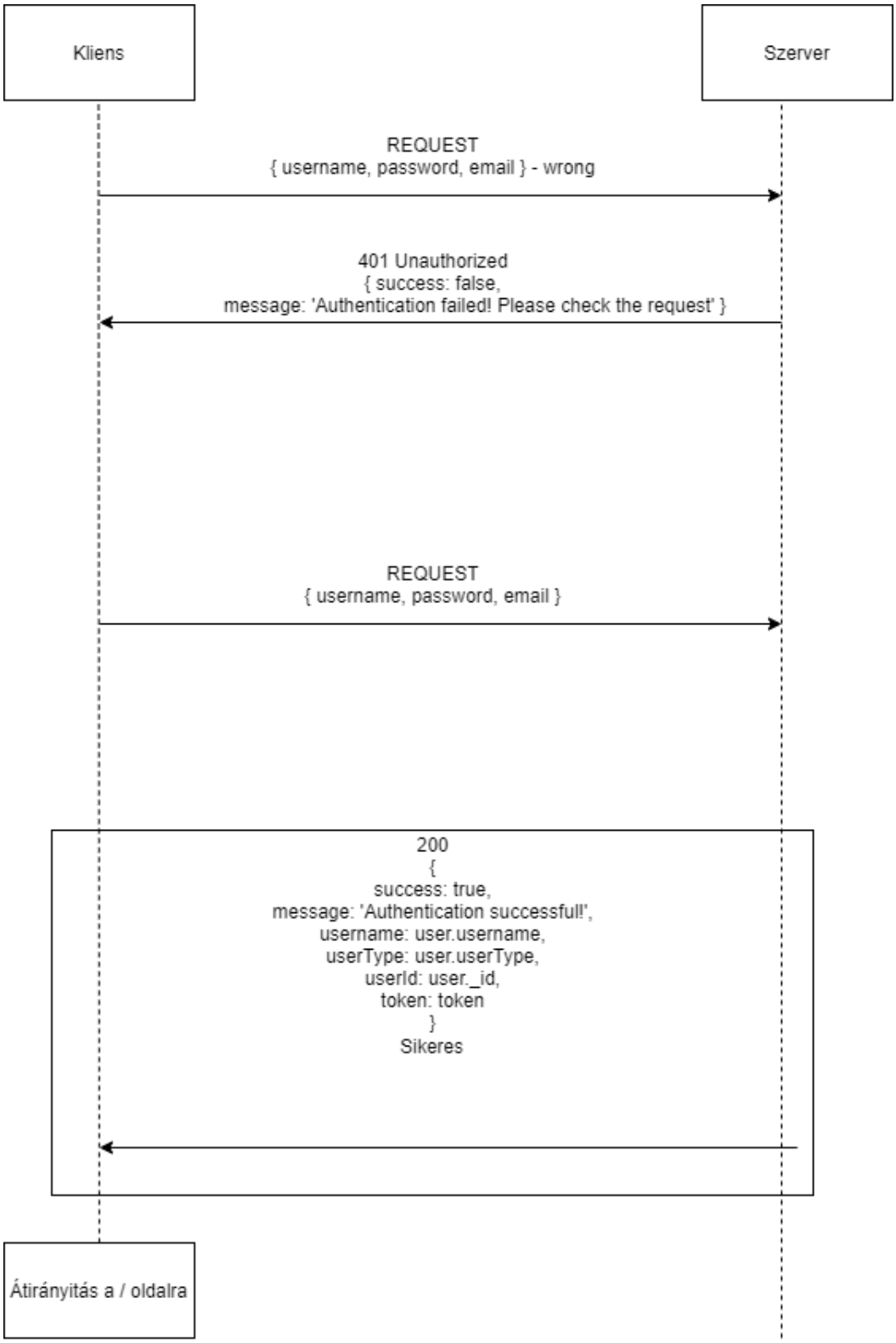

A felhasználók a `/login` oldalon keresztül tudják magukat authentikálni, a megfelelő felhasználónévvel, jelszóval és emaillel. A művelet sikeressegéről toast notificationt kapunk a kliens oldalon. Ha nem sikerült bejelentkezni, akkor 401 Unauthorized hibát kapunk és a kliens figyelmeztet, hogy ellenőrizzük le a belépési adatainkat. A bejentkezési requestet axios library segitségével egy HTTP POST metódussal küldjük el a szervernek, json formátumban. Az axios promise-based megoldásával tudjuk, hogy ha a catch részbe értünk akkor vagy nem érhető el a szerver, vagy nem tudtunk bejelentkezni. Sikeres bejelentkezés esetén elmentjük a fentebb irt adatokat a localStoragebe, majd a főoldalra irányitjuk a felhasználót.

```
    axios.post(config.userAuthURL, this.form, {
        headers: {
          'Content-Type': 'application/json'
        }
      })
```

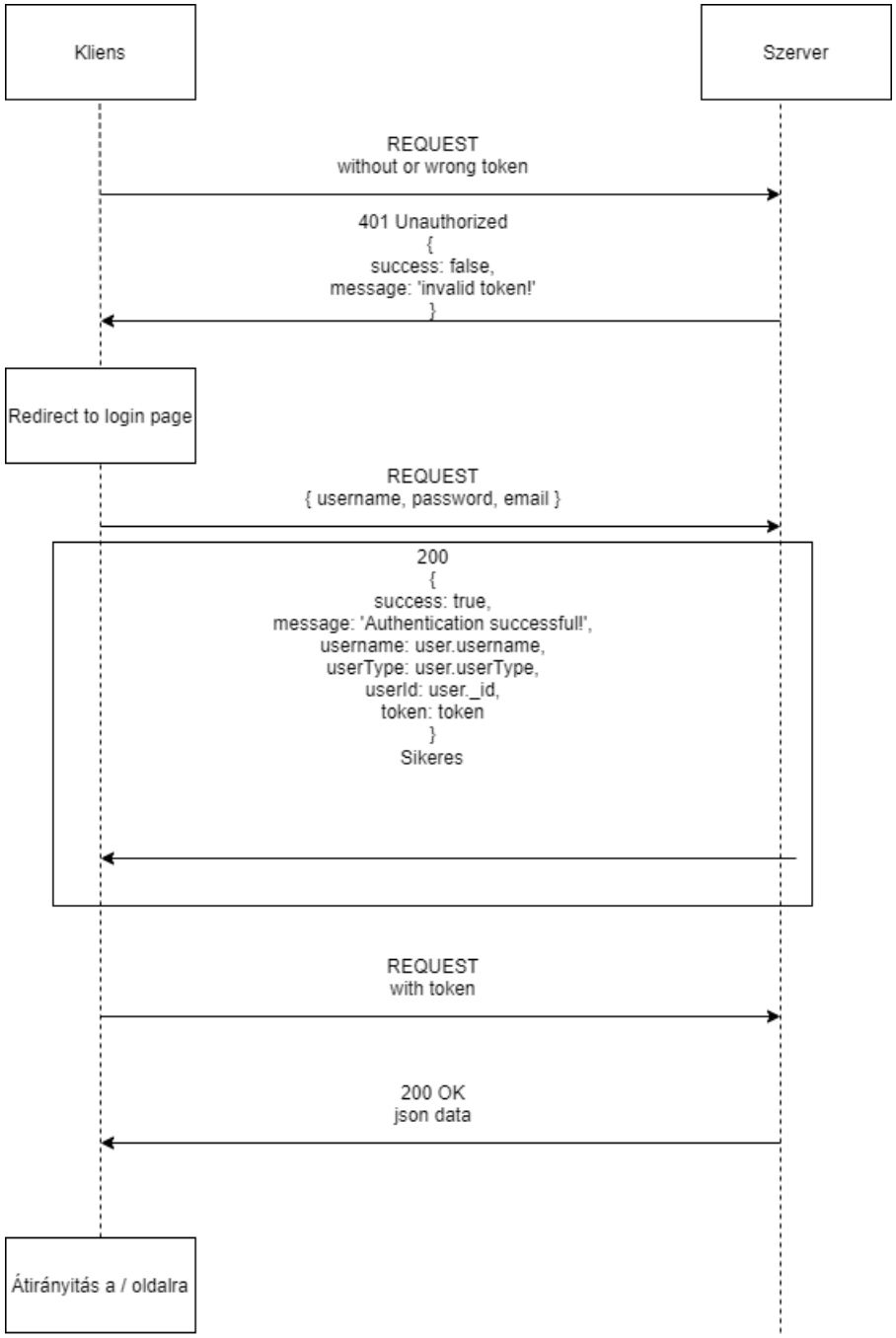
```
6         .then(response => {
7           console.log(response)
8           if (response.data.success === true) {
9             localStorage.setItem('token', response.data.token)
10            localStorage.setItem('username', response.data.username
                )
11            localStorage.setItem('userType', response.data.userType
                )
12            localStorage.setItem('userId', response.data.userId)
13
14            this.$bvToast.toast('Logged in!', {
15              title: 'Logged in successfully',
16              toaster: 'b-toaster-top-full',
17              variant: 'success',
18              solid: true
19            })
20
21            setTimeout(() => {
22              router.push('/')
23              location.reload()
24            }, 1500)
25          }
26        }).catch(err => {
27          if (err.response.status === 401) {
28            this.$bvToast.toast('Check your login data', {
29              title: 'ERROR',
30              toaster: 'b-toaster-top-full',
31              variant: 'danger',
32              solid: true
33            })
34          } else {
35            this.$bvToast.toast(err.message, {
36              title: 'ERROR',
37              toaster: 'b-toaster-top-full',
38              variant: 'danger',
39              solid: true
40            })
41          }
```

```
42            })
```

3.19. forráskód. app.js



Az általános requestek a kliensről axioson keresztül mennek. Az axios request headerjébe beleillesztjük a tokenként a localStorageben tarolt Bearer tokent, igy érjük el az endpointokat.



### 3.3.6. Kliens oldali routing

A szerver oldalhoz hasonlóan itt is meg kell mondanunk, hogy milyen urlen milyen komponens érhető el. Ehhez a `vue-router` modult használjuk. Ez lehetővé teszi hogy meghatározzuk melyik url-re melyik komponens renderelődjön.

Hogy megelőzzük azt hogy a felhasználó olyan oldalt érjen el ahol szükség lenne authentikációra, a router `beforeEach` metódusát használom. Ez minden egyes routing előtt lefut, és ahol a route rendelkezik a `requireAuth` metával azoknál ellenőrzi, hogy van-e tokenje a bejelentkezett felhasználónak. Ha nincsen akkor átirányitja a felhasználó a bejelentkezési oldalra.

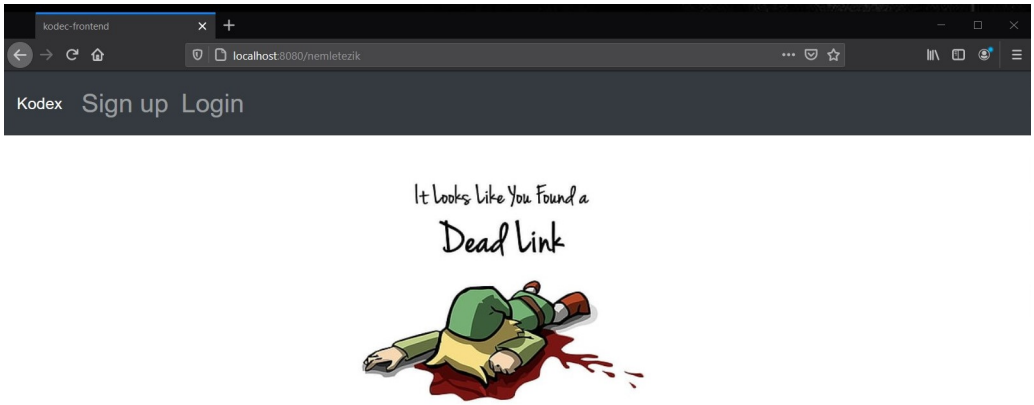
```
router.beforeEach((to, from, next) => {
  if (to.matched.some(record => record.meta.requiresAuth)) {
    // this route requires auth, check if logged in
    // if not, redirect to login page.
    if (!localStorage.getItem('token')) {
      next({ name: 'Login' })
    } else {
      next() // go to wherever I'm going
    }
  } else {
    next() // does not require auth, make sure to always call next
        ()!
  }
})
```

3.20. forráskód. src/main.js

Ha a felhasználó olyan URL-t akar elérni, ami nem létezik a routing moduleunkban, akkor átirányítja arra az oldalra ami felhivja a figyelmét erre a problémára. A routing moduleban ezt a következőféleképpen tehetjük meg.

```
{
  path: '*',
  name: 'WrongLink',
  component: WrongLink
}
```

3.21. forráskód. src/main.js




Példa routing modulra:

```
import Vue from 'vue'
import Router from 'vue-router'
import LandingPage from '@/components/LandingPage'
import Registration from '@/components/Registration'
import WrongLink from '@/components/WrongLink'
import Login from '@/components/Login'
import Albums from '@/components/AlbumsPage'
import AlbumEdit from '@/components/AlbumEdit'


Vue.use(Router)

export default new Router({
  mode: 'history',
  base: process.env.BASE_URL,
  routes: [
    {
      path: '/',
      name: 'LandingPage',
      component: LandingPage
    },
    {
      path: '/registration',
      name: 'Registration',
      component: Registration
```

```
26      },
27      {
28        path: '/login',
29        name: 'Login',
30        component: Login
31      },
32      {
33        path: '/albums',
34        name: 'Albums',
35        component: Albums,
36        meta: {
37          requiresAuth: true
38        }
39      },
40      {
41        path: '/album/:id',
42        name: 'AlbumEdit',
43        component: AlbumEdit,
44        meta: {
45          requiresAuth: true
46        }
47      },
48      {
49        path: '*',
50        name: 'WrongLink',
51        component: WrongLink
52      }
53    ]
54  })
```

3.22. forráskód. Routing modul példa

A kliens oldali routing táblázat a különbözőképpen néz ki:

| URL | Név | Komponens | Meta |
|---|---|---|---|
| / | LandingPage | LandingPage | Nem |
| /registration | Registration | Registration | Nem |
| /login | Login | Login | Nem |
| /albums | Albums | Albums | Igen |
| /album/:id | AlbumEdit | AlbumEdit | Igen |
| /list/album | AlbumList | AlbumList | Igen |
| /view/artist/:id | AlbumListByArtist | AlbumListByArtist | Igen |
| /view/album:id | SongListByAlbum | SongListByAlbum | Igen |
| /list/artist | ArtistList | ArtistList | Igen |
| /track/:id | TrackEdit | TrackEdit | Igen |
| /list/songs | ListAllSongs | ListAllSong | Igen |
| /list/playlist | Playlist | PlayList | Igen |
| /play/:id | Player | PlayerComponent | Igen |
| /user/auth/userverify/:id | UserVerify | UserVerify | Nem |
| /discover | Discover | Discover | Igen |
| * | WrongLink | WrongLink | Nem |

## 3.4. Tesztelés

### 3.4.1. Szerver oldali unit teszt

A szerver oldalon a végpontokat unit tesztekkel tudjuk tesztelni. Ezek az úgynevezett API tesztek. Ezt a mocha és a supertest könyvtárakkal tudjuk a legegyszerübben megvalósítani. Itt teszteljük a sikeres felhasználó lérehozást, lekérést, változtatást, törlést. Ugyanezt elvégezzük az albumra is. Valamint leteszteljük az authentikációs modult is. Ezeknél mock felhasználókat használunk, akik csak addig szerepelnek az adatbázisban amig a tesztek futnak. Példa egy ilyen test suitra:

```
'use strict';

const { ObjectId } = require('bson');
const request = require('supertest');
var should = require('chai').should();

const jwt = require('../middleware/jwt');
const app = require('../app');

describe('API TEST -- POST /album', () => {
    it('Create a album object, response a success message', (done)
        => {
        request(app)
            .post('/album')
            .set('Authorization', `Bearer ${jwt.generateToken('
                testuser')}`)
            .send({
                _id: new ObjectId('5fd68fd21d72b23500cfb182'),
                artist: 'TesztArtiszt',
                name: 'Elso EP',
                releaseYear: 2020
            })
            .set('Accept', 'application/json')
            .expect('Content-Type', /json/)
            .expect(200, {
                success: true,
                message: 'Elso EP created successfully!'
            }, done);
```

```
27     });
28 });
29
30 describe('API TEST -- GET /album', () => {
31     it('Get an album object', (done) => {
32         request(app)
33             .get('/album/5fd68fd21d72b23500cfb182')
34             .set('Authorization', `Bearer ${jwt.generateToken('
                   testuser')}`)
35             .set('Accept', 'application/json')
36             .expect('Content-Type', /json/)
37             .expect(200, {
38                 __v: 0,
39                 _id: '5fd68fd21d72b23500cfb182',
40                 artist: 'TesztArtiszt',
41                 name: 'Elso EP',
42                 releaseYear: 2020
43             }, done);
44     });
45 });
46
47 describe('API TEST -- PUT /album', () => {
48     it('Update an album object, response the updated object', (done
           ) => {
49         request(app)
50             .put('/album/5fd68fd21d72b23500cfb182')
51             .set('Authorization', `Bearer ${jwt.generateToken('
                   testuser')}`)
52             .send({
53                 _id: new ObjectId('5fd68fd21d72b23500cfb182'),
54                 artist: 'TesztArtiszt',
55                 name: 'Elso Nagylemez',
56                 releaseYear: 2021
57             })
58             .set('Accept', 'application/json')
59             .expect('Content-Type', /json/)
60             .expect(200, {
61                 succes: true,
62                 message: 'Album updated!'
```

```
63                }, done);
64        });
65    });
66
67    describe('API TEST -- DELETE /album', () => {
68        it('Get an result message', (done) => {
69            request(app)
70                .delete('/album/5fd68fd21d72b23500cfb182')
71                .set('Authorization', `Bearer ${jwt.generateToken('
                      testuser')}`)
72                .set('Accept', 'application/json')
73                .expect('Content-Type', /json/)
74                .expect(200, {
75                    message: 'Album deleted'
76                }, done);
77        });
78    });
```

3.23. forráskód. test/album.api.test.js

A unit tesztek eredménye:

```
  API TEST -- POST /album
Connected to the database at mongodb://localhost:27017/thesis
POST /album 200 30.083 ms - 58
    ✓ Create a album object, response a success message (96ms)

  API TEST -- GET /album
GET /album/5fd68fd21d72b23500cfb182 200 10.944 ms - 102
    ✓ Get an album object

  API TEST -- PUT /album
PUT /album/5fd68fd21d72b23500cfb182 200 13.855 ms - 42
    ✓ Update an album object, response the updated object

  API TEST -- DELETE /album
DELETE /album/5fd68fd21d72b23500cfb182 200 10.914 ms - 27
    ✓ Get an album object

  API TEST -- Auth test
POST /user 200 22.279 ms - 26
    ✓ Create a listener user object, response with an object
POST /auth 200 7.410 ms - 300
    ✓ Login with the user
DELETE /user/5fad67f50922fc684a3ec5c3 200 4.434 ms - 26
    ✓ Remove an user object

  API TEST -- Auth fail - not verified
POST /user 200 3.876 ms - 26
    ✓ Create a listener user object, response with an object
POST /auth 401 5.766 ms - 77
    ✓ Login with the user
DELETE /user/5fad67f50922fc684a3ec5c3 200 3.639 ms - 26
    ✓ Remove an user object

  API TEST -- Auth fail - wrong password
POST /user 200 3.895 ms - 26
    ✓ Create a listener user object, response with an object
POST /auth 401 5.884 ms - 77
    ✓ Login with the user
DELETE /user/5fad67f50922fc684a3ec5c3 200 3.184 ms - 26
    ✓ Remove an user object

  API TEST -- GET /
GET / 200 0.442 ms - 15
    ✓ Response with an { status: Ok } message

  API TEST -- POST /user
POST /user 200 3.371 ms - 26
    ✓ Create a listener user object, response with an object

  API TEST -- PUT /user
PUT /user/5fad67f50922fc684a3ec5c3 200 9.303 ms - 20
    ✓ Update an user object

  API TEST -- GET /user
GET /user/5fad67f50922fc684a3ec5c3 200 3.699 ms - 260
    ✓ Get an user object

  API TEST -- DELETE /user
DELETE /user/5fad67f50922fc684a3ec5c3 200 2.915 ms - 26
    ✓ Remove an user object

  API TEST -- POST /user
POST /user 200 4.188 ms - 26
    ✓ Create a listener user object, response with an object

  API TEST -- PUT /user
PUT /user/5fd68eb30160f9eb9aa034f5 200 6.661 ms - 20
    ✓ Update an user object

  API TEST -- GET /user
GET /user/5fd68eb30160f9eb9aa034f5 200 4.013 ms - 255
    ✓ Get an user object

  API TEST -- DELETE /user
DELETE /user/5fd68eb30160f9eb9aa034f5 200 3.265 ms - 26
    ✓ Remove an user object


  22 passing (360ms)
```

### 3.4.2. Szerver oldali integrációs teszt

Egy integrációs tesztben már komplexebb feladatokat kell letesztelni mint egy egyszerü api tesztben. Vegyünk példának egy Album elkészítését egy előadói felhasználóval. Ekkor a következő lépéseket kell végig csinálnunk egy tesztben:

1. Készítünk egy előadói felhasználót
2. Authentikáljuk magunkat
3. Létrehozzuk az albumot
4. Lekérjük az albumot hogy leellezőrizzük sikeresen létre jött-e
5. Töröljük az albumot
6. Töröljük a felhasználót

Ugyanez kódban:

```
'use strict';

const { ObjectId } = require('bson');
const request = require('supertest');
var should = require('chai').should();


const jwt = require('../middleware/jwt');
const app = require('../app');

describe('INTEGRATION TEST -- Create album with an artist user', ()
    => {
    it('Create a listener user object, response with an object', (
        done) => {
         request(app)
             .post('/user')
             .send({
                 _id: new ObjectId('5fad67f50922fc684a3ec5c3'),
                 username: 'testuser',
                 password: 'testpassword',
                 email: 'test@test.com',
                 userType: 'artist',
                 genre: 'Metalcore',
                 isVerified: true,
                 createdAt: '2020-12-13T20:28:36.476Z'
```

```
24            })
25            .set('Accept', 'application/json')
26            .expect('Content-Type', /json/)
27            .expect(200, {
28                registration: 'success'
29            }, done);
30    });
31
32    it('Login with the user', (done) => {
33        request(app)
34            .post('/auth')
35            .send({
36                _id: new ObjectId('5fad67f50922fc684a3ec5c3'),
37                username: 'testuser',
38                password: 'testpassword',
39                email: 'test@test.com',
40                userType: 'artist',
41                genre: 'Metalcore',
42                isVerified: true,
43                createdAt: '2020-12-13T20:28:36.476Z'
44            })
45            .set('Accept', 'application/json')
46            .expect('Content-Type', /json/)
47            .expect(200)
48            .end((err, res) => {
49                if (err) { return done(err); }
50                res.body.userId.should.equal('5
                     fad67f50922fc684a3ec5c3');
51                res.body.message.should.equal('Authentication
                     successful!');
52                res.body.username.should.equal('testuser');
53                res.body.token.should.match(/.+/);
54                res.body.userType.should.equal('artist');
55                res.body.success.should.equal(true);
56                done();
57            });
58    });
59
60    it('Create a album object, response a success message', (done)
```

```
        => {
61          request(app)
62              .post('/album')
63              .set('Authorization', `Bearer ${jwt.generateToken('
                    testuser')}`)
64              .send({
65                  _id: new ObjectId('5fd68fd21d72b23500cfb182'),
66                  artist: 'testuser',
67                  name: 'Elso EP',
68                  releaseYear: 2020
69              })
70              .set('Accept', 'application/json')
71              .expect('Content-Type', /json/)
72              .expect(200, {
73                  success: true,
74                  message: 'Elso EP created successfully!'
75              }, done);
76      });
77
78      it('Get an album object', (done) => {
79          request(app)
80              .get('/album/5fd68fd21d72b23500cfb182')
81              .set('Authorization', `Bearer ${jwt.generateToken('
                    testuser')}`)
82              .set('Accept', 'application/json')
83              .expect('Content-Type', /json/)
84              .expect(200, {
85                  __v: 0,
86                  _id: '5fd68fd21d72b23500cfb182',
87                  artist: 'testuser',
88                  name: 'Elso EP',
89                  releaseYear: 2020
90              }, done);
91      });
92
93      it('Get an result message', (done) => {
94          request(app)
95              .delete('/album/5fd68fd21d72b23500cfb182')
96              .set('Authorization', `Bearer ${jwt.generateToken('
```

```
                testuser')}`)
            .set('Accept', 'application/json')
            .expect('Content-Type', /json/)
            .expect(200, {
                message: 'Album deleted'
            }, done);
    });

    it('Remove an user object', (done) => {
        request(app)
            .delete('/user/5fad67f50922fc684a3ec5c3')
            .set('Authorization', `Bearer ${jwt.generateToken('
                testuser')}`)
            .set('Accept', 'application/json')
            .expect('Content-Type', /json/)
            .expect(200, {
                message: 'User deleted'
            }, done);
    });
});
```

3.24. forráskód. test/album.api.test.js

A futási eredménye:

```
  INTEGRATION TEST -- Create album with an artist user
POST /user 200 3.343 ms - 26
    √ Create a listener user object, response with an object
POST /auth 200 7.110 ms - 300
    √ Login with the user
POST /album 200 5.017 ms - 58
    √ Create a album object, response a success message
GET /album/5fd68fd21d72b23500cfb182 200 3.486 ms - 98
    √ Get an album object
DELETE /album/5fd68fd21d72b23500cfb182 200 9.456 ms - 27
    √ Get an result message
DELETE /user/5fad67f50922fc684a3ec5c3 200 2.732 ms - 26
    √ Remove an user object
```

### 3.4.3. Kliens oldali integrációs teszt

A kliens oldali integrációs tesztelést seleniummal oldottam meg. Itt is megcsináljuk az előző tesztet, csak itt belevesszük a kliens oldalt is. A teszt lefuttatásához szükséges, hogy legyen 83.0-ás Firefox és 87.0.4280.88 verzió számu Google Chrome böngészőre. A Google Chromehoz továbbá szükséges program még a chromedriver, a Firefox böngészőhöz pedig a geckodriver. A megjelölt verziókhoz szükséges geckodriver és chromedriver mellékelve van.

Az integration-test mappában található a RunTests.py fájl ami tartalmazza a teszteket. A tesztek pythonban iródtak, selenium és unittest könyvtárak használják. A tesztek párhuzamosan futnak, a Firefox headless módban (ilyenkor nem nyit böngésző ablakot), viszont a chromedriver mostani verziójában egy bug miatt nem tudja headless módban megtalálni xpath alapján az elemeket, igy az nem headless módban fut, szóval böngésző ablakokat nyit. Teszt eredménye:

```
PS D:\Projects\university_thesis\integration-test> python .\RunTests.py

DevTools listening on ws://127.0.0.1:57996/devtools/browser/2f040ae0-8372-4d18-a929-18c8c9b4f055
.
DevTools listening on ws://127.0.0.1:58044/devtools/browser/32a544ef-8235-45be-acbd-08140bac626d
.
DevTools listening on ws://127.0.0.1:58095/devtools/browser/10568234-190a-4780-baa2-ce5205b6e40b
.
DevTools listening on ws://127.0.0.1:58261/devtools/browser/e6c2f9b4-ac5a-4823-a1c2-f46ea58349a0
.
DevTools listening on ws://127.0.0.1:58355/devtools/browser/3178ed5e-26d6-485f-9ef2-97554dc2a8fe
.
DevTools listening on ws://127.0.0.1:58391/devtools/browser/13fcd9b9-d3d2-401b-8ebf-ddc16bf497b3
.
DevTools listening on ws://127.0.0.1:58440/devtools/browser/70259f9a-2f92-4c4c-a7da-236a8d90981c
.
DevTools listening on ws://127.0.0.1:58521/devtools/browser/b5288bab-be27-4329-b5b2-7dc45e2aefe4
..........
----------------------------------------------------------------------
Ran 18 tests in 168.571s

OK
```

## 3.5. További fejlesztési lehetőségek

1. Admin felület bevezetése
2. ACL Roleok bevezetése
3. Egyedi lejátszó létrehozása
4. Folyamatos lejátszás megvalósitása Indexed DB segitségével
5. Mobilra portolás

# 4. fejezet

# Összegzés

A szakdolgozatom célja az volt hogy egy kezdetleges platformot hozzak létre ahol előadók tudnak feltölteni számaikat, és a felhasználók tudják azt hallgatni és kapnak új ajánlásokat. Az alkalmazás teljesiti az előirt szükséges feltételeket viszont a valódi céljától még messze van. Jelenleg hiányzik egy nagyon ember barát UI a programból és a legfontosabb dolog. Ami pedig az hogy sokan használják a platformot. Minél több ember tölt fel rá tartalmat, minél jobban irják körül a feltöltött számokat, annál jobb lesz az ajánlás.

Valamint felkell készíteni az alkalmazást a mobilos platformokra. Az emberek jelentős része (magamat is beleértve) csak telefonon használ akármilyen zene streaming szolgáltatast.

Viszont mindezek mellett sem garantált hogy sikeres lenne az alkalmazás. Hiszen hiába hogy a feltételeket teljesiti, ha nincs elég tartalom vagy éppen megfelelő tartalom ami megtartja a felhasználókat akkor az alkalmazás sosem fogja kinőni magát.

# 5. fejezet

# Köszönetnyilvánítás

Ezúton köszönöm Pataki Norbertnek a témavezetést, hogy hasznos ötleteivel és tanácsaival támogatott. Külön köszönöm a dolgozat lektorálásában nyújtott fáradhatatlan segítségét.

# Irodalomjegyzék

[1] *Vue.js*. (2020. december 14.) URL: https://vuejs.org/.

[2] *Docker*. (2020. december 14.) URL: https://www.docker.com/.

[3] Wikipedia. *tf-idf*. (2020. december 14.) URL: https://en.wikipedia.org/wiki/Tf%E2%80%93idf.

# Forráskódjegyzék


3.1. Users schema . . . . . 24
3.2. Albums schema . . . . . 25
3.3. Tracks schema . . . . . 27
3.4. Playlists schema . . . . . 28
3.5. User create controller . . . . . 31
3.6. Album update controller . . . . . 32
3.7. AWS config . . . . . 33
3.8. JWT middleware . . . . . 34
3.9. Tracks schema . . . . . 35
3.10. getData.js . . . . . 37
3.11. getRecommendations függvény . . . . . 38
3.12. getRecommendationsByPlaylist függvény . . . . . 39
3.13. routes/index.js . . . . . 41
3.14. routes/user.js . . . . . 42
3.15. app.js . . . . . 44
3.16. app.js . . . . . 45
3.17. localStorage usage . . . . . 49
3.18. app.js . . . . . 50
3.19. app.js . . . . . 51
3.20. src/main.js . . . . . 55
3.21. src/main.js . . . . . 55
3.22. Routing modul példa . . . . . 56
3.23. test/album.api.test.js . . . . . 59
3.24. test/album.api.test.js . . . . . 63