



Eötvös Loránd Tudományegyetem

Informatikai Kar

Információs rendszerek Tanszék

Zenei albumokkal foglalkozó webalkalmazás személyre szabott ajánlórendszerrel

Szerző:

Marczis Bálint (wep5gk)

Programtervező informatikus BSc.

Témavezető:

Brányi László

Egyetemi oktató

Budapest 2025

EÖTVÖS LORÁND TUDOMÁNYEGYETEM
INFORMATIKAI KAR

SZAKDOLGOZAT TÉMABEJELENTŐ

Hallgató adatai:

Név: Marczis Bálint

Neptun kód: WEP5GK

Képzési adatak:

Szak: programtervező informatikus, alapképzés (BA/BSc/BProf)

Tagozat : Nappali

Belső témavezetővel rendelkezem

Témavezető neve: Brányi László

munkahelyének neve, tanszék: ELTE IK, Információs rendszerek Tanszék

munkahelyének címe: 1117, Budapest, Pázmány Péter sétány 1/C.

beosztás és iskolai végzettsége: Egyetemi oktató

A szakdolgozat címe: Zenei albumokkal foglalkozó webalkalmazás személyre szabott ajánlórendszerrel

A szakdolgozat témája:

(A témavezetővel konzultálva adja meg 1/2 - 1 oldal terjedelemben szakdolgozat témajának leírását)

Szakdolgozatom célja egy olyan zenei albumokkal foglalkozó modern webes applikáció elkészítése laravellel, amely lehetővé teszi a felhasználók számára, hogy szabadon böngészhetzenek, értékelhessenek és megismerjenek, a saját ízlésükhez passzoló új és régi albumokat, ezeken felül pedig hogy naplózhassák milyen albumokat hallgattak meg vagy akamak a későbbiekben.

Az oldal rendelkezni fog egy Python kódban íródott, machine learning modellel, ami a személyre szabott ajánlásokat fogja biztosítani a teljes körű felhasználói élmény elérése érdekében. A rendszer figyelembe fogja venni a felhasználók zenei preferenciáit és tevékenységeit az ajánlások kialakításához. Az ajánlórendszer folyamatosan alkalmazkodni fog az új értékelésekhez és tevékenységekhez.

Az applikáció különböző API-k használatával fogja folyamatosan frissíteni a zenei adatbázisát így minden naprakész információkkal fog rendelkezni az oldal. Bejelentkezés után (regisztráció és bejelentkezés lehetséges lesz facebook, apple, google vagy email fiókkal is) a főoldalon fognak majd megjelenni az újdonságok a személyre szabott ajánlások és az éppen népszerű albumok (a rendszer ki fogja emelni az éppen legtöbbet értékelt vagy legtöbb interakciót kiváltó albumokat).

A felhasználók képesek lesznek minden időszakban mindegyik 1-5-ig terjedő számskálán értékelni az albumokat. Továbbá kívánságlistára helyezni különöző zenéket, amelyeket az oldalon találtak és később hallgathának meg. Ezekben túl a közösségi élmény kialakítása érdekében a felhasználók bekövethetik egymást és ezáltal figyelemmel követhetik egymás tevékenységét is. Továbbá a felhasználók generálthatnak az oldallal különböző, általuk megadott szempontok alapján (pl. év, műfaj stb.) toplistákat. Később a program, a felsoroltak egy részét figyelembe véve fogja kialakítani az ajánlásokat a felhasználók számára.

Ezekben felül a program készíteni fog statisztikákat, évi és havi szinten a felhasználók számára, hogy láthatók hány és milyen műfajú albumokat értékeltek / jelölték be meghallgatottként, tettek fel kívánságlistára.

Budapest, 2023. 12. 11.

Tartalomjegyzék

1.	Bevezetés	1
2.	Felhasználói dokumentáció	3
2.1.	Gépigény	3
2.2.	Telepítési útmutató.....	4
2.2.1.	Ajánló rendszer telepítése és futtatása.....	4
2.2.2.	Laravel applikáció telepítése és futtatása	5
2.3.	Az alkalmazás bemutatása	6
2.3.1.	Site map – A webalkalmazás szerkezete	6
2.3.2.	Bejelentkezés és Regisztráció.....	8
2.3.3.	Fejléc funkciói	9
2.3.4.	Main Page (főoldal)	9
2.3.5.	Album oldal	11
2.3.6.	Előadó oldal.....	12
2.3.7.	Toplista oldal	14
2.3.8.	Saját toplista létrehozása	14
2.3.9.	Toplista generálás	17
2.3.10.	Profiloldal.....	17
3.	Fejlesztői dokumentáció	20
3.1.	Az ajánló rendszer.....	20
3.1.1.	Felhasznált technológiák – Python, FastAPI.....	20
3.1.2.	Collaborative Filtering, Cosine Similarity	21
3.2.	Az ajánló rendszer működése	22
3.2.1.	Hitelesítés	22
3.2.2.	Kommunikáció	23
3.2.3.	Ajánlások generálása.....	24

3.2.4.	Ajánlások visszaküldése	26
3.3.	A Webalkalmazás	26
3.4.	Felhasznált technológiák.....	27
3.4.1.	Keretrendszer – Laravel	27
3.4.2.	Frontend – Blade, Livewire	27
3.4.3.	Tailwind.....	28
3.4.4.	Adatbázis – PostgreSQL.....	28
3.5.	A Webalkalmazás Rétegei és Működése.....	28
3.6.	Adatbázis Modellek (Models)	29
3.6.1.	Az Alkalmazás Adatmodelljei	31
3.6.2.	Az adatmodellek részletezése és a fontosabb függvények:	33
3.6.3.	Wilson Score Interval – Népszerű Albumok	36
3.7.	Kontrollerek (Controllers) és Nézetek (Views)	39
3.7.1.	Egy Kontroller Működésének szemléltetése – AlbumController	40
3.8.	API Integrációk kontrollereken keresztül	43
3.8.1.	Spotify Web API	43
3.8.2.	Kommunikáció az Ajánlórendszerrel.....	45
3.9.	Livewire Komponensek	46
3.10.	Fontosabb Livewire Komponensek és funkcionálitások.....	48
3.10.1.	SearchDropDown	48
3.10.2.	EditList.....	50
3.10.3.	RatingComponent	51
3.11.	Autentikáció	52
3.12.	Tesztelés.....	52
3.12.1.	Ajánlórendszer Tesztelése.....	52
3.12.2.	Webalkalmazás Tesztelése.....	53

4.	Összefoglalás.....	61
5.	További fejlesztési lehetőségek	62
6.	Ábrajegyzék.....	63
7.	Irodalomjegyzék.....	65

1. Bevezetés

A szakdolgozatom tématerülete a modern webalkalmazás fejlesztés, az ehhez tartozó adatbáziskezelés és a felhasználói élmény javítását szolgáló személyre szabott ajánlórendszer építése FastAPI keretrendszer segítségével, Spotify API integrációval.

Szakdolgozatom célja egy felhasználó-központú webalkalmazás megvalósítása, ahol a zeneszerető felhasználók szabadon böngészhetnek zenei albumokat (az album információk begyűjtéséhez van szükség a Spotify API-ra), azokat értékelhetik szöveges vagy egy 1-5-ig terjedő skálán, csillagozással. Ezeken felül létrehozhatnak saját toplistákat vagy, akár generálhatnak műfajonként ranglistákat, ilyenkor az oldal különböző függvények segítségével kiszámolja az egyes albumokhoz tartozó népszerűségi pontszámot (erről a „fejlesztői dokumentáció” fejezetben lesznek további részletek) és azok alapján sorba rendezi őket. A felhasználók továbbá követhetik egymás tevékenységét és megtekinthetik saját statisztikáikat. Az oldal a felhasználók tevékenységét összegzi és ezek alapján ad személyre szabott album ajánlásokat (erről is bővebben a fejlesztői dokumentációban lesz szó).

A téma választásomat a zene iránti szenvedélyem, valamint a webalkalmazások fejlesztése és a Big Data technológiák iránti mély érdeklődésem inspirálta. A megvalósított programhoz hasonló alkalmazásokat magam is rendszeresen használok és mindig is érdekkelt milyen kihívásokat jelent egy ilyen webalkalmazás és az ehhez tartozó személyre szabott ajánló rendszer fejlesztése, különösen az utóbbi mivel a Big Data BSc-s tantárgyon az ajánló rendszerekről csak érintőlegesen volt szó. A szakdolgozat tökéletes alkalmat kínált arra, hogy mélyebben beleássam magam ebbe az izgalmas témaába és egy saját fejlesztésű ajánlórendszer megvalósításával gyakorlatban is elsajátítsam a szükséges ismereteket.

A megvalósítás során a Laravel full-stack PHP alapú webes keretrendszert használtam, amelyben az alkalmazás fő funkcionálitását hoztam létre. A webalkalmazás a Python nyelven írt ajánlórendszerrel egy FastAPI alapú REST API-n keresztül kommunikál, biztosítva a két rendszer közötti hatékony adatcserét.

A **felhasználói dokumentációban** bemutatom az alkalmazás használatát a felhasználók szemszögéből, illusztrációként képernyőfotókkal és ábrákkal kiegészítve. Emellett ismertetem a program telepítésének és futtatásának lépéseit. A **fejlesztői dokumentációban** részletesen

bemutatom a program hátterét, működését. Ezentúl a fejlesztés során meghozott döntések okaira is kitérek. A szakdolgozat zárásaként felvázolom a továbbfejlesztési lehetőségeket, irányokat.

2. Felhasználói dokumentáció

Az alkalmazás célközönségébe azon zeneszerető felhasználók tartoznak, akik szeretnének új és érdekes albumokat felfedezni. Az oldal lehetőséget kínál arra, hogy felhasználók személyre szabott ajánlásokat kapjanak, toplistákat böngésszenek, és mások zenei tevékenységeit követhessék. Emellett megoszthatják véleményüket bármely albumról, és napló formájában nyomon követhetik saját zenei felfedezéseiket és aktivitásukat az oldalon.

2.1. Gépigény

Minimális Gépigény

Windows:

- Operációs rendszer: Windows 8 64-bit vagy újabb
- Processzor: 2 magos, 1.8 GHz-es (Intel Core i3 vagy AMD megfelelője)

macOS:

- Operációs rendszer: macOS Mojave (10.14) vagy újabb
- Processzor: 2 magos Intel Core i3 vagy M1 (Apple Silicon)
- RAM: 4 GB
- Tárhely: Legalább 5 GB szabad hely

Optimális Gépigény

Windows:

- Operációs rendszer: Windows 10 64-bit vagy újabb
- Processzor: 4 magos, 2.5 GHz-es (Intel Core i5 vagy AMD megfelelője)

macOS:

- Operációs rendszer: macOS Monterey (12.0) vagy újabb
- Processzor: 4 magos Intel Core i5, M1/M2 (Apple Silicon) vagy újabb
- RAM: 8 GB

- Tárhely: Legalább 5 GB szabad hely

Egyéb

- Fejlesztői eszközök: Visual Studio Code vagy PyCharm és PHPStorm

2.2. Telepítési útmutató

Az program két fő részből áll, a Laravel webalkalmazásból (aminek a kódja a MusicApp mappában található) és az ajánlórendszerből (aminek a kódja az Api mappában található). Az alábbiakban részletezem a telepítés és futtatás menetét a két program esetén. Külön-külön és működnek, de a kettő együttes működésével válik teljessé az alkalmazás.

2.2.1. Ajánló rendszer telepítése és futtatása

Győződjön meg arról, hogy számítógépen a Python 3.12.7-es vagy újabb verziója elérhető. Ha nem rendelkezik semmilyen Python verzióval akkor a következőket kell tennie:

Windows:

Az alábbi oldalról telepítse a legújabb Python verziót:

<https://www.python.org/downloads/>.

macOS:

A Python telepítéséhez használja a Homebrew csomagkezelőt: `brew install python3`

Futtatás

Navigáljon el a terminálban abba a mappába, ahol az ajánló rendszer kódja található (Api mappa). Itt létre kell hoznia egy „virtuális környezetet” (virtual environment). Ezt érdemes VS Code-ban csinálni, az alábbi oldal részletesen leírja hogyan kell csinálni: <https://code.visualstudio.com/docs/python/python-tutorial>

Ezt követően a terminálban futtassa a következő parancsokat:

```
pip install -r requirements.txt (bizonyos környezetekben pip3 install -r requirements.txt)
```

```
pytest test_recommendations.py (a tesztek futtatásához)  
uvicorn main:app --reload --port 8001 (A fastAPI futtatásához)
```

A szervert a Ctrl+C billentyűzetkombinációval állíthatja le a terminálban.

2.2.2. Laravel applikáció telepítése és futtatása

Folyamatos internetkapcsolat szükséges az alkalmazás futtatásához!

Győződjön meg arról, hogy számítógépén a PHP 8.1-es vagy újabb verziója elérhető. Ha nem rendelkezik semmilyen PHP verzióval akkor a következő parancsok egyikével telepítheti a PHP-t a Composer csomagkezelőt és a Laravel-t:

Windows:

```
# Run as administrator...  
  
Set-ExecutionPolicy Bypass -Scope Process -Force;  
[System.Net.ServicePointManager]::SecurityProtocol =  
[System.Net.ServicePointManager]::SecurityProtocol -bor  
3072; iex ((New-Object System.Net.WebClient).DownloadString('https://php.new/install/windows'))
```

macOS:

```
/bin/bash -c "$(curl -fsSL https://php.new/install/mac)"
```

Ha már rendelkezik PHP-val és Composer-rel akkor a következő parancssal tudja telepíteni a Laravel-t: composer global require laravel/installer

Laravel konfigurálása és futtatása

Navigáljon a terminálban a Laravel kódot tartalmazó mappába (**MusicApp**) majd futassa a következő parancsokat az adott sorrendben:

```
composer install  
npm install  
php artisan key:generate
```

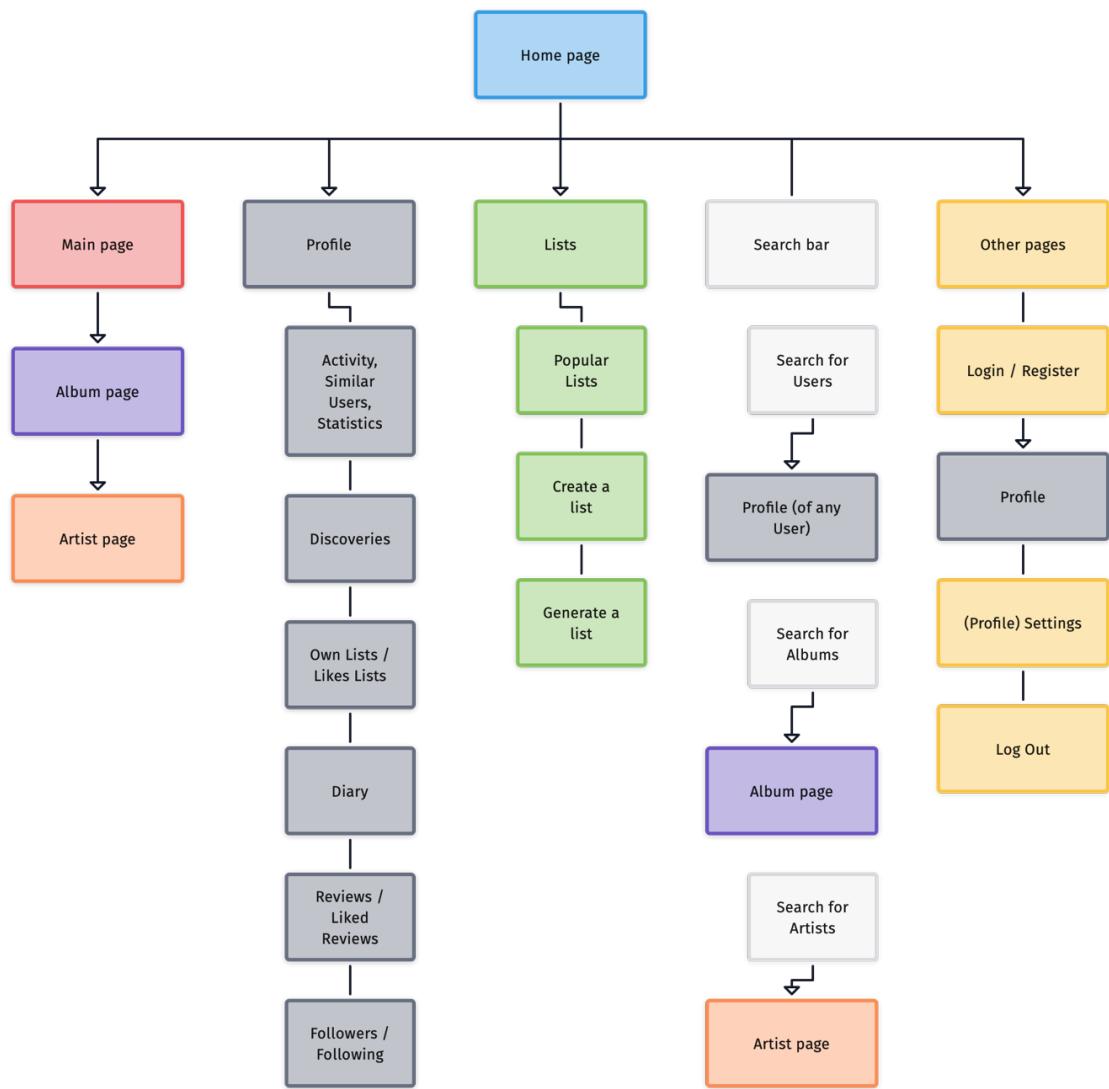
```
php artisan storage:link  
php artisan migrate  
  
php artisan test (ez a parancs lefuttatja a unit és feature teszteket, ez pár másodpercet vesz igénybe)  
  
php artisan migrate:fresh --seed  
  
npm run dev  
  
Ezután nyisson egy második terminált is, navigáljon a projekt mappájába (MusicApp) és futtassa a következő parancsot: php artisan serve
```

Ezt követően az alkalmazás elérhető a <http://127.0.0.1:8000> URL-en keresztül.

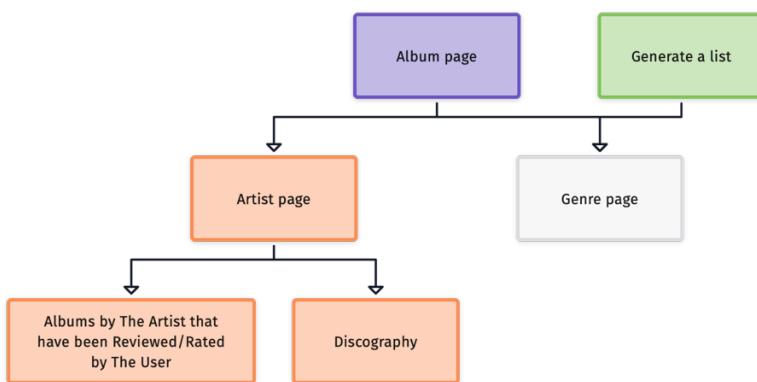
2.3. Az alkalmazás bemutatása

2.3.1. Site map – A webalkalmazás szerkezete

A Site map diagram (1. és 2. ábra) hivatott bemutatni az alkalmazás szerkezetét, milyen kapcsolatban állnak a különböző oldalak, hogyan lehet eljutni egyikről a másikra.

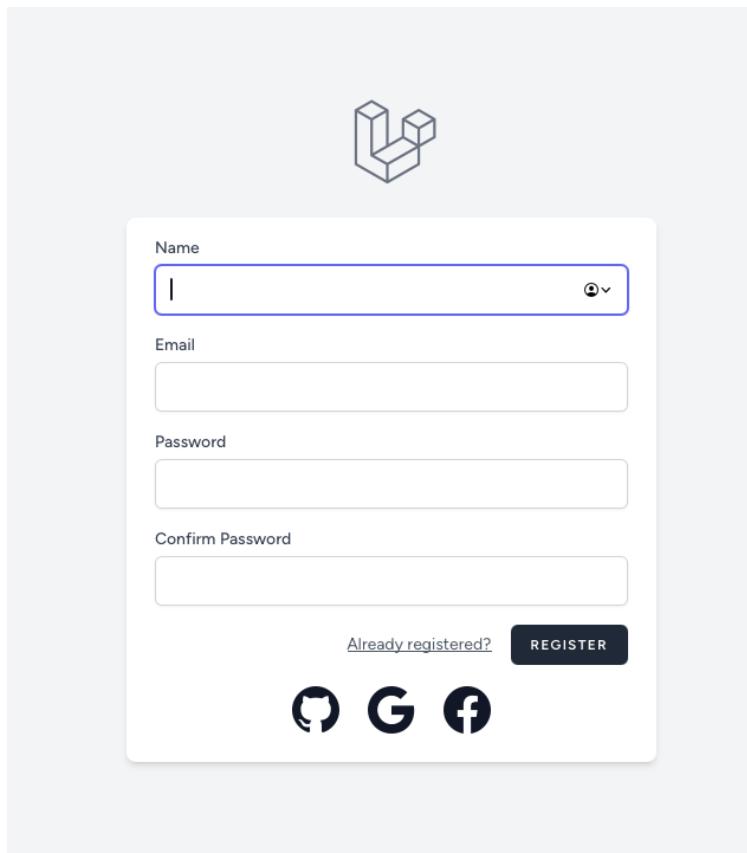


1. ábra: Site map az alkalmazás oldalainak kapcsolatáról



2. ábra: Site map az **Album** és **Generate a list** oldalakról

2.3.2. Bejelentkezés és Regisztráció



3. ábra: regisztrációs panel

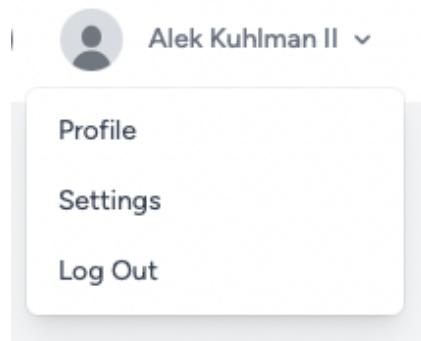
Az alkalmazást nem bejelentkezett felhasználók is tudják használni, azonban a főbb funkciókat csak regisztráció és bejelentkezés után tudják igénybe venni. Az alkalmazás lehetőséget biztosít arra, hogy a felhasználók egyszerűen regisztrálhassanak GitHub, Facebook vagy Google fiókjuk használatával, így nincs szükség új fiók létrehozására, ilyenkor a regisztrációs (register) vagy bejelentkező (login) oldalon (3. ábra) a megfelelő szimbólumot kiválasztva az oldal átirányítja a felhasználót a választott szolgáltató (GitHub, Facebook, Google) hitelesítő oldalára. Ha a hitelesítés sikeres akkor a felhasználót visszairányítja az oldalra és már is igénybe veheti az alkalmazás szolgáltatásait. Természetesen van lehetőség a „hagyományos” bejelentkezési és regisztrációs módra is (egyedi e-mail, felhasználónév, jelszó megadásával).

2.3.3. Fejléc funkciói



4. ábra: az alkalmazás fejléce, bejelentkezett felhasználó esetén

A fejléc (4. ábra) minden oldalon megjelenik és elérhetőek róla a főbb oldalak (Main Page, Profile, Lists). Továbbá rendelkezik egy kereső mezővel, ahol a felhasználó kiválaszthatja, hogy albumok, más felhasználók vagy előadók között szeretne böngészni. A fejléc jobb szélén látható a bejelentkezett felhasználó profilképe és a neve, utóbbira kattintva megjelenik egy kisebb menü. (5. ábra)



5. ábra: Profil menü

A **Profile** menüpont ugyanarra az oldalra irányít, mint a fejléc bal oldalán elhelyezkedő Profile link. A **Settings** oldalon érhetőek el a felhasználói fiók módosítási lehetőségei. Ezekhez tartozik a profil kép, felhasználó név, e-mail és jelszó módosítása. Továbbá itt lehet törleni a felhasználói fiókot. A **Log Out** lehetőséggel pedig kijelentkezhet a felhasználó.

2.3.4. Main Page (főoldal)

A főoldal három szekcióra oszlik:

- az **elsőben** az új megjelenések láthatók, amelyek a legfrissebb mainstream zenei albumokat tartalmazzák. (6. ábra)
- A **második** szekcióból a bejelentkezett felhasználók személyre szabott albumajánlásokat találnak, amelyek az aktivitásuk és preferenciáik alapján kerülnek összeállításra. (7. ábra)
- A **harmadikban**, az oldal alján az elmúlt napok felhasználói aktivitásai alapján rangsorolt legnépszerűbb albumok jelennek meg. (8. ábra)

New Releases



PLAYHOUSE
★ 3.2 / 5 (44 ratings) | 2024-11-07



ROOM UNDER THE STAIRS
★ 3.1 / 5 (37 ratings) | 2024-05-17



HIT ME HARD AND SOFT
★ 3.0 / 5 (43 ratings) | 2024-05-17



Radical Optimism
★ 2.9 / 5 (41 ratings) | 2024-05-03

Showing 1 to 4 of 20 results

< 1 2 3 4 5 >

6. ábra: Új megjelenések

Recommended Albums



The Black Saint And The Sinner Lady



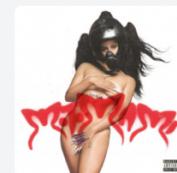
Karma



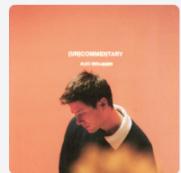
Journey in Satchidananda



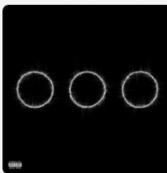
I Told Them...



MOTOMAMI



(Un)Commentary



Paradise Again



Versions of Me

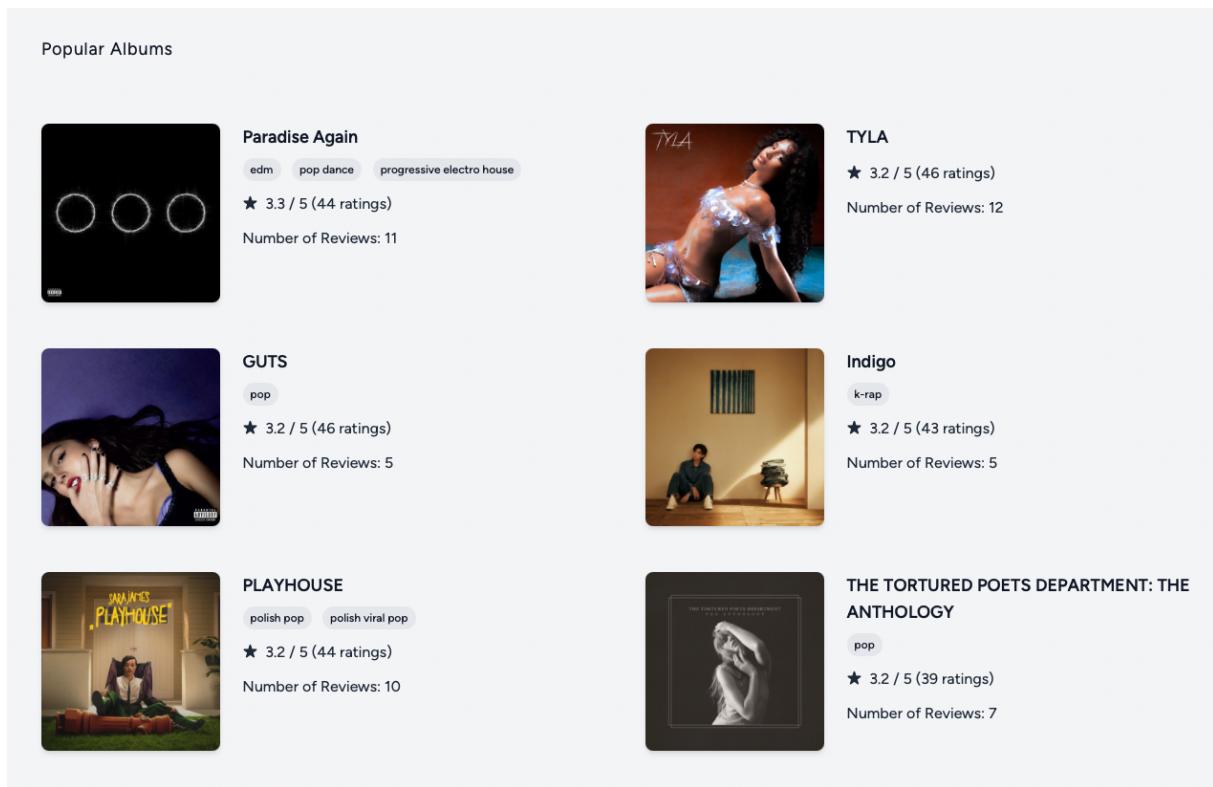


Harry's House



Indigo

7. ábra: személyre szabott ajánlások



8. ábra: népszerű albumok

2.3.5. Album oldal

Az alkalmazásban számos helyen megjelennek albumborítók, amelyek mindegyike kattintható linkként funkcionál. A borítókra kattintva a felhasználó az adott album „bemutató” oldalára (9. ábra) navigálhat, ahol megtekintheti az albumhoz kapcsolódó információkat például az előadó nevét, megjelenés dátumát, műfaj(okat), a dalok listáját a többi felhasználó összesített értékeléseit (számszerűen kiírva az átlag értékelés és az összes értékelés száma). Ezeken felül pedig böngészheti a többi felhasználó szöveges értékelését az adott albumról, amiket akár lájkolhat is a szív formájú emblémára kattintva.

A felhasználónak ezen az oldalon lehetősége van kifejteni véleményét "review" formájában a az albumról, illetve értékelni az albumot egy 1-től 5-ig terjedő skálán, csillagozással. Az albumokra adott értékelés később hozzájárul ahhoz, hogy az alkalmazás személyre szabott album ajánlásokkal tegye még teljesebbé az oldal használatát a felhasználó számára.

Az albumot elmentheti az **Add to Discovery Queue** gombra kattintva, amely egyfajta kívánságlistaként funkcionál, segítve a felhasználót, hogy ne feledkezzen el azokról az albumokról, amiket az oldalon fedezett fel és később szeretne meghallgatni.



Radical Optimism

Artist [Dua Lipa](#)
Released 2024-05-03
Rating ★ 2.9 / 5 (41 ratings)
Genres dance pop, pop, uk pop

Track list

End Of An Era	3:16
Houdini	3:05
Training Season	3:29
These Walls	3:37
Whatcha Doing	3:18
French Exit	3:21
Illusion	3:08
Falling Forever	3:43
Anything For Love	2:21
Maria	3:07
Happy For You	4:05

Rate the album!
Your rating: ★★★★☆

Add to
Discovery Queue

Reviews

Share your opinion!

[Post review](#)

 Kellie Frami 6 days ago

0

Magnam voluptas nesciunt tempora recusandae velit voluptate. Rerum officiis nobis quo qui. Dicta neque cum nam esse laudantium sit fuga iste. Voluptatibus reprehenderit tenetur aut adipisci.

9. ábra: Album bemutató oldal példa

Az album részletező oldalát két módon érhetjük el: egyszerűen, ha egy listában vagy a főoldalon találkozunk egy albumborítóval, arra kattintva közvetlenül megnyílik az adott album oldala. Másrészt célzottan is rákereshetünk egy albumra a keresőmező segítségével, majd a találatok között kiválasztva az albumot, annak oldalára navigálhatunk.

2.3.6. Előadó oldal

Bármely album oldalán az előadó nevére kattintva eljuthatunk az előadó saját oldalára (10. ábra), ahol megtekinthető a teljes diszkográfiája, valamint azok az albumok, amelyeket a felhasználó már értékelte, csillagozással vagy szöveges formában.



John Coltrane

Bebop Free jazz Jazz Jazz saxophone Vocal jazz

Albums by This Artist You've Reviewed/Rated

1.82% (2 / 110)

[View Discography](#)

10. ábra: az előadó oldal

Az **Albums by This Artist You've Reviewed/Rated** linken keresztül érhetőek el azok az albumok, amiket a felhasználó már értékelt ettől az előadótól időrendi sorban kilistázva. (11. ábra)



John Coltrane

Albums You've Interacted With

2022 June

15 Jun  ★★★★  Sun Ship Jun 15, 2022

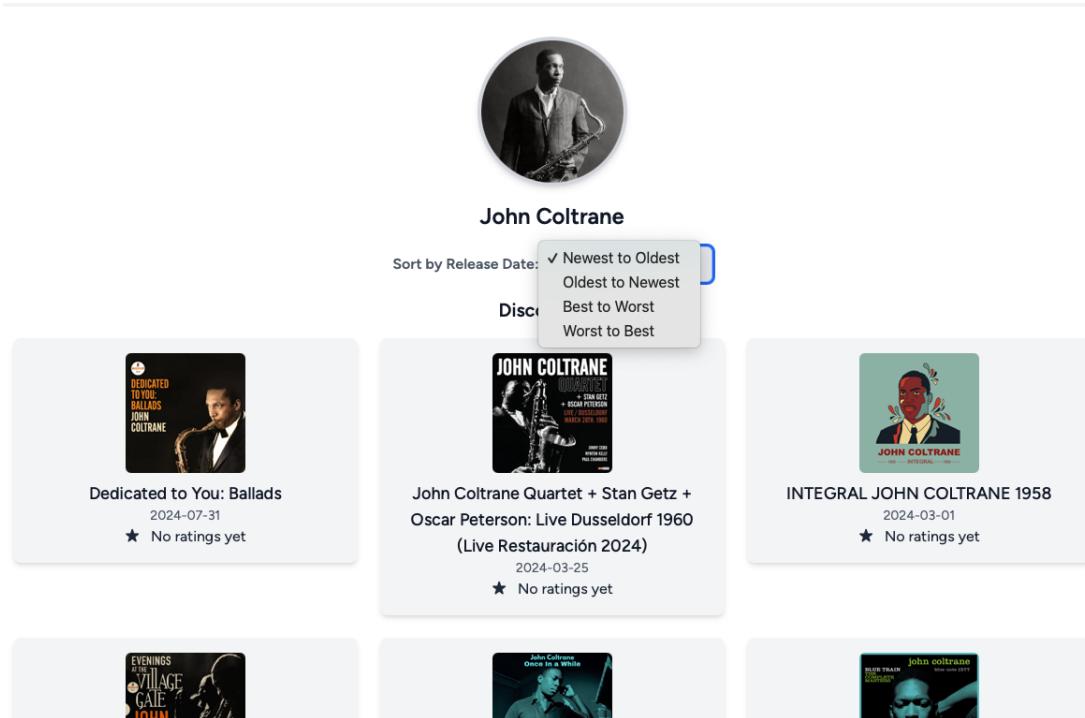
2021 May

10 May  A Love Supreme May 10, 2021

11. ábra: a felhasználó aktivitása az adott előadónál

A **View Discography** linkre kattintva pedig az előadó teljes diszkográfiáját lehet megtekinteni (12. ábra), szűrési lehetősséggel. Legnépszerűbb albumjától a legkevésbé népszerűig és

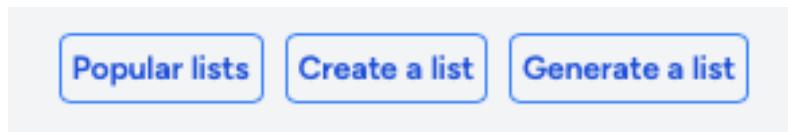
fordítva, továbbá az időrendi sorrendet is lehet módosítani (legújabbtól a legrégebbig és fordítva).



12. ábra: az előadó diszkográfiája

2.3.7. Toplista oldal

A fejlécen elhelyezkedő **Lists** menüpontra kattintva érhetjük el a lista oldalt. Itt a bejelentkezett felhasználónak lehetősége van kiválasztani egy navigációs menüben (13. ábra), hogy a már létező, többi felhasználó által létrehozott toplisták között szeretne böngészni egy kereső mező segítségével vagy pedig új listát szeretni létrehozni, vagy esetleg műfaj alapján szeretne listát generálni (alapértelmezetten a lista böngésző oldal jelenik meg).



13. ábra: Lista menü

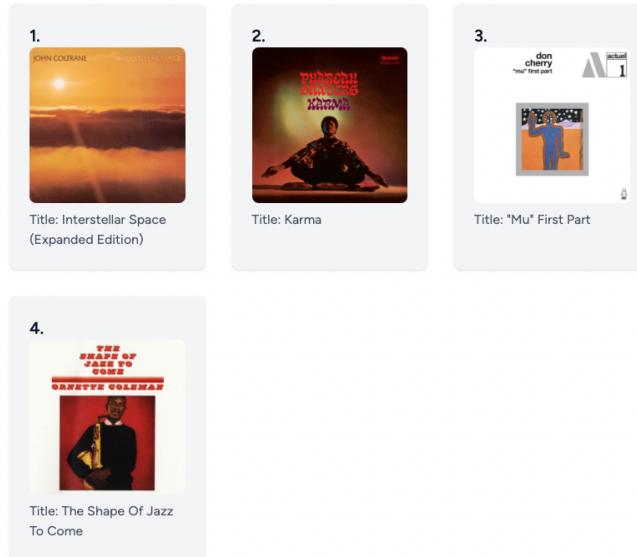
2.3.8. Saját toplista létrehozása

Első lépésként a felhasználónak címet kell adnia a leendő listának, ezt követően elkezdheti albumokkal feltölteni az **Edit / Add albums** gombra kattintva, amely átirányítja a lista birtokost a módosító oldalra.

best Free jazz albums

0

[Edit / Add albums](#) [Delete list](#)



14. ábra: példa toplista

Az **edit list** oldalon (15. ábra) bővítheti a felhasználó új albumokkal a listáját a kereső mező használatával. Ezen felül módosíthatja azok sorrendjét vagy akár ki is törölhet albumokat a listáról. Maximum 20 album adható hozzá egy-egy listához. A **Delete list** gombbal pedig az egész lista törölhető a rendszerből.

#	Cover	Title	Delete
No albums added yet.			

SAVE

Interstellar Space (Expanded Edition)
by John Coltrane

Add to the list **Search again**

15. ábra: lista kreálás

4 / 20 albums

#	Cover	Title	Delete
1		Interstellar Space (Expanded Edition)	
2		Karma	
3		"Mu" First Part	
4		The Shape Of Jazz To Come	

SAVE

16. ábra: albumok listája a módosító oldalon

Az albumok sorszáma mellett elhelyezkedő "drag-and-drop" szimbólummal "megragadhatják" a felhasználók az albumokat és azokat tetszésük szerint húzhatják a kívánt helyre, ha módosítani szeretnék a sorrendet (16. ábra). A program ezután automatikus frissíti a sorszámokat. A piros kuka szimbólumra kattintva pedig törlhetik az adott albumot. A rendszer figyelmezteti a felhasználót, ha olyan albumot szeretne adni a listához, amit már tartalmaz (17. ábra). Hasonló hibaüzenettel jelzi a felhasználónak az alkalmazás, ha már megtelt a lista (maximum 20 album), de új albumot szeretne hozzáadni.

This album is already in your list.

2 / 20 albums

#	Cover	Title

17. ábra: Figyelmeztető szöveg ha a felhasználó egy olyan albumot ad a listához amit már az tartalmaz

2.3.9. Toplista generálás

A lista generáló oldalon a felhasználó először kiválaszthatja a kívánt műfajt. Az oldal ezután a kiválasztott műfajhoz tartozó albumokból összeállít egy toplistát, amelyet a felhasználó kedve szerint rendezhet legjobbtól legrosszabbig vagy fordítva. (18. ábra)

A generáló oldal kétféleképpen érhető el: egyrészt a **Lists** oldalról, másrészt az alkalmazás különböző pontjain megjelenő műfajokra kattintva. Például, ha egy album oldalán egy műfaj nevére kattint a felhasználó, az adott műfajhoz tartozó albumok jelennek meg a generáló oldalon.

Generated List

Sort by: Best to Worst ▾

Selected genre: Free jazz

Select new genre

Rank	Album	Rating
1	 The Black Saint And The Sinner Lady	★ 4.5 / 5 (2 ratings)
2	 Karma	★ 4.5 / 5 (2 ratings)
3	 Symphony For Improvisers (Remastered / Rudy Van Gelder Edition)	★ 4.0 / 5 (3 ratings)
4	 Sun Ship	★ 3.5 / 5 (4 ratings)

18. ábra: generált lista példa

2.3.10. Profiloldal

A bejelentkezett felhasználó a saját profiloldalát (19. ábra) könnyedén elérheti a fejlécen található **Profile** navigációs linkre kattintva. Más felhasználók oldalát pedig az alkalmazás több pontjáról is elérheti, egyfelöl a fejlécben elhelyezkedő kereső mezőt használva, célzottan

rákereshet bármely felhasználóra. Továbbá az alkalmazás számos pontján - például egy szöveges értékelés fejlécén - megjelenő felhasználónévre kattintva.

A felhasználók a saját profiloldalukon tekinthetik meg legutóbbi tevékenységüket. Az egyes albumoknál fel van tüntetve hány csillaggal értékelték az albumot, vagy írt-e szöveges értékelést az albumhoz, utóbbit egy szöveg buborék ikon jelzi, amire, ha rákattint a felhasználó akkor láthatja a véleményt, amit írt. Ezen felül a felhasználó a **Discoveries** gombra kattintva megtekintheti az általa elmentett albumokat. A **Lists** gombra kattintva pedig hozzáférhet saját toplistáihoz, valamint azokhoz a listákhoz is, amelyeket más felhasználóktól kedvelt.

The screenshot shows a user profile for 'Alek Kuhlman II'. At the top is a placeholder profile picture. Below it is the user's name, 'Alek Kuhlman II'. Underneath the name are three buttons: 'Activity' (highlighted in blue), 'Discoveries', and 'Lists'. To the left of the buttons is the text 'Recent Activity' and a link 'See all the activity at the Diary page'. Below these are five thumbnail cards, each representing a different activity:

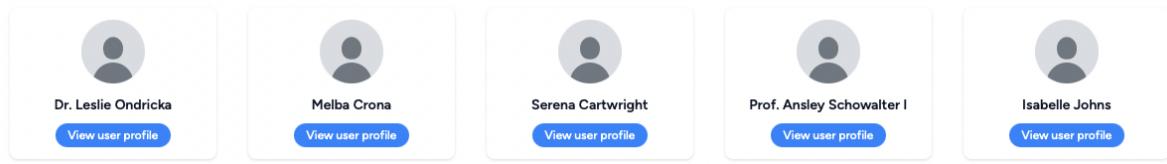
- Sun Ship**: A thumbnail of a person playing a brass instrument. Rating: ★★★★☆. Description: 'John Coltrane sun ship'.
- Symphony For Improvisers (Remastered / Rudy Van Gelder Edition)**: A thumbnail of a person in a colorful sweater. Rating: ★★★★☆. Description: 'STEREO SYMPHONY FOR IMPROVISERS DON CHERRY'.
- Just Another Diamond Day**: A thumbnail of a painting of a woman in a blue dress standing next to two horses. Rating: ★★★★☆. Description: 'VASILI POLENOV Just Another Diamond Day'.
- Lookaftering**: A thumbnail of a brown dog. Rating: ★★★★☆. Description: 'Ladyfinger Pepe Goyze'.
- Crumbling**: A thumbnail of a person walking on a beach. Rating: ★★★★☆. Description: 'Crumbling'.

19. ábra: felhasználó legutóbbi 5 aktivitása

Ahogy egyre több felhasználót követ, az alkalmazás létrehoz egy tevékenység panelt, ahol láthatja a felhasználó az általa követett emberek legutóbbi tevékenységét. Ez a rész a bejelentkezett felhasználó legutóbbi tevékenységét bemutató panelhez hasonlóan néz.

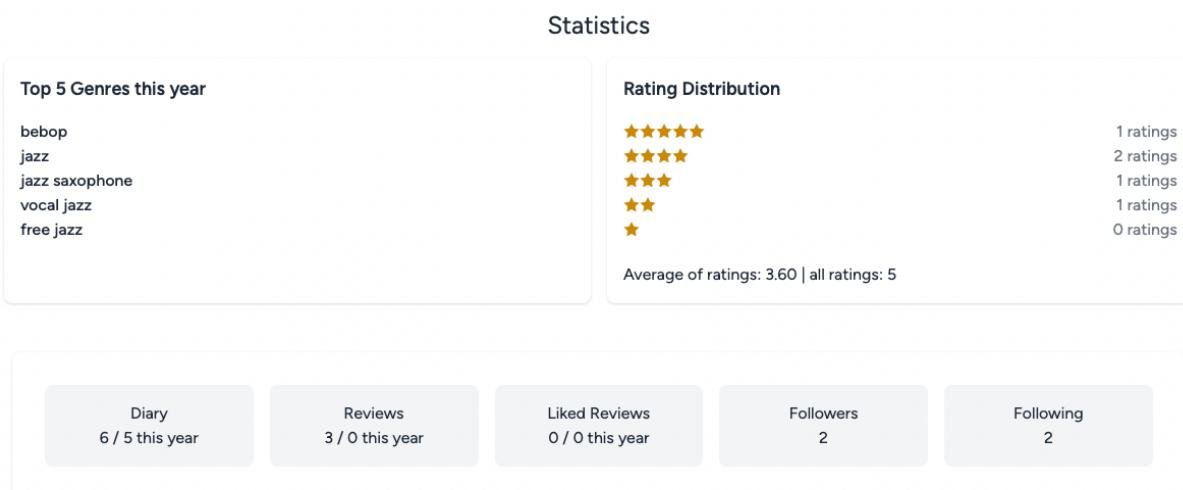
A profiloldalon követési ajánlások is helyet kapnak (20. ábra), ahol új és releváns felhasználókat fedezhet fel a felhasználó. Új embereket a **follow** gomb megnyomásával tud bekövetni.

Users to follow



20. ábra: követési ajánlások

Az alkalmazás folyamatosan frissíti a profiloldalon található **Statistics (statisztikák)** panelt (21. ábra), ahol a felhasználó saját tevékenységéről talál műfaj elemzést, illetve elérheti a különböző menüpontokra kattintva az összesített tevékenységét naplószerűen (**Diary**), az általa írt, illetve kedvelt szöveges értékeléseket (**Reviews, Liked Reviews**), továbbá az általa követett és az őt követő felhasználókról listát (**Followers, Following**).



21. ábra: a felhasználó statisztikái

3. Fejlesztői dokumentáció

3.1. Az ajánló rendszer

Az alkalmazás fontos részét képezi a személyre szabott ajánlórendszer, amely egy különálló microservice-ként működik. A webalkalmazás API-hívásokon keresztül kommunikál az ajánló rendszerrel. Az alkalmazás továbbítja az összes elérhető felhasználó értékelési adatait (mely albumokat értékelték, és milyen pontszámot adtak) és a cél felhasználó egyedi azonosítóját, akinek a rendszer az ajánlásokat fogja generálni. Az ajánlórendszer ezeket az adatokat feldolgozza, majd visszaküldi az adott felhasználó számára releváns albumajánlásokat és a hasonló felhasználókat, követési javaslatként.

Az ajánló rendszer az *API* mappában található és 4 fájlból áll:

- ***main.py***: Ez a fájl az ajánló rendszer központi eleme, amely kezeli a beérkező API kéréseket, tartalmazza az ajánlási logikát.
- ***.env***: A hitelesítéshez használt titkos kulcsot tartalmazza.
- ***requirements.txt***: Az ajánlórendszer működéséhez szükséges Python-könyvtárak listája, amelyek könnyen telepíthetők ezzel a fájllal.
- ***test_recommendations.py***: Az ajánlórendszer működésének és pontosságának ellenőrzésére szolgáló tesztfájl.

3.1.1. Felhasznált technológiák – Python, FastAPI

A személyre szabott ajánlórendszer alapját a FastAPI webes keretrendszer és a Python programozási nyelv számos könyvtára alkotja.

A FastAPI egy olyan Python alapú modern keretrendszer, amely segítségével gyors és megbízható API-kat lehet fejleszteni. Könnyebb használni, mint a piacra fellelhető más hasonló rendszereket (pl. Flask), nem mellesleg nagyobb teljesítménnyel bír, és több beépített funkcionálitást tartalmaz. Továbbá az igényesen szerkesztett és gazdag dokumentációjának köszönhetően könnyű megtanulni használni. Hátránya viszont, hogy költséges nagyobb projektekhez használni.[1]

Az ajánló rendszer megvalósításához a FastAPI-n kívül a Python nyelv különböző könyvtárait használtam: *pandas*, *scikit-learn*, *python-dotenv*.

- **pandas:** Adatelemző és adatmanipulációs eszköz, amely lehetővé teszi az adatok strukturált feldolgozását. Az ajánlórendszer az értékelési adatok csoportosítására és az átlagértékek kiszámítására használja.
- **scikit-learn:** Egy gépi tanulási könyvtár, amely kulcsszerepet játszik az ajánlórendszer hasonlósági számításaiban. A **cosine similarity** algoritmus segítségével azonosítja a hasonló felhasználókat és súlyozza azok értékeléseit (következő fejezetekben erről részletesebben szót fogok ejteni).
- **python-dotenv:** Az érzékeny adatok, például API-tokenek és környezeti változók kezelésére szolgáló eszköz. Ez biztonságosabbá és áttekinthetőbbé teszi az alkalmazás konfigurációját.

3.1.2. Collaborative Filtering, Cosine Similarity

Az ajánlórendszereknél két fő típus létezik:

- Content-Based: Az ajánlások a felhasználó által korábban preferált tételek tulajdonságain alapulnak. Például, ha egy felhasználó szereti a metál zenét, akkor több metál albumot ajánl.[3]
- Collaborative filtering: Az algoritmus megkeresi azokat a felhasználókat, akik hasonló preferenciákkal rendelkeznek, és az ő választásaik alapján ajánl elemeket.[3]

Az ajánlórendszer fejlesztése során a Collaborative filtering módszerét választottam. Bár a Content-based megközelítés számos előnyvel bír, például hatékonyan működik egyedi ízlésű felhasználók esetében, és nem igényli más felhasználók adatainak felhasználását, a Collaborative filtering megközelítés különlegesebb ajánlásokat képes generálni, továbbá lehetőséget biztosít arra is, hogy a felhasználók számára más, hasonló érdeklődésű felhasználókat ajánljon követési javaslatként. Viszont fontos megjegyezni, hogy hátrányához tartozik az, hogy új felhasználók vagy elemek esetén kevés az adat az ajánlások számításához. Továbbá nagy felhasználó vagy elemkészlet esetén az algoritmus számítási igénye gyorsan növekedhet.[3]

A Collaborative filtering alapját képezi a cosine similarity (22. ábra) amely „két nem nullvektor közötti hasonlóság mértéke, amelyet széles körben alkalmaznak gépi tanulási és adatalemzési feladatokban. Valójában a két vektor közötti szög koszinuszát méri, így képet ad arról, hogy a két vektor mennyire mutat azonos irányba, függetlenül azok

nagyságától. Gyakran használják szövegelemzési feladatokban, például dokumentumok vagy keresési lekérdezések közötti hasonlóság összehasonlítására, valamint ajánlórendszerben, hogy a felhasználók preferenciáit egyeztesse.”[5]

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

22. ábra: cosine similarity[4]

3.2. Az ajánló rendszer működése

A kód egy FastAPI alapú microservice-t valósít meg, amely kommunikációs interfésként működik az ajánlórendszerhez. A szolgáltatás fogadja a webalkalmazás kéréseit (API hívások formájában), feldolgozza azokat, majd az eredményeket visszaküldi a kliensnek.

3.2.1. Hitelesítés

A kommunikáció biztonságossága érdekében a *verify_token* függvény ellenőrzi a kérések érvényességét. Ehhez a küldő félnek egy Bearer¹ típusú hitelesítési tokent kell küldenie az *Authorization* fejlécben. Ha a token nem egyezik meg a *.env* fájlban tárolt titkos kulccsal, a rendszer "Unauthorized" hibát dob.

A rendszer a **Pydantic** könyvtár **BaseModel** osztályát használja adatmodell-ellenőrzésre és struktúrák definiálására:

- **Rating:** Egyetlen albumra vonatkozó értékelési adatokat írja le.
- **RecommendationRequest:** Az API hívás teljes struktúráját tartalmazza.

¹ „A bearer token egy titkos karakterlánc, amelyet általában a szerver generál egy bejelentkezési kérésre adott válaszként. Az ügyfélnek ezt a tokent kell elküldenie az Authorization fejlécben, amikor védett erőforrásokhoz próbál hozzáérni.”[2]

3.2.2. Kommunikáció

A rendszer a `/recommend` végpontra várja az ajánlási kéréseket, amelyeknek tartalmaznia kell a következőket (példa request body: 23. ábra):

- **Felhasználói azonosító:** Annak a felhasználónak az egyedi id-ja, akinek az ajánlásokat kéri a kliens.
- **Felhasználói adatok:** Egy dictionary (szótár, python adatszerkezet), amely az összes elérhető felhasználó értékelési adatait tartalmazza (milyen albumot értékelt, milyen értékkel, 1-5-ig terjedő skálán).
- **Követett felhasználók:** Azok a felhasználók, akiket az aktuális felhasználó már követ.
- **Szövegesen értékelt és a kívánságlistára helyezett albumok:** Azon albumok listája, amelyeket a felhasználó már értékelt szövegesen vagy a kívánságlistájára (Discovery queue) helyezett. Erre azért van szükség, hogy a rendszer ne ajánljon olyan albumokat, amelyeket bár nem értékelt az 1-5-ig terjedő skálán a felhasználó, de írt már róla szöveges véleményt vagy már szerepel a kívánságlistán, tehát tudomása van az adott albumról ezért nem szükséges ajánlani neki külön.

```
{
    "user_id": 1,
    "all_users_data": {
        "1": [
            { "album_id": 101, "rating": 4.5 },
            { "album_id": 102, "rating": 5.0 }
        ],
        "2": [
            { "album_id": 101, "rating": 4.0 },
            { "album_id": 103, "rating": 3.5 }
        ],
        "3": [
            { "album_id": 104, "rating": 2.0 },
            { "album_id": 105, "rating": 3.0 }
        ]
    },
    "following": [2],
    "reviewed_albums": [103, 104],
    "discovery_queue": [105]
}
```

23. ábra: request body példa

3.2.3. Ajánlások generálása

A *recommend_albums* függvény a felhasználók értékelési adatait felhasználva személyre szabott album ajánlásokat generál, valamint hasonló felhasználókat keres. Ennek működése az alábbi lépésekkel áll:

- **1. Átlagos értékelések és szűrés**
- **Cél:** Csak olyan albumokat vegyen figyelembe az ajánlórendszer, amelyek elegendő számú értékelést kaptak.
- **Működés:**
 - A *ratings_df* (értékeléseket tartalmazó DataFrame) alapján kiszámítja az albumonkénti átlagos értékeléseket és az értékelések számát (*avg_rating, number_of_ratings*).
 - Szűri azokat az albumokat, amelyek nem haladják meg az előre meghatározott minimális értékelésszámot (*number_of_ratings*).
- **2. Pivot tábla létrehozása**
- **Cél:** Egy olyan táblázat készítése, ahol a sorok felhasználókat, az oszlopok albumokat, a cellák pedig az értékeléseket reprezentálják.
- **Működés:**
 - Az értékelési adatokat pivot táblává alakítja (*albums_ratings_pivot*), ahol a hiányzó értékeket nullával tölti ki. Lehetséges példa táblázat: (24. ábra)

user_id	album_101	album_102	album_103
1	4	3	0
2	5	0	2
3	0	0	5

24. ábra: példa pivot táblázat szemléltetésnek

- **3. Hasonlóság kiszámítása (Cosine similarity)**
- **Cél:** Megállapítani, hogy az adott felhasználó mely más felhasználókhöz hasonlít a legjobban az értékelései alapján.
- **Működés:**
 - Az **scikit-learn** könyvtárból kölcsönöztött *cosine_similarity* függvény segítségével kiszámítja a felhasználók közötti hasonlósági mátrixot (25. ábra).
 - Kiválasztja az aktuális felhasználóhoz (*user_id*) leginkább hasonló felhasználókat.

```
similarity_matrix = cosine_similarity(albums_ratings_pivot)
similarity_matrix_df = pd.DataFrame(similarity_matrix, index=albums_ratings_pivot.index,
                                     columns=albums_ratings_pivot.index)
```

25. ábra: kód részlet a *cosine_similarity* függvény alkalmazásáról

- **4. Követett felhasználók kizárása**
- **Cél:** Az aktuális felhasználó által követett személyeket ne vegye figyelembe a hasonló felhasználók listájában.
- **Működés:**
 - A kapott hasonló felhasználók listájából (*similar_users*) kizárja a *following* paraméterben kapott felhasználókat.
 - A legjobb 20 hasonló felhasználót (*top_similar_users*) választja ki a maradékból.
- **5. Nem értékelt albumok szűrése**
- **Cél:** Csak azokat az albumokat ajánlja, amelyeket az aktuális felhasználó még nem értékelt semmilyen módon, beleértve a szöveges értékeléseket, illetve nem szerepel a kívánságlistán sem (**Discovery queue**).

- **Működés:**
 - Megkeresi azokat az albumokat, amelyeket az aktuális felhasználó nem értékelt (*albums_ratings_pivot.loc[user_id, :] == 0*).
 - Eltávolítja azokat az albumokat, amelyek a *reviewed_albums* vagy *discovery_queue* listákban szerepelnek.
- **6. Súlyozott ajánlások kiszámítása**
- **Cél:** Az albumokhoz tartozó ajánlási pontszámok kiszámítása a hasonló felhasználók értékelései alapján.
- **Működés:**
 - Az albumok értékeléseit súlyozottan átlagolja a hasonló felhasználók hasonlósági pontszámaival.
 - Az albumokat ajánlási pontszám alapján rendezzi, és kiválasztja a legjobb 10-et (*top_recommendations*).

3.2.4. Ajánlások visszaküldése

Az ajánló rendszer a fentebb említett lépések után összesíti a kapott elemeket és visszaküldi a kliensnek az ajánlott albumok és a leginkább hasonló felhasználók azonosítóinak és hasonlósági pontszámainak listáját.

3.3. A Webalkalmazás

Az alkalmazás másik fontos részét képezi a Laravel keretrendszerben fejlesztett webalkalmazás, amelyet a korábban bemutatott személyre szabott ajánló rendszer szolgál ki ajánlásokkal. Az ajánló rendszer nélkül is egy teljes értékű alkalmazásként funkcionál.

Az webalkalmazás fontos részét képezi a Spotify Web API integrációja, ugyanis ez a külső API látja el az alkalmazást a szükséges album, előadó és műfaj információkkal. Ezzel a külső API szolgáltatással való kommunikációt a későbbi fejezetekben mutatom be.

A következő fejezetekben részletes bemutatásra kerül a webalkalmazás felépítése, a használt technológiák, valamint a legfontosabb funkciók és rétegek működése.

A webalkalmazás a *MusicApp* mappában található. Számos mappából és fájlból áll, ezek felsorolására szintén a későbbi fejezetekben kerül sor.

3.4. Felhasznált technológiák

3.4.1. Keretrendszer – Laravel

A piacon számos full-stack keretrendszer található (pl. Ruby on Rails, Django, ASP.NET). Az én választásom a PHP nyelvre épülő Laravel-re esett mivel egy backend-központú alkalmazás megvalósítása volt a cél. A Laravel pedig egy igazán gazdag és könnyen használható eszköztárral bír, amivel hatékony backend rendszereket lehet építeni. További előnyeihez sorolható más keretrendszerhez képest, hogy könnyű megtanulni használni igényesen szerkesztett dokumentációjának és „elegáns szintaxisának” köszönhetően. Azonban fontos megjegyezni, hogy lassabb versenytársaihoz képest. Például a Ruby on Rails (külön érdekesség, hogy ez a keretrendszer szolgált alapként a Laravelhez) gyorsabb és hatékonyabb, mivel követi a „Konvenció a konfiguráció felett” elvét.[6][7]

A korábban említett Ruby on Rails-hez hasonlóan a Laravel is követi a Model-View-Controller (MVC) architektúrát, erről a későbbi fejezetekben lesz szó részletesebben.

3.4.2. Frontend – Blade, Livewire

A webalkalmazás frontend részét két fő komponens alkotja, a **Blade** és a **Livewire**, előbbi a statikus HTML rendereléséért, utóbbi pedig az interaktív komponensekért felel.

- **Blade:** „A Blade egy könnyű és elegáns sablonnyelv, amely egyszerű és olvasható szintaxist biztosít az adatok megjelenítéséhez, iterálásához és dinamikus tartalmak kezeléséhez, miközben elősegíti a tiszta és strukturált kódírást.”[9]
- **Livewire:** Egy keretrendszer, amely lehetővé teszi interaktív, dinamikus komponensek egyszerű létrehozását, közvetlenül a Blade fájlokban.[10]

A Laravel lehetőséget biztosít számos frontend keretrendszer integrálására, mint például a React.js. A fejlesztés során a Livewire használata mellett döntöttem mivel ezt a keretrendszert kifejezetten a Laravelhez terveztek, ugyanúgy a PHP nyelvet használja, ezáltal rendkívül kényelmes együtt használni a kettő rendszert. Habár nem rendelkezik más hasonló keretrendszerök összetettségével és funkcionálitásával, a Livewire megfelelt az alkalmazás igényeinek.[11]

3.4.3. Tailwind

CSS-nek a Tailwind-et használtam a fejlesztés során, mivel ezzel az eszközzel gyorsan és hatékonyan lehet frontend stílusokat létrehozni.[12]

3.4.4. Adatbázis – PostgreSQL

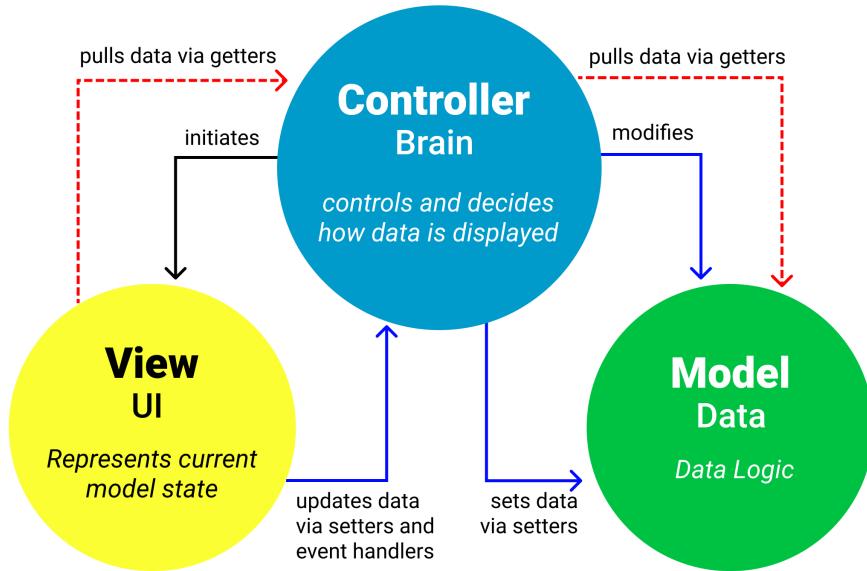
A fejlesztés során a PostgreSQL-t használtam adatbázisnak. A PostgreSQL egy robusztus, megbízható és rendkívül skálázható relációs adatbázis-kezelő rendszer[13], amelyet Docker konténerben futtattam. Más adatbázisokat, mint például a MySQL-t könnyebb beüzemelni és használni, azonban rosszabb skálázhatósággal bír és az összetettebb műveletek végrehajtása nehezebb és lassabb.[15]

A Docker használata lehetővé teszi az adatbázis gyors és egyszerű beállítását, miközben biztosítja a fejlesztési környezet hordozhatóságát.[14]

3.5. A Webalkalmazás Rétegei és Működése

A Laravel a Model-View-Controller (MVC) architektúra elveire épül, amely lehetővé teszi az alkalmazás logikai rétegeinek elkülönítését. (26. ábra). Ez a struktúra elősegíti a kód átláthatóságát és karbantarthatóságát azáltal, hogy az adatkezelést (Model), a megjelenítést (View), és az alkalmazás logikai vezérlését (Controller) külön komponensekbe szervezi. Az MVC megközelítés nemcsak a fejlesztési folyamatot teszi hatékonyabbá, hanem megkönnyíti az új funkciók hozzáadását is az alkalmazáshoz.[7][8]

MVC Architecture Pattern



26. ábra: MVC architektúra ábrázolása [8]

3.6. Adatbázis Modellek (Models)

Az MVC architektúrában az egyik legfontosabb réteg a Model, amely az alkalmazás adatainak kezeléséért felel. A Laravel keretrendszerben a modellek az adatbázis táblák reprezentációját biztosítják, és lehetőséget nyújtanak az adatok lekérdezésére, manipulálására és az üzleti logika megvalósítására. A modellek a következő módon épülnek fel:[16]

Migration fájlok (*MusicApp / database / migrations mappa*)

A migration fájlok felelősek az adatbázis struktúrájának kezeléseért. Ezek a fájlok tartalmazzák az adatbázis táblák létrehozásához, módosításához vagy törléséhez szükséges utasításokat.

- Tábla létrehozása:** A migration fájlok segítségével hozhatunk létre új táblákat és meghatározhatjuk azok oszlopait.
- Oszlopok és adattípusok:** A migration fájlokban megadjuk az oszlopokat, azok adattípusait, az esetleges alapértelmezett értékeket, valamint a kapcsolatok, például idegen kulcsok beállítását.

- **Migrálás és visszavonás:** A migrálás során az adatbázis struktúráját frissítjük (pl. új tábla hozzáadása), míg a visszavonás lehetővé teszi, hogy az adatbázis változtatásait visszavonjuk.[16]
- **Példa Migration fájl:** (27. ábra)

```
public function up(): void
{
    Schema::create('users', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->string('email')->unique();
        $table->timestamp('email_verified_at')->nullable();
        $table->string('password')->nullable();
        $table->string('profile_picture')->nullable();
        $table->string('bio')->nullable();
        $table->rememberToken();
        $table->timestamps();
    });
}
```

27. ábra: példa adatbázis schema (users)

Model fájlok (*MusicApp / App / models mappa*)

A **model fájlok** az adatbázis táblák PHP osztályai, ezek felelősek az adatok kezeléséért. A modellek az Eloquent ORM (Object-Relational Mapping) segítségével kezelik az adatbázis rekordokat, így a modellek közvetlenül kapcsolódnak az adatbázis táblákhoz.

- **Adatok lekérése és módosítása:** A modellek segítségével lekérdezhetjük, beszűrhetjük, frissíthetjük vagy törölhetjük az adatokat.
- **Kapcsolatok:** A modellek segítségével beállíthatjuk a táblák közötti kapcsolatok típusait, például egy-az-egyhez (one-to-one), egy-a-többhöz (one-to-many), vagy több-a-többhöz (many-to-many).
- **Példa Model fájlra:** (28. ábra) A példa Artist model tartalmaz egy albums metódust, amely az Album és az Artist modellek közötti kapcsolatot definiálja (egy artist több albummal is rendelkezhet).[17]

```

class Artist extends Model
{
    use HasFactory;

    no usages

    protected $fillable = [
        'spotify_id'
    ];

    • Marczis Bálint
    public function albums()
    {
        return $this->belongsToMany( related: Album::class, table: 'artist_album' );
    }
}

```

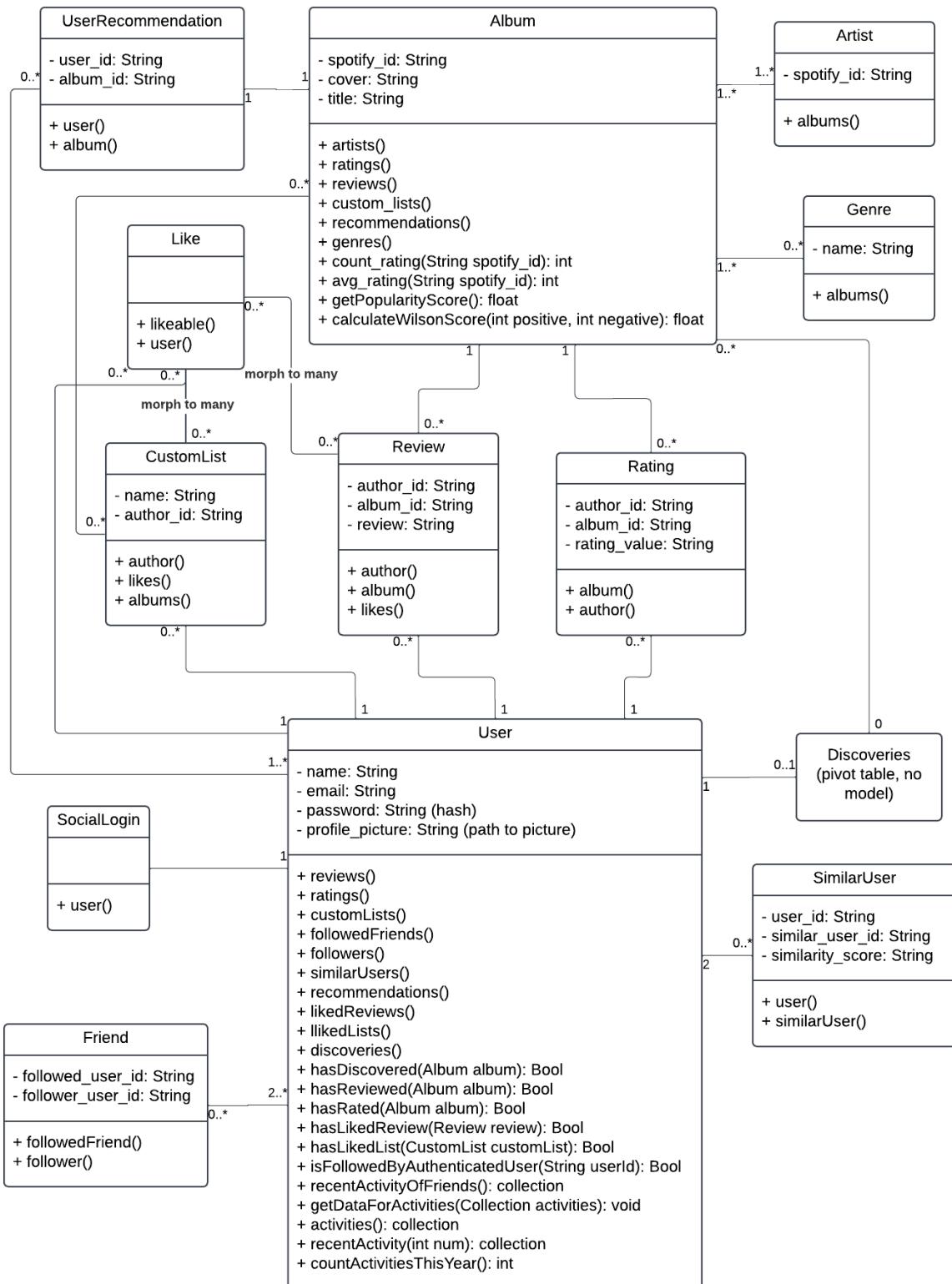
28. ábra: model fájl példa (Artist model)

Kapcsolat a Migration és Model fájlok között

- A migration fájlokban meghatározzuk az adatbázis táblát és annak struktúráját, míg a model fájlok segítségével dolgozunk az adatbázis rekordjaival, tehát az adatokkal.
- Mivel a modellek Eloquent ORM-et használnak, azok automatikusan a migrációkban létrehozott táblákhoz kapcsolódnak, ha a tábla neve megegyezik a modell nevének többszörösével (pl. Album modell az albums táblához).
- A **tábla és modell közötti kapcsolatot** a Laravel automatikusan kezeli, de a migrációk és modellek szoros együttműködése biztosítja, hogy a rendszer adatbázis-struktúrája és a program logikai adatmodellje egyensúlyban legyenek.[17]

3.6.1. Az Alkalmazás Adatmodelljei

UML diagram az adatmodellek felépítéséről és kapcsolataikról: (29. ábra)



29. ábra: az adatbázis modellek és kapcsolataik diagramma

3.6.2. Az adatmodellek részletezése és a fontosabb függvények:

1. User Modell

A *users* tábla (30. ábra) reprezentációja, amely az alkalmazás regisztrált felhasználóit kezeli.

Mező neve	Adattípus	Nullable	Korlátozások
id	INTEGER	NOT NULL	PRIMARY KEY
name	VARCHAR	NOT NULL	
email	VARCHAR	NOT NULL	UNIQUE
email_verified_at	TIMESTAMP	NULL	
password	VARCHAR	NULL	
profile_picture	VARCHAR	NULL	
remember_token	VARCHAR	NULL	
created_at	TIMESTAMP	NULL	
updated_at	TIMESTAMP	NULL	

30. ábra: users adatbázistábla mező szerkezeti leírása

Fontosabb metódusok:

- **activities** és **recentActivity**: A felhasználó aktivitásainak lekérdezése, beleértve az értékeléseket és véleményeket.
- **recentActivityOfFriends**: Lekérdezi a felhasználó követett barátainak legutóbbi aktivitásait.
- **getDataForActivities**: Segít az adatokat összegyűjteni az aktivitásokhoz, például album információkat és a felhasználó aktivitásait.

2. Album Modell

- Az *albums* tábla (31. ábra) reprezentációja. Az Album modell az albumokkal kapcsolatos adatokat kezeli, mint például a cím, a Spotify ID, a borítókép, és az

albumhoz tartozó kapcsolatok, például az előadó, értékelések, vélemények, műfajok és egyéni listák.

Mező neve	Adattípus	Nullable	Korlátozások
<code>id</code>	INTEGER	NOT NULL	PRIMARY KEY
<code>spotify_id</code>	VARCHAR	NOT NULL	UNIQUE
<code>cover</code>	VARCHAR	NOT NULL	
<code>title</code>	VARCHAR	NOT NULL	
<code>created_at</code>	TIMESTAMP	NULL	
<code>updated_at</code>	TIMESTAMP	NULL	

31. ábra: albums adatbázistábla mező szerkezeti leírása

Fontosabb metódusok:

- **`getPopularityScore()`:** Számítja az album népszerűségi pontszámát az utolsó 7 nap értékelései alapján, figyelembe véve a különböző értékelések számát (1-5 csillag).
- **`calculateWilsonScore()`:** A népszerűségi pontszám számításához használt *Wilson-score intervallumot* alkalmazza, hogy megbecsülje az album valódi népszerűségét.

3. CustomList Model

- A `custom_lists` tábla reprezentációja. A modell a felhasználók által létrehozott toplistákat kezeli. Tartalmazza az egyes listák nevét, és hozzárendeli a létrehozó felhasználót. Emellett kapcsolódik a felhasználókhöz, akik kedvelhetik a listát, és az albumokhoz, amelyek szerepelnek benne.

4. Friend Modell

- A `friends` pivot tábla reprezentációja, amely a felhasználók közötti kapcsolatokat (követések) kezeli.

5. Genre Modell

- A *genres* tábla reprezentációja, amely a zenei műfajokat kezeli.

6. Rating Modell

- A *ratings* tábla reprezentációja, amely az albumokra adott értékeléseket kezeli.

7. Review Modell

- A *reviews* tábla reprezentációja, amely a felhasználók által adott szöveges értékeléseket kezeli.

8. SimilarUser Modell

- A *similar_users* pivot tábla reprezentációja, amely a felhasználók közötti hasonlósági pontszámot tárolja.

9. Like Modell

- A *likeables* tábla reprezentációja, amely az alkalmazás különböző elemeinek like-okat kezeli.
- Ez egy különlegesebb tábla mivel ez a struktúra lehetővé teszi, hogy a kedvelések (likeok) polimorf módon kapcsolódjanak különböző modellekhez, például szöveges véleményekhez, listákhoz stb. (32. ábra) (33. ábra)

Mező neve	Adattípus	Nullable	Korlátozások
id	INTEGER	NOT NULL	PRIMARY KEY
user_id	INTEGER	NOT NULL	FOREIGN KEY
likeable_id	INTEGER	NOT NULL	FOREIGN KEY
likeable_type	VARCHAR	NOT NULL	
created_at	TIMESTAMP	NULL	
updated_at	TIMESTAMP	NULL	

32. ábra: *likeables* adatbázistábla mező szerkezeti leírása

id	user_id	likeable_id	likeable_type
1	5	101	App\Models\Review
2	3	202	App\Models\Review
3	8	101	App\Models\CustomList

33. ábra: példa adatokat tartalmazó lehetséges likeables tábla

Ennek a polimorf megoldásnak köszönhetően nincs szükség külön táblára az egyes modellekhez tartozó kedvelések tárolásához, elég egyetlen tábla.[18]

10. UserRecommendation Modell

- A *user_recommendations* tábla reprezentációja, amely a felhasználók számára ajánlott albumokat tárolja.

11. SocialLogin Modell

- A *social_logins* tábla reprezentációja, amely a felhasználók külső szolgáltatásokkal való bejelentkezését kezeli.

12. Artist Modell

- Az *artists* tábla reprezentációja, amely az előadók adatait tárolja.

3.6.3. Wilson Score Interval – Népszerű Albumok

A **Wilson score interval** egy statisztikai módszer, amelyet gyakran használnak arányok becslésére egy adott bizonytalansági szint mellett.[19] (33. ábra)

$$\left(\hat{p} + \frac{z_{\alpha/2}^2}{2n} \pm z_{\alpha/2} \sqrt{[\hat{p}(1 - \hat{p}) + z_{\alpha/2}^2/4n]/n} \right) / (1 + z_{\alpha/2}^2/n).$$

34. ábra: A Wilson score interval képlete [20]

Az **Album** modell tartalmazza a *getPopularityScore()* és *calculateWilsonScore()* függvényeket, amelyek lehetővé teszik az albumok népszerűségének kiszámítását az elmúlt 7 nap során kapott értékelések alapján. Ezek a függvények az adott albumra érkezett értékeléseket

súlyozzák, majd a népszerűségi pontszámot a **Wilson score intervallum** segítségével számítják ki.

A kiszámított népszerűségi pontszám lehetővé teszi az albumok rendezését népszerűség szerint (pl. legnépszerűbb albumok az elmúlt napokban). (35. ábra)

```
$popularAlbums = Album::withCount('reviews')
->get()
->sortByDesc(function ($album) {
    return [
        $album->getPopularityScore(),
        $album->reviews()->count(),
        $album->ratings()->avg('rating_value'),
        $album->ratings()->count()
    ];
})
->take(10);
```

35. ábra: példa kód az alkalmazásból a *getPopularityScore()* metódus használatára (népszerű albumok)

A népszerűség kiszámítására több módszer is létezik, például egyszerű átlagolás az értékelések alapján vagy az összes kapott pozitív érétkelés száma szerinti sorrendbe állítás. Ezek a megközelítések azonban nem veszik figyelembe a pozitív és negatív értékelések súlyozását továbbá az egyszerű átlagolásnál előrébb kerülhetne a népszerűségi rangsorban olyan album amire csak egy darab 5 csillagos értékelés érkezett (1 pozitív értékelés) olyan albummal szemben, aminek az átlag érékelése 4,5 35 darab értékelésből (amiből mondjuk 30 pozitív értékelés van és csak 5 negatív).**Hiba! A hivatkozási forrás nem található.**

Ezekre a problémákra nyújt megoldást a Wilson score alkalmazása, mert ez egy olyan statisztikai mérőszám, amely figyelembe veszi az értékelések számát és azok eloszlását.**Hiba!**
A hivatkozási forrás nem található.

A *getPopularityScore()* metódus működése:

- **Szűrés az elmúlt 7 napra:** csak azokat az értékeléseket veszi figyelembe, amelyeket az utolsó 7 napban hoztak létre
- **Értékelések számának csoportosítása és összegzése:** Kiszámolja, hogy hány értékelés tartozik az egyes minősítési szintekhez (1-től 5-ig terjedő értékelési skálán). (36. ábra)

```

$oneStar = $recentRatings->where( key: 'rating_value', operator: 1)->count();
$twoStar = $recentRatings->where( key: 'rating_value', operator: 2)->count();
$threeStar = $recentRatings->where( key: 'rating_value', operator: 3)->count();
$fourStar = $recentRatings->where( key: 'rating_value', operator: 4)->count();
$fiveStar = $recentRatings->where( key: 'rating_value', operator: 5)->count();

```

36. ábra: Az értékelések csoportosítása és összegzése példa kód

- Pozitív és negatív értékelések összesítése és súlyozása:** Súlyozza az egyes értékelési szintekhez. A magasabb értékelési szintek pozitívabb hatással bírnak, míg az alacsonyabbak negatív hatást gyakorolnak. Ha nem egy 1-5 terjedő skálán mozognának az értékelések, hanem csak pozitív (like) és negatív (dislike) értékelések lennének akkor elég lenne csak összeszámolni a pozitív és negatív értékelésket. Így viszont szükséges a súlyozás ($\$positive$ -nál nagyobb súllyal számáljuk a magasabb értékeléseket a $\$negative$ -nál viszont a kisebb értékelések kapnak nagyobb súlyt) [21] (37.ábra)

```

$positive = $twoStar * 0.25 + $threeStar * 0.5 + $fourStar * 0.75 + $fiveStar;
$negative = $oneStar + $twoStar * 0.75 + $threeStar * 0.5 + $fourStar * 0.25;

```

37. ábra: $\$positive$ $\$negative$ kiszámolása, kód részlet

- Wilson score kiszámítása:** Végső lépésként a *calculateWilsonScore()* metódust hívja meg a népszerűségi pontszám kiszámításához. Ez a függvény számolja ki a Wilson score-t, amely a népszerűségi pontszámot adja vissza. Ha nincs értékelés, az eredmény 0 lesz.

A *CalculateWilsonScore()* metódus. [21] (38. ábra)

```

function calculateWilsonScore($positive, $negative): float
{
    return ((($positive + 1.9208) / ($positive + $negative) - 1.96 * sqrt( num: (
        ($positive * $negative) / ($positive + $negative)) + 0.9604) / ($positive +
        $negative)) / (1 + 3.8416 / ($positive + $negative)));
}

```

38. ábra: A *calculateWilsonScore()* metódus

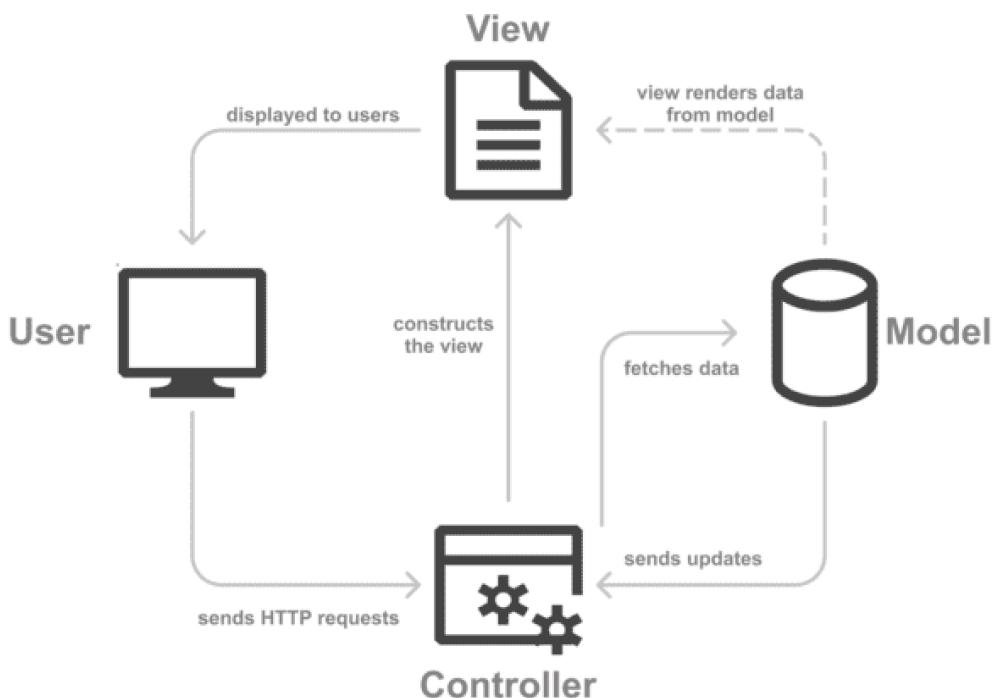
Magyarázat:

- $\$positive$ és $\$negative$:** A pozitív és negatív súlyozott értékelések összesítése.

- **1.9208 és 1.96:** Ezek a konfidenciaszint (95%) paraméterei. A Wilson score ezt a bizonytalanságot veszi figyelembe.
- ***qrt(...)*:** A négyzetgyök a valószínűségek eloszlásának szórását jelöli.
- Az képlet biztosítja, hogy az eredmény statisztikailag korrekt legyen még kis értékelésszám esetén is.

A két metódus megvalósításhoz az alábbi két hivatkozott oldalt vettem alapul: [20] [21]

3.7. Kontrollerek (Controllers) és Nézetek (Views)



39. ábra: A Controller osztályok szerepének szemléltetése[27]

A Laravel controllerek a webalkalmazás **irányítói**. Ők kezelik a beérkező HTTP-kéréseket, és felelősek azért, hogy az alkalmazás megfelelő logikát futtasson, majd a válaszokat továbbítsák a frontend (Blade nézetek) felé (39. ábra).[22]

A kontrollerek felelősek az adatokkal kapcsolatos CRUD (create, read, update, delete) műveletek kezeléséért. Egy kontrollerben általában külön metódusokat definiálunk minden a négy CRUD művelethez, és ezek a metódusok az útvonalak (routes) segítségével érhetők el a felhasználói felületről.[22]

A kontrollerek a **MusicApp / App / Http / Controllers** mappában találhatók.

A Blade nézet fájlok a ***Musicapp / resources / views*** mappában találhatóak almappákba rendezve (pl. users, albums stb.).

3.7.1. Egy Kontroller Működésének szemléltetése – AlbumController

A következőkben szemléltetem hogyan működnek a kontrollerek az *AlbumController* osztályon keresztül.

Osztály áttekintése

Az *AlbumController* több műveletet kezel:

- Albumok listázása (index).
- Egy adott album megjelenítése (show).
- További CRUD metódusok, amelyek jelenleg üresen állnak (create, update, destroy).

Az *index()* metódus – Új albumok, ajánlások és népszerű albumok megjelenítése

Ez a metódus az albumok főoldaláért felel.

Folyamat:

1. Felhasználó-azonosítás és ajánlások lekérése:

- Ha a felhasználó be van jelentkezve, a controller meghívja a *RecommendationController* metódusát.
- Az ajánlásokat a *user->recommendations* kapcsolaton keresztül tölti be.
- Ez lehetővé teszi, hogy az ajánlásokat a nézetben megjelenhessenek.

2. Új megjelenések lekérése a Spotify API-tól:

- A *getNewReleases* metódus a Spotify API-tól kérdezi le az új albumokat.
- Hiba esetén átirányítja a felhasználót egy "oops" hibakezelő oldalra.

3. Népszerű albumok kiválasztása:

- A rendszer az adatbázisból lekérdezi az albumokat és több szempont alapján rendezzi őket:

- Népszerűségi pontszám (*getPopularityScore*).
- Vélemények száma.
- Átlagos értékelés.
- Értékelések száma.

4. Adatok továbbítása a nézetnek

- A *view()* metódussal az *albums.index* Blade sablonhoz továbbítja az adatokat megjelenítésre:(40. ábra)

```
return view( view: 'albums.index', [
    'newReleases' => $newReleases,
    'albumInformation' => $albumInformation,
    'albums' => Album::all(),
    'recommendations' => $recommendations,
    'popularAlbums' => $popularAlbums,
]);
```

40. ábra: példa kód az *AlbumController* osztályra

A *albums.index* blade nézetben ezután a következő módon tudjuk ábrázolni az adatokat:

```
<div class="mt-6 grid grid-cols-4 gap-8">
    @foreach($newReleases as $album)
        <div class="mt-8">
            <a href="{{route('albums.show', $album->id)}}>
                
            </a>
            <div class="dark:text-gray-300">
                <a href="{{route('albums.show', $album->id)}}" class="text-lg mt-2">{{{$albumInformation[$loop->index]->name}}}</a>
                <div class="flex items-center">
                    <span class="mr-2">{{{$album->rating}}}</span>
                    <span>{{\App\Models\Album::avg_rating($album->id)}}</span>
                    <span class="mx-2">|</span>
                    <span>{{{$album->release_date}}}</span>
                </div>
            </div>
        </div>
    @endforeach
</div>
```

41. ábra: *albums.index* blade nézet részlet

A blade kód részleten (41. ábra) az látható ahogy végig iterál a program a controller által átadott *newRelesaes* tömbbön és ábrázolja az abban tárolt adatokat.

A **show()** metódus – Egy album részleteinek megjelenítése

Ez a metódus az egyedi album megjelenítéséért felelős.

Folyamat:

1. Album és előadó lekérése a Spotify API-ból:

- Az album adatait a Spotify API segítségével kérdezi le (*getAlbum*).
- Hiba esetén az "oops" oldalra irányít.

2. Adatbázis-album ellenőrzése vagy mentése:

- Ellenőrzi, hogy az album már létezik-e az adatbázisban.
- Ha nem, létrehozza az albumot, és társítja a műfajokat (*genres*).

3. Adatok továbbítása a nézetbe:

- A *view()* metódussal átadja az adatokat a *albums.show* Blade sablonnak

Privát **paginate()** metódus

Ez a metódus egy tömböt manuálisan oszt fel oldalakra.

Folyamat:

- Gyűjteményt alakítja a tömböt (*collect()*).
- A Laravel *LengthAwarePaginator* segítségével oldalakra bontja a tartalmat.
- Az *index()* metódusban például az új megjelenések oldalakra bontásához használja.

Összegzés

- Az **index** metódus többféle adatot (új megjelenések, népszerű albumok, ajánlások) lekérdez, rendez, és átadja a nézetnek.
- A **show** metódus egy konkrét album részleteit jeleníti meg, beleértve az adatbázis és az API adatait.

- Az adatok feldolgozása és megjelenítése közötti híd a Blade sablonok használatával valósul meg.

3.8. API Integrációk kontrollereken keresztül

Az alkalmazás két külső API-val kommunikál kontroller osztályokon keresztül:

- **Spotify Web API:** A Spotify zenei streaming szolgáltató cég által létrehozott API, amely lehetővé teszi a zenei adatok (albumok, előadók, műfajok) lekérdezését.
- A saját fejlesztésű, korábban taglalt **FastAPI-alapú ajánlórendszer:** az egyedi felhasználói preferenciák alapján ajánl albumokat, illetve azonosít hasonló érdeklődésű felhasználókat.

3.8.1. Spotify Web API

A Spotify Web API szolgáltatja az alkalmazás számára az album és előadó információkat. Ezáltal az egyik legfontosabb részét képezi az alkalmazásnak. Emiatt különösen fontos volt egy megbízható kapcsolat kiépíteni a két rendszer között.

Az integrációhoz a [Spotify Web API PHP](#) [23] csomagot használom, amely jelentősen megkönnyíti az autentikációs folyamatot és a kommunikációt a Spotify API-val.

A webalkalmazás és a Spotify API közötti kommunikáció logikája a szülő *Controller* osztályban került implementálásra, így származtatás révén az összes *Controller* osztály tudja használni azt a Spotify API-val való kommunikációról.

A segéd csomag függvényeinek használatával a következőképpen valósul meg a kommunikáció a Spotify API-val:

Változók deklarálása

- **\$api:** A Spotify API-val való kommunikációra szolgáló objektum.
- **\$spotifyApiSession:** A Spotify API hitelesítési munkamenet-kezeléséhez szükséges objektum.

Controller osztály Konstruktora: kapcsolat a Spotify API-val

- **spotifyApiSession():** Új hitelesítési munkamenetet hoz létre a Spotify API kliensazonosítójával és kliens titkos kulcsával, amelyeket az .env fájlból olvas ki.

- ***SpotifyWebAPI()***: Létrehozza az api objektumot, amelyen keresztül majd a Spotify végpontjait lehet majd használni. (42. ábra)

```
public function __construct() {
    $this->spotifyApiSession = new SpotifyWebAPI\Session(
        env('key: SPOTIFY_CLIENT_ID'),
        env('key: SPOTIFY_CLIENT_SECRET')
    );
    $this->api = new SpotifyWebAPI\SpotifyWebAPI();

    try {
        $this->handleAccessToken();
    } catch (Exception $e) {
        Log::error('Spotify API access token error: ' . $e->getMessage());
        return redirect()->route('oops');
    }
}
```

42. ábra: A Controller osztály konstruktora

Hozzáférési token kezelése

- A *handleAccessToken()* metódus biztosítja, hogy az alkalmazás minden érvényes hozzáférési tokennel rendelkezzen.
- Ha hiba történik (például hitelesítési hiba), az elkapott kivételt naplózza, és az *oops* hibakezelő oldalra irányít.

handleAccessToken() metódus

Ez a metódus az aktuális hozzáférési tokent kezeli:

1. **Ha nincs token a munkamenetben (Session):** Kér egy új tokent a Spotify API-tól.
2. **Ha a token hamarosan lejár:** Új tokent kér.
3. **Ha van érvényes token:** Beállítja azt az *api*-objektumban a kommunikációhoz.

Új hozzáférési token kérése

- A Spotify API kliens-hitelesítési tokent kér a *SpotifyApiSession* objektummal.
- Az új tokent eltárolja a munkamenetben, és beállítja az *api* objektumnak.

Hibakezelés

Ha bármelyik lépés során kivétel keletkezik:

- A hibát a Laravel naplófájljába menti (MusicApp / storage / logs mappa).
- Az *oops* nevű hibakezelő oldalra irányítja a felhasználót.

A többi Controller osztály ezt követően a következő metódusokat használhatja API hívásokra az *api* objektum segítségével:

- *api->getArtist(\$id)*: A megfelelő id birtokában lekérhető bármely előadóról információ (név, műfajok, kép).
- *api->getAlbum(\$id)*: A megfelelő id birtokában lekérhető bármely album információ (cím, előadó, megjelenési dátum, dallista stb.).
- *api->getNewReleases()*: Ennek segítségével lekérhetők az új mainstream album megjelenések, minden szükséges információval.
- *api->search(\$search, \$type)*: Ennek segítségével valósítottam meg a kereső mezőt. Az endpoint úgy működik, hogy meg kell adni egy string-et és a keresés típusát (pl. album) a string bemenet alapján az API visszaadja a lehetséges keresési találatokat.

3.8.2. Kommunikáció az Ajánlórendszerrel

A személyre szabott ajánló rendszerrel való kommunikáció a következőképpen zajlik a *RecommendationController* osztályon keresztül:

***getRecommendations* függvény:** Ez a fő függvény, amely kezeli az albumajánlásokat.

Adatok összegyűjtése

- **Minden felhasználó adatainak lekérése:** Az *getAllUsersData* metódus meghívásával minden felhasználó által adott értékelést lekérek, hogy ezek felhasználhatóak legyenek az ajánlórendszerben.
- **Jelenlegi felhasználó kiválasztása:** A *User::find(\$userId)* lekéri az aktuális bejelentkezett felhasználót, akihez az ajánlások készülnek.
- **Követett felhasználók listája:** A *followedFriends* kapcsolat segítségével lekérem azoknak a felhasználóknak az azonosítóit, akiket az aktuális felhasználó követ.

- **Értékelt, de nem osztályozott albumok:** A *reviews* kapcsolatot használom, hogy kiderítsem, mely albumokat véleményezett, de nem értékelt a felhasználó.
- **Discovery queue albumok:** A *discoveries* kapcsolatból kiderül, hogy mely albumok vannak a felhasználó felfedezési listáján.

Kérés az ajánlórendszerhez

HTTP kérés összeállítása:

Egy POST kérés küldése a Python alapú FastAPI ajánlórendszerhez az alábbi adatokkal:

- Felhasználói azonosító
- minden felhasználó adatai (*all_users_data*)
- követett felhasználók listája
- véleményezett albumok
- Discovery queue albumok

Válasz feldolgozása:

Ha a válasz sikeres:

- **Ajánlott albumok és hasonló felhasználók** listájának kinyerése.
- Az adatokat adatbázisba mentem az alábbi két metódus segítségével:
 - *storeRecommendations*: Ajánlott albumok mentése.
 - *storeSimilarUsers*: Hasonló felhasználók mentése.

Hibakezelés:

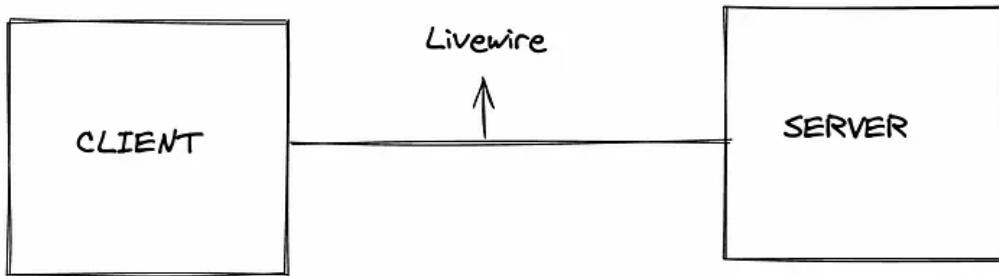
- Ha a kérés nem sikeres, vagy a FastAPI nem elérhető, az oldal jelzi a felhasználónak egy üzenet formájában, hogy jelenleg nem érhetők el ajánlások a számára.

3.9. Livewire Komponensek

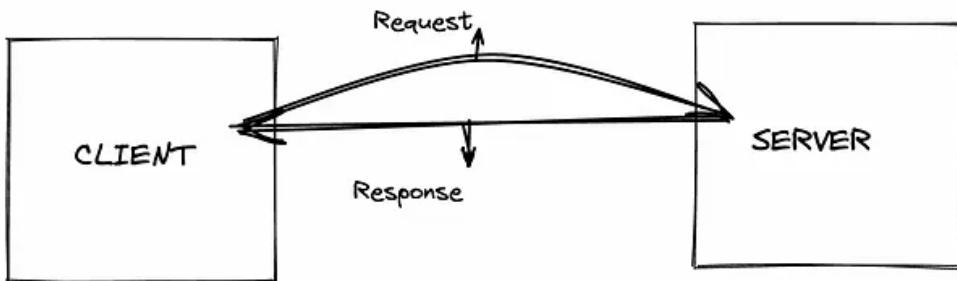
A Livewire komponensek fontos részét képezik az alkalmazásnak, egyfelől ezek biztosítanak interaktív funkciókat, továbbá a Livewire komponensekben is menthetünk el új adat objektumokat vagy akár módosíthatunk és törölhetünk is. Azt is mondhatjuk, hogy a Livewire komponensek interaktivitást biztosító controller osztályok.

A Livewire komponens egy PHP osztály, amely tartalmazza a szükséges logikát, és egy kapcsolódó Blade sablont, amely a HTML-t és az interaktív elemeket tartalmazza. Ezt a Blade sablont bármely nézetbe beágyszabhatjuk. A Livewire komponens osztálya tartalmazza az adatokat és az eseményeket, amelyek a felhasználói interakciókra reagálnak. A Livewire az eredeti komponens kimenetét az oldal betöltésekor rendereli. Amikor egy interakció történik, a Livewire AJAX-kérést küld a szervernek a frissített adatokkal. A szerver újrarendereli a komponenst, majd az új HTML-lel válaszol.[25]

Lényegében a Livewire egy "híd", amely összeköti a Klienst és a Szervert.(43. és 44. ábra)



43. ábra: Livewire elméleti működésének szemléltetése[26]



44. ábra: Livewire elméleti működésének szemléltetése[26]

A Livewire komponens osztályok a **MusicApp / app / Livewire** mappában találhatóak.

A Livewire komponensekhez tartozó Blade nézetek a **MusicApp / resources / views / livewire** mappában találhatóak.

3.10. Fontosabb Livewire Komponensek és funkciionalitások

3.10.1. SearchDropDown

A *SearchDropdown* Livewire komponens célja egy dinamikus keresőmező biztosítása, amely lehetővé teszi a felhasználók számára, hogy különböző kategóriákban (albumok, felhasználók, előadók) keressenek az alkalmazásban. A keresési eredmények valós időben frissülnek, és a keresési kategória a kereső mező melletti. opciók közül választható ki.

Változók:

- **public \$search:** A keresési mezőbe beírt szöveg, amely alapján a keresés végrehajtásra kerül.
- **public \$option:** A felhasználó által kiválasztott keresési opción (Albums, Users, Artists). Az alapértelmezett érték: Albums.

Metódusok

1. *apiSearch(\$type)*

Feladat: Kapcsolódás a Spotify API-hoz, hogy albumokat vagy előadókat keressen a megadott keresési kifejezés alapján.

Főbb funkciók:

- Ellenőrzi, hogy a Spotify API elérhető és van-e érvényes hozzáférési token:
 - Ha nincs token vagy az lejárt, új tokent kér a Spotify API-ból.
 - Az új tokent elmenti a session-be a későbbi használathoz.
- Ha a keresési szöveg hossza meghaladja a 3 karaktert, akkor a metódus elküldi a kérést az API-nak a keresési szöveggel és a keresési opciónnal (albums, artists).
- Hibakezelést végez: bármilyen API-hiba esetén az alkalmazás egy *Oops* hibaoldalra irányítja a felhasználót.

2. *userSearch()*

Feladat: Felhasználók keresése az adatbázisban a megadott keresési kifejezés alapján.

Főbb funkciók:

- A keresési kifejezést szóközök mentén bontja fel, és minden szóra külön feltételt ad az SQL lekérdezéshez.
- Az SQL lekérdezés kis- és nagybetű-független (*LOWER* függvény).
- Legfeljebb 10 találatot ad vissza.

3. *selectOption(\$selectedOption)*

Feladat: Beállítja a felhasználó által kiválasztott keresési kategóriát (opción).

Főbb funkciók:

- A *selectedOption* paraméter alapján frissíti az *option* tulajdonságot, amely meghatározza, hogy az alkalmazás melyik kategóriában hajt végre keresést (Albums, Users, Artists).

4. *render()*

Feladat: A Livewire komponens nézetének megjelenítéséért felelős metódus, amely visszaadja a keresési eredményeket a felhasználó által kiválasztott kategóriának megfelelően.

Főbb funkciók:

- Ellenőrzi, hogy a kiválasztott opción melyik kategóriának felel meg:
 - **Albums:** A *apiSearch('album')* metódust hívja meg.
 - **Users:** A *userSearch()* metódust hívja meg.
 - **Artists:** A *apiSearch('artist')* metódust hívja meg.
- A keresési eredményeket továbbítja a nézetnek.

Nézet (*search-dropdown.blade.php*)

- A nézet tartalmazza a keresőmezőt és a keresési eredmények megjelenítését a kiválasztott kategória szerint.

- Dinamikusan frissíti az eredményeket a keresési kifejezés vagy a kiválasztott opció változásakor.
- A keresési eredményekre kattintva pedig átirányítja a felhasználót az alkamzás a megfelelő oldalra (user keresés esetén az adott user profil oldalára stb.).

3.10.2. EditList

Feladata: Az *EditList* komponens feladata, hogy lehetővé tegye a felhasználók számára a saját egyedi listáik (*CustomList*) szerkesztését az alkalmazásban. A komponens biztosítja a lista nevénk módosítását, albumok hozzáadását és eltávolítását a listából, valamint az albumok sorrendjének frissítését.

A felhasználónak lehetősége van az albumok sorrendjének módosítására egy úgy nevezett "drag-and-drop" módszerrel. Ez azt jelenti, hogy a felhasználó rányom egy albumra és azt "megragadva" tetszése szerint tudja húzni bárhol a listán belül, miután elengedte az egeret az album sorrend módosul. Ennek a funkciónak a megvalósításához a <https://github.com/livewire/sortable> plugin használtam. Ez a csomag biztosít olyan funkcionálitást, amellyel könnyedén implementálható a "drag-and-drop" funkció.

Nézet:

A nézet (*MusicApp / resources / views / livewire / edit-list.blade.php*) biztosítja a következő funkciókat:

Album Keresőmező

- Keresési input mező a Spotify albumok keresésére.
- Valós idejű keresési eredmények megjelenítése.

Kiválasztott Album Megjelenítése

- Ha az albumot kiválasztották, megjeleníti annak adatait (borító, cím, előadó).

Album Hozzáadása és Eltávolítása

- Hozzáadás gomb az albumok listához való hozzáadásához.
- Eltávolítás gomb a meglévő albumok törléséhez a listából.

Lista Sorrendjének Kezelése

- Drag-and-drop funkció a lista elemeinek sorrendjének frissítésére.

Visszajelzések

- Hibák és státuszüzenetek megjelenítése.

3.10.3. RatingComponent

Feladata: A *RatingComponent* Livewire komponens, lehetővé teszi a felhasználók számára, hogy albumokat értékeljenek (csillagokkal, 1-5-ig terjedő skálán), módosítsák meglévő értékeléseiket, vagy töröljék azokat. Emellett vizuális visszajelzést ad az egérmutató értékelés fölé húzásakor.

Funkcionalitás és logika:

- ***mount(\$album)*:** Inicializálja a komponenst az album adataival. Ha a felhasználó be van jelentkezve és már értékelte az albumot, beállítja az aktuális értéket az adatbázisból.
- ***setRating(\$val)*:** Beállítja a kiválasztott értékelést:
 - Ha a felhasználó nincs bejelentkezve, a bejelentkezési oldalra irányít.
 - Ha a felhasználó korábban nem értékelte az albumot, új értékelést hoz létre.
 - Ha a felhasználó már értékelte, frissíti az értékelést, feltéve hogy jogosult rá (*Gate::allows*).
- ***deleteRating()*:** Törli a meglévő értékelést, ha a felhasználó jogosult erre.
- ***setHoverRating(\$val)*:** Beállítja az aktuális hover értéket (csillagszám, amit az egérmutató jelez).
- ***resetHoverRating()*:** Visszaállítja a hover értéket alaphelyzetbe (nullára).

Nézet:

- A komponens a *livewire.rating-component* nézetet használja az értékelések megjelenítésére, beleértve az aktuális értékelési állapotot, az egérmutató interakciók

vizuális visszajelzéseit, valamint az értékelés hozzáadásának, módosításának vagy törlésének lehetőségét.

3.11. Autentikáció

Az autentikációhoz a Laravel "Breeze"[24] névre keresztelt kezdőcsomagját használtam, ez tartalmaz minden autentikációs logikát, illetve biztosít login és register oldalakat, így ezt külön nekem már nem kellett lefejleszteni.

3.12. Tesztelés

3.12.1. Ajánlórendszer Tesztelése

Az ajánlórendszernél a két végpont működését teszteltem:

- A gyökér végpont: /
- Az ajánlásokat kezelő végpont: /recommend

A teszteléshez írtam egy tesztfájlt: *test_recommendations.py*

Ebben a tesztfájlban a TestClient segítségével emulálom a FastAPI alkalmazást.

Tesztadatok:

- Felhasználói értékelések, követett felhasználók listája, korábban értékelt albumok, valamint a felfedezőlistában lévő albumok.
- Az elvárt kimenet a következőket tartalmazza:
 - Az ajánlott albumok nem tartalmazzák a már ismert albumokat.
 - A hasonló felhasználók listájában nem szerepelnek követett felhasználók.

Tesztelés részei:

- **Gyökér végpont tesztje (/):**
 - Ellenőri a válasz státuskódját és üzenetét.
- **Ajánlási végpont tesztje (/recommend):**
 - Ellenőri, hogy az ajánlott albumok megfelelnek a követelményeknek.

- Vizsgálja, hogy a hasonló felhasználók listája megfelelően szűrt.
- Ellenőri az elvárt ajánlásokat és hasonló felhasználókat.

Futási eredmények:

A kód, mind a két tesztesetre jól reagált, az elvárt módon működik. Az ajánlott albumok nem tartalmazzák a *reviewed_albums* vagy a *discovery_queue* listákban szereplő albumokat. A hasonló felhasználók nem tartalmazzák a követett felhasználókat (following). A várhatóan ajánlott albumok között szerepel a 106-os album id. A várhatóan hasonló felhasználók között szerepel a 3 és az 5-ös user id.

3.12.2. Webalkalmazás Tesztelése

A webalkalmazáshoz írtam számos unit és feature tesztet a **MusicApp / tests / Feature** illetve a **MusicApp / tests / Unit** mappákban, azonban ezek nem teljes körűek ezért az oldal működésének a tesztelését manuálisan végzem el és jegyzem le az alábbi teszt jegyzőkönyvben user story formájában (45. ábra):

Tevékenység	AS A... I WANT TO...	GIVEN... WHEN... THEN...
Regisztráció	AS A: vendég felhasználó	GIVEN: az alkalmazás fut
	I WANT TO: regisztrálni egy profillal az oldalon	WHEN: rákattintok a regisztráció gombra vagy az általam használt közösségi oldal vagy Google profil gombra
		THEN: a beviteli mezőket helyesen kitöltve létrejön egy felhasználói profil az adatbázisban, vagy ha az általam használt közösségi oldalon keresztül szeretnék regisztrálni akkor egy ahhoz tartozó token
		Tesztelés eredménye: OK

Bejelentkezés	AS A: vendég felhasználó	GIVEN: az alkalmazás fut
	I WANT TO: belépn az oldalra a jelszavammal és a felhasználó nevemmel, ha közösségioldal felhasználó fiókkal regisztráltam akkor azon keresztül	WHEN: rákattintok a belépés vagy az általam használt közösségi oldal vagy Google profil gombra.
		THEN: belépek az oldalra és igénybe tudom venni az oldal szolgáltatásait
		Tesztelés eredménye: OK
Ajánlások megtekintése	AS A: bejelentkezett felhasználó	GIVEN: autentikációt követően, fut az ajánló rendszer
	I WANT TO: megtekinteni az én személyre szabott ajánlásaimat	WHEN: belépek a főoldalra
		THEN: megjelennek azok az albumok, amiket a rendszer ajánl számomra a korábbi, appon belüli tevékenységet elemezve
		Tesztelés eredménye: OK

Újdonságok megtekintése	AS A: vendég vagy bejelentkezett felhasználó	GIVEN: az alkalmazás fut
	I WANT TO: megtekinteni az újdonságokat	WHEN: belépek a főoldalra
		THEN: a felső sávban megjelenő új albumokat lapozós galériaként megtekinthetem
		Tesztelés eredménye: OK
Népszerű albumok megtekintése	AS A: bejelentkezett felhasználó	GIVEN: autentikációt követően
	I WANT TO: megtekinteni a népszerű albumokat	WHEN: belépek a főoldalra
		THEN: megjelennek a népszerű albumok helyes sorrendben
		Tesztelés eredménye: OK
Követett felhasználók tevékenységének megtekintése	AS A: bejelentkezett felhasználó	GIVEN: autentikációt követően
	I WANT TO: Megtekinteni az általam követett emberek tevékenységét	WHEN: belépek a saját profil oldalamra vagy a kereső mezőben rákeresek egy másik felhasználó profiljára

	(Milyen albumokat értékeltek)	
		THEN: meg tudom tekinteni az általam követett felhasználók tevékenységét
		Tesztelés eredménye: OK
Böngészés	AS A: vendég vagy bejelentkezett felhasználó	GIVEN: az alkalmazás fut
	I WANT TO: Böngészni a szöveg alapú keresőn keresztül, albumok, felhasználók vagy előadók szerint	WHEN: fejlécbe épített kereső mezőre kattintok
		Tesztelés eredménye: OK
Album adatlapjának megnyitása	AS A: vendég vagy bejelentkezett felhasználó	GIVEN: az alkalmazás fut
	I WANT TO: megtekinteni egy album adatlapját	WHEN: Belépek a főoldalra, vagy, amikor rákeresek egy adott albumra
		THEN: Megnyílik az adott album adatlapja
		Tesztelés eredménye: OK

Előadó adatlapjának megnyitása	AS A: vendég vagy bejelentkezett felhasználó	GIVEN: az alkalmazás fut
	I WANT TO: megtekinteni egy előadó adatlapját	WHEN: rákeresek egy adott előadóra
		THEN: Megnyílik az adott előadó adatlapja
		Tesztelés eredménye: OK
Album értékelése	AS A: bejelentkezett felhasználó	GIVEN: autentikációt követően egy album adatlapján
	I WANT TO: Értékelni egy albumot, szövegesen és/vagy csillagozással	WHEN: Megnyitottam egy album adatlapját
		THEN: az album értékelése eltárolódik az adatbázisban ezt követően a szöveges értékelés minden felhasználó számára látható lesz, a csillagozott értékelés pedig beleszámít az adott album átlagolt értékelésébe, az ajánló rendszer pedig megjegyzi, hogy milyen típusú albumot értékeltünk és mennyire pozitívan
		Tesztelés eredménye: OK
Album kívánságlistára tétele	AS A: bejelentkezett felhasználó	GIVEN: autentikációt követően egy album adatlapján

	I WANT TO: Kívánságlistára tenni egy albumot, hogy emlékezzek rá később, hogy majd egyszer szeretném meghallgatni	WHEN: Megnyitottam egy album adatlapját
		THEN: rákattintok a Discovere Quee gombra majd elmenti az albumot a rendszer, és bármikor kikereshető lesz a kívánságlistából
		Tesztelés eredménye: OK
Egy felhasználó bekövetése	AS A: bejelentkezett felhasználó	GIVEN: autentikációt követően
	I WANT TO: követni egy másik felhasználó tevékenységét	WHEN: Rákerestem egy felhasználóra, vagy valamelyik oldalon, pl. egy album adatlapja, ahol láthatók más felhasználók értékelései, rákattintok egy profilra
		THEN: innentől fogva látom annak a felhasználónak a tevékenységét, akit bekövettem.
		Tesztelés eredménye: OK
Toplisták megtekintése	AS A: felhasználó	GIVEN: az alkalmazás fut
	I WANT TO: megtekinteni a toplistákat,	WHEN: kiválasztom a toplisták menüpontot

	különböző szempontok alapján	
		THEN: meg tudom nézni a toplistákat. és rákeresni, hogy van e számomra érdekes lista
		Tesztelelés eredménye: OK
Saját toplisták létrehozása/generálása általam megadott szempontok alapján	AS A: bejelentkezett felhasználó	GIVEN: autentikációt követően
	I WANT TO: létrehozni saját toplistákat	WHEN: kiválasztom a toplisták menüpontot
		THEN: az általam megadott feltételek alapján létrejön egy lista
		Tesztelelés eredménye: OK
Saját statisztikáim megtekintése	AS A: bejelentkezett felhasználó	GIVEN: autentikációt követően
	I WANT TO: Megtekinteni a rám vonatkozó statisztikákat (általam legtöbbször hallgatott műfajok stb.)	WHEN: rákattintok a profilomra
		THEN: megjelennek a statisztikák

		Tesztelés eredménye: OK
Toplisták / kommentek (review) likeolása	AS A: bejelentkezett felhasználó	GIVEN: autentikációt követően
	I WANT TO: a szív szimbólumra kattintva kedvelni (likeolni) az adott toplistát / kommentet	WHEN: Az album adatlapján / tolista oldalon vagyok
		THEN: kedvelés-t elmenti az adatbázisba és megjelenik, hogy likeoltam az adott commentet / toplistát
		Tesztelés eredménye: OK

45. ábra: Manuális tesztelés táblázata

4. Összefoglalás

Szakdolgozatom keretében sikerült megvalósítanom egy olyan webalkalmazást, ahol a felhasználók kedvük szerint értékelhetnek albumokat. Böngészhetnek új albumok és előadók iránt kutatva. Saját toplistákat készíthetnek és generálhatnak, amelyeket aztán más felhasználók megtekinthetnek, hogy új kedvenceket találhassanak. Ezeken felül pedig a webalkalmazás által személyre szabott ajánlásokat kapnak, így bővíve zenei tudásukat. Továbbá a követési ajánlások révén megismerhetnek új érdekes felhasználókat is.

A program készítése során lehetőségem volt sok új érdekes és hasznos tapasztalatra szert tenni. Megtapasztalhattam, hogy mekkora munka egy ilyen alkalmazás felépítése. Az elsőre egyszerűnek tűnő problémák mögött meghúzódó komplex megoldási stratégiák feltérképezése során is sok újat tanultam, például, hogyan érdemes értékelések alapján albumokat sorrendbe rendezni (Wilson Confidence Interval).

Lehetőségem volt jobban elmélyedni a Laravel webes keretrendszerben. Megtanulni olyan funkcióit, amiről korábban nem is tudtam. Megtanulhattam hogyan kell külső API-kat integrálni a rendszerbe.

Ezen felül pedig megtanultam a személyre szabott ajánló rendszerek felépítéséhez szükséges alap tudást, különböző megközelítéseket és a megvalósításhoz szükséges technológiákat. Ezen felül megtanulhattam hogyan és milyen technológiákkal lehet Python alapú microservice-t fejleszteni.

Sikeresen megvalósítottam az előre jelzett funkciókat. A program felépítésének köszönhetően az alkalmazás könnyedén bővíthető új funkciókkal.

5. További fejlesztési lehetőségek

Az ajánló rendszert lehetne bővíteni olyan szinten, hogy az egyes albumokhoz találjon hasonlókat. Ezt egy új végpont és egy új ajánlási logika (Content-Based Recommendation) implementálásával lehetne megcsinálni. A megvalósított FastAPI microservice alkalmas az ilyen fajta bővítésre.

A webalkalmazást lehetne bővíteni admin funkciókkal. Például a sértő és nem oda való szöveges vélemények jelentése és törlése lehetne egy ilyen admin funkció. Természetesen ezekhez létre kellene hozni egy admin felületet, továbbá adatbázis táblák szintjén is megvalósítani, hogy a felhasználók különböző rangokkal rendelkezzenek (admin felhasználó, sima felhasználó).

Egy hosszabb távú fejlesztés lenne, hogy az alkalmazás egyre kevésbé függjen a Spotify Web API által szolgáltatott adatoktól. Például lehetne, hogy több információ mentésre kerüljön az albumokról és az előadókról az alkalmazás saját adatbázisába. Továbbá lehetne egy olyan funkciója az alkalmazásnak, ahol az arra jogosult felhasználók regisztrálhatnának új albumokat, megadva a szükséges információkat (cím, album borító stb.). Erre azért is lenne szükség mert bár a Spotify hatalmas zenei könyvtárral rendelkezik, korántsem tartalmaz minden albumot.

A saját toplista készítő oldalt lehetne olyan funkciókkal bővíteni, például, hogy a felhasználó írhasson különböző megjegyzéseket a listán szereplő albumokhoz, hogy például egy adott albumról mik a személyes benyomásai, miért így rangsorolta... stb. További fejlesztési lehetőség lenne, hogy ne csak 20 albumot adhasson hozzá a felhasználó a listához, hanem többet, ilyenkor pedig kiválaszthatná, hogy milyen elrendezésben jelenjenek meg az albumok (pl. 1 oldalon 10 album vagy 1 oldalon csak 5 album).

6. Ábrajegyzék

1. ábra: Site map az alkalmazás oldalainak kapcsolatáról	7
2. ábra: Site map az Album és Generate a list oldalakról.....	7
3. ábra: regisztrációs panel	8
4. ábra: az alkalmazás fejléce, bejelentkezett felhasználó esetén.....	9
5. ábra: Profil menü	9
6. ábra: Új megjelenések	10
7. ábra: személyre szabott ajánlások	10
8. ábra: népszerű albumok	11
9. ábra: Album bemutató oldal példa	12
10. ábra: az előadó oldal.....	13
11. ábra: a felhasználó aktivitása az adott előadónál	13
12. ábra: az előadó diszkográfiája.....	14
13. ábra: Lista menü	14
14. ábra: példa toplista	15
15. ábra: lista kreatívítés	15
16. ábra: albumok listája a módosító oldalon.....	16
17. ábra: generált lista példa	17
18. ábra: felhasználó legutóbbi 5 aktivitása.....	18
19. ábra: követési ajánlások.....	19
20. ábra: a felhasználó statisztikái	19
21. ábra: cosine similarity[4].....	22
22. ábra: request body példa	23
23. ábra: példa pivot táblázat szemléltetésnek	24
24. ábra: kód részlet a cosine_similarity függvény alkalmazásáról.....	25
25. ábra: MVC architektúra ábrázolása [8]	29
26. ábra: példa adatbázis schema (users).....	30
27. ábra: model fájl példa (Artist model)	31
28. ábra: az adatbázis modellek és kapcsolataik diagramma	32
29. ábra: users adatbázistábla mező szerkezeti leírása	33
30. ábra: albums adatbázistábla mező szerkezeti leírása	34

31. ábra: likeables adatbázistábla mező szerkezeti leírása	35
32. ábra: példa adatokat tartalmazó lehetséges likeables tábla.....	36
33. ábra: A Wilson score interval képlete [20].....	36
34. ábra: példa kód az alkalmazásból a getPopularityScore() metódus használatára (népszerű albumok).....	37
35. ábra: Az értékelések csoportosítása és összegzése példa kód.....	38
36. ábra: \$positive \$negative kiszámolása, kód részlet	38
37. ábra: A calculateWilsonScore() metódus	38
38. ábra: A Controller osztályok szerepének szemléltetése[27]	39
39. ábra: példa kód az AlbumController osztályra	41
40. ábra: albums.index blade nézet részlet	41
41. ábra: A Controller osztály konstruktora	44
42. ábra: Livewire elméleti működésének szemléltetése[26].....	47
43. ábra: Livewire elméleti működésének szemléltetése[26].....	47
44. ábra: Manuális tesztelés táblázata.....	60

7. Irodalomjegyzék

- [1] Python FastApi vs Flask: A Detailed Comparison <https://www.turing.com/kb/fastapi-vs-flask-a-detailed-comparison#pros-and-cons-of-fastapi-and-flask> (utolsó elérés: 2024. november. 30.)
- [2] Bearer Authentication ismertető
https://swagger.io/docs/specification/v3_0/authentication/bearer-authentication/
(utolsó elérés: 2024. november. 30.)
- [3] Recommendation System in Python <https://www.geeksforgeeks.org/recommendation-system-in-python/> (utolsó elérés: 2024. november. 30.)
- [4] Cosine Similarity simertető cikk <https://towardsdatascience.com/cosine-similarity-how-does-it-measure-the-similarity-maths-behind-and-usage-in-python-50ad30aad7db>
(utolsó elérés: 2024. november. 30.)
- [5] Cosine Similiralty ismertető <https://www.geeksforgeeks.org/cosine-similarity/> (utolsó elérés: 2024. november. 30.)
- [6] Is laravel full stack <https://acquaintsoft.com/answers/is-laravel-full-stack> (utolsó elérés: 2024. november. 30.)
- [7] Ruby on rails vs laravel <https://flatirons.com/blog/ruby-on-rails-vs-laravel-which-is-better-2024/> (utolsó elérés: 2024. november. 30.)
- [8] MVC ismertetése <https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/> (utolsó elérés: 2024. november. 30.)
- [9] Laravel dokumentáció: frontend <https://laravel.com/docs/11.x/frontend#php-and-blade> (utolsó elérés: 2024. november. 30.)
- [10] Livewire dokumentáció <https://livewire.laravel.com> (utolsó elérés: 2024. november. 30.)
- [11] Livewire vs más frontend rendzserek <https://varshaaweblabs.com/blog/leveraging-laravel-livewire-a-comparative-analysis-with-vuejs-reactjs-and-nextjs> (utolsó elérés: 2024. november. 30.)
- [12] TailwindCSS dokumentáció <https://tailwindcss.com> (utolsó elérés: 2024. november. 30.)

- [13] PostgreSQL ismertető <https://www.enterprisedb.com/postgres-tutorials/why-more-and-more-enterprises-are-choosing-postgresql-their-go-database> (utolsó elérés: 2024. november. 30.)
- [14] Hivatalos Doscker ismertető <https://docs.docker.com/get-started/docker-overview/> (utolsó elérés: 2024. november. 30.)
- [15] Datacamp blog poszt a PostgreSQL és MySQL összehasonlításáról
https://www.datacamp.com/blog/postgresql-vs-mysql?utm_source=google&utm_medium=paid_search&utm_campaignid=19589720824&utm_adgroupid=152984014494&utm_device=c&utm_keyword=&utm_matchtype=&utm_network=g&utm_adposition=&utm_creative=720362650519&utm_targetid=ds-a-2222697809758&utm_loc_interest_ms=&utm_loc_physical_ms=9062987&utm_content=DSA~blog~SQL&utm_campaign=230119_1-sea~dsa~tofu_2-b2c_3-row-p2_4-prc_5-na_6-na_7-le_8-pdsh-go_9-nb-e_10-na_11-na-bfcf24&gad_source=1&gbraid=0AAAAAADQ9WsFyFcvdoGRO0Yhopcuukcv3T&gclid=Cj0KCQiAo5u6BhDJARlsAAVoDWvFX6QVGs3174SLSeWgH-AgYYIXQ8DYDgbk0N404tcOnaZMVcgoEpoaAt2yEALw_wcB (utolsó elérés: 2024. november. 30.)
- [16] Laravel dokumentáció: migrations <https://laravel.com/docs/11.x/migrations> (utolsó elérés: 2024. november. 30.)
- [17] Laravel dokumentáció: eloquent <https://laravel.com/docs/11.x/eloquent> (utolsó elérés: 2024. november. 30.)
- [18] Polymorphic Relationships and how they work?
<https://medium.com/@hendelRamzy/what-is-polymorphic-relationships-and-how-they-work-61df8d1562c5> (utolsó elérés: 2024. november. 30.)
- [19] Binomial proportion confidence interval
https://en.wikipedia.org/wiki/Binomial_proportion_confidence_interval (utolsó elérés: 2024. november. 30.)
- [20] Evan Miller: how not to sort by average <https://www.evanmiller.org/how-not-to-sort-by-average-rating.html> (utolsó elérés: 2024. november. 30.)

- [21] Github kód Wilson score PHP implementáció
<https://gist.github.com/julienbourdeau/77eaca0fd1e4af3fde9fe018fdf13d7d> (utolsó elérés: 2024. november. 30.)
- [22] Laravel dokumentáció: Controllers <https://laravel.com/docs/11.x/controllers> (utolsó elérés: 2024. november. 30.)
- [23] Github: segéd csomag Spotify API-hoz <https://github.com/jwilsson/spotify-web-api-php> (utolsó elérés: 2024. november. 30.)
- [24] Laravel dokumentáció: Starter kits <https://laravel.com/docs/11.x/starter-kits> (utolsó elérés: 2024. november. 30.)
- [25] Livewire dokumentáció <https://livewire.laravel.com> (utolsó elérés: 2024. november. 30.)
- [26] Medium cikk a Livewire működéséről <https://medium.com/@developer.olly/an-overview-of-how-livewire-works-85395746d10a> (utolsó elérés: 2024. november. 30.)
- [27] Flatlogic cikk a Laravel működéséről <https://flatlogic.com/blog/what-is-laravel/> (utolsó elérés: 2024. november. 30.)