
Soproni Egyetem
Simonyi Károly Műszaki,
Faanyagtudományi és Művészeti Kar
Informatikai és Gazdasági Intézet

# „ArtistHUB” webapplikáció fejlesztése

**Vértesi Patrik**

BSc Szakdolgozat

Konzulens: Koncz Adrienn

Külső konzulens: Gludovátz Attila

2021.05.07.

*Soproni Egyetem*

**Simonyi Károly Műszaki,**
**Faanyagtudományi és Művészeti Kar**

*H-9401 Sopron, Bajcsy-Zs. u. 4. Pf.: 132.*

*Tel: +36 (99) 518-101 Fax: +36 (99) 518-259*

# NYILATKOZAT

Alulírott **Vértesi Patrik** (neptun kód: **JPS7FL)** jelen nyilatkozat aláírásával kijelentem, hogy a **„ArtistHUB" webapplikáció fejlesztése** című

**szakdolgozat**

(a továbbiakban: dolgozat) **önálló munkám**, a dolgozat készítése során betartottam a szerzői jogról szóló 1999. évi LXXVI. tv. szabályait, különösen a hivatkozások és idézések tekintetében.

*Hivatkozások és idézések szabályai:*

*Az 1999. évi LXXVI. tv. a szerzői jogról 34. § (1) és 36. § (1) első két mondata.*

Kijelentem továbbá, hogy a dolgozat készítése során az önálló munka kitétel tekintetében a konzulenst, illetve a feladatot kiadó oktatót **nem tévesztettem meg.**

Jelen nyilatkozat aláírásával tudomásul veszem, hogy amennyiben bizonyítható, hogy a dolgozatot nem magam készítettem, vagy a dolgozattal kapcsolatban szerzői jogsértés ténye merül fel, a Soproni Egyetem megtagadja a dolgozat befogadását és ellenem fegyelmi eljárást indíthat.

A dolgozat befogadásának megtagadása és a fegyelmi eljárás indítása nem érinti a szerzői jogsértés miatti egyéb (polgári jogi, szabálysértési jogi, büntetőjogi) jogkövetkezményeket.

2021.05.07.

……..

hallgató

Soproni Egyetem, Simonyi Károly Műszaki, Faanyagtudományi és Művészeti Kar

Informatikai és Gazdasági Intézet

9400 Sopron, Bajcsy-Zs. u. 4.

# SZAKDOLGOZAT FELADAT

| | |
|---|---|
| Szakdolgozatot készítő neve: | **Vértesi Patrik** gazdaságinformatikus BSc hallgató |
| A szakdolgozatot készítő Neptun kódja: | JPS7FL |
| Szakdolgozat címe: | **„ArtistHUB" webapplikáció fejlesztése** |
| Intézeti konzulens(ek): | **Koncz Adrienn,** tanársegéd |
| Külső konzulens: | Gludovátz Attila, GAIN BI Kft. |
| A dolgozat kódja | SKK-INGA-01-2021-SZ |

## Elvégzendő feladatok

1. Oauth belépés megvalósítása
2. Chat funkció létrehozása
3. Bejegyzés létrehozása, kedvélese, hozzászólások írásának megvalósítása
4. Zenelejátszó implementálása.
5. Új zenék ajánlása korábban kedvelt stílusok szerint
6. Aktív felhasználók jelzése
7. Követés,kikövetés, követők számlálása, követettek tartalmának megjelenítése
8. Modern és reszponzív frontend a legújabb UX technikák figyelembevételével
9. Mindezek megvalósítása PHP-alapú Laravel keretrendszerben

Beadási határidő: 2021. május 07. 12:00

Sopron, 2021.02.19.

**Prof. Dr. Magoss Endre**
dékán

**Dr. Hegyháti Máté**
intézetigazgató

P.H.

# Tartalom

## Absztrakt

Szakdolgozatom témájának egy olyan közösségi oldal fejlesztését választottam, amely a zenei előadók közös munkájának megkönnyítésére alkalmas portál, amin keresztül egyszerűen és könnyen tudnak a felhasználók zenéket megosztani és kapcsolatokat építeni. Zenei hobbimból adódóan volt már szerencsém más előadókkal együtt dolgozni, és mind a zenemegosztás, beszélgetés, projekt megosztás több erre a célra kifejlesztett weboldalon vagy alkalmazáson keresztül történt. Az én webapplikációm célja, hogy ne legyen szükség több felület használatára, így segítve más előadók közti beszélgetést, zenemegosztást, hírek megosztását, valamint a követőbázis kialakítását. Egy platform független web applikációt alkottam meg, amely segíti az előadók közös munkáját, egymás megismerését, azáltal, hogy a hagyományos zene/videó lejátszó és képnézegető lehetőség mellett zenei projektjeiket, hangokat oszthassanak meg egymással, és azonnali üzenetváltást lehetővé téve gyorsan és hatékonyan tudjanak haladni a munkával egy adott platformon belül. Így nem kell 2-3 felületen keresztül küldeni, fogadni a fájlokat, vagy érdeklődni a másik munkássága felől. Kezdésnek bemutatom a felhasznált technológiákat, valamint a projekt előéletét. Később ismertetem a weboldal reszponzív grafikai tervével, a mögöttes adatbázissal, valamint a funkciókkal, amelyeket a regisztrált és nem regisztrált felhasználók vehetnek igénybe. A lefejlesztett funkciók után pedig tartok egy betekintést a tesztelés menetéről, továbbá a további terveimről, amelyeket a projekt keretein belül még meg szeretnék valósítani.

## Absctract

For my thesis, I chose to develop a community site, a portal to facilitate the collaboration of music artists, where users can easily share music and build relationships. Due to my musical hobby, I have had the opportunity of working with other performers. Sadly, all the music sharing, discussion and project sharing has been done through several dedicated websites or applications. My web application aims to eliminate the need to use multiple platforms, thus facilitating discussion, music sharing, news sharing and building a follower base among other performers. I have created a platform independent web application that helps performers to work together, get to know each other, by allowing them to share music projects, sounds, and to move work quickly and efficiently within a platform, allowing instant messaging, in addition to the traditional music/video player and image viewer. No need to send and receive files across 2-3 interfaces or enquire about each other's work. At the start, I will describe the technologies used and the project history. Later, I will describe the responsive graphic design of the website, the underlying database and the features that registered and non-registered users can use. And after the features have been developed, I will give an insight into the testing process and the further plans I have for the project.

# 1. Bevezetés

Napjaink talán egyik legfontosabb technológiája az internet. Az emberiség nagy része nap mint nap használja, ezen találják a legújabb híreket, legfontosabb tudnivalókat és még számos tudást, amely a mai életben elengedhetetlen. Már egyesek számára elképzelhetetlenné vált, hogyha valamire kíváncsiak ne tudják azt a zsebükben lévő telefon keresőjébe beírni, és információhoz jutni másodpercek alatt. Ma a fejlődő országokban szinte minden háztartás rendelkezik internettel, akár személyenként több számítógéppel, vagy telefonnal, amely rendelkezik internet eléréssel.

Az interneten elérhető szolgáltatások nagy részét web böngésző segítségével érjük el. Régebben egy weboldal csak statikus tartalmakat lehetővé tevő HTML [1] elemekből épült fel. A kezdetben statikus leírásokat, dokumentációkat tartalmazó web mára feledésbe merült, felváltotta a dinamikus tartalmú, felhasználó központú weboldalak tömkelege. Az új „trend” lehetővé tette az olyan weboldalak megjelenését melyeket komplexitásuknak köszönhetően webes alkalmazásoknak nevezünk.

Az első közösségi oldalak, beszélgető platformok megoldották az emberek közötti lassú üzenetváltás problémáját, ezzel felgyorsítva a kommunikációt és az adatáramlást. Ez a ma is fejlődő ágazat nagyban meghatározza az emberek életet. Számos változata létezik manapság a közösségi oldalaknak, van, amely kifejezetten fényképek, videók megosztására van kiélezve, van pedig, amely inkább híreket, és a csoportos tevékenységeket helyezte előtérbe. Ma akár már egy kattintással beszélhetünk egy ismerősünkkel, aki az Egyesült Államokban él, vagy akár ismerhetünk meg új embereket a saját érdeklődési körünkből, esetleg dolgozhatunk egy projekten csapatban, a világ minden tájáról.

## 1.1. Szakdolgozat felépítése

Szakdolgozatom a következő fejezetekből épül fel.

**Bevezetés**

Ebben a fejezetben mutatom be a szakdolgozat motivációját és a téma alapjait

**Felhasznált technológiák**

Itt írok azokról az eszközökről, amelyeket felhasználtam a webalkalmazás fejlesztése során.

**Felhasználói felület**

Bemutatom a felhasználói felület tervét, valamint végül a kész alkalmazás kinézetét, tartalom nélkül.

**Adatbázis felépítése**

Részletezem a sémában felhasznált táblákat és a felhasználásuk módját.

**Séma megtervezése**

A korábban bemutatott táblákat összekapcsolva megkapjuk a sémát. Írok a kapcsolatok fajtájáról, valamint arról, hogy az esetemben hogyan kellett alkalmazni ezeket.

**Specifikáció és tervezés**

Az alkalmazástól elvárt funkciókról, és a megalkotásuknak a módjáról írok itt.

**Implementáció**

Ebben a fejezetben írok a funkciók megalkotásáról, valamint a kész funkció felhasználásának lehetőségeiről

**Tesztelés**

Fontos egy weboldal vagy alkalmazás fejlesztése során, hogy a végterméket a megfelelő módon teszteljük. Ebben a fejezetben foglalkozom a tesztelés menetéről, és az eredményekkel.

**Összefoglalás és további tervek, tovább lepési lehetőségek**

Miután kész az alkalmazás, lehetőség van még fejleszteni a funkcióit. Ezekről ebben a fejezetben fogok írni.

**Ábrajegyzék**

A képek hollétéről írok az ábrajegyzék fejezetben.

**Irodalomjegyzék**

A felhasznált források neve valamit elérhetősége található ebben a fejezetben.

## 1.2. Motiváció, dolgozat célja

Témámnak egy olyan közösségi oldal fejlesztését választottam, amely a zenei előadók közös munkájának megkönnyítése, kifejezetten zenével, és kapcsolatok kiépítésével foglalkozó portál. Zenei hobbimból adódóan volt már szerencsém más előadókkal együtt dolgozni, és mind a zenemegosztás, beszélgetés, projekt megosztás más-más erre kifejlesztett portálokon történt. Az én webapplikációm célja, hogy ne legyen szükség több felület használatára, így segítve más előadók közti beszélgetést, zenemegosztást, hírek megosztását, valamint a követőbázis kialakítását.

Egy platform független web applikációt alkottam meg, amely segíti az előadók közös munkáját, egymás megismerését, azáltal, hogy a hagyományos zene/videó lejátszó és képnézegető lehetőség mellett zenei projektjeiket, hangokat oszthassanak meg egymással, és azonnali üzenetváltást lehetővé téve gyorsan és hatékonyan tudjanak haladni a munkával, egy adott platformon belül. Így nem kell 2-3 felületen keresztül küldeni, fogadni a fájlokat, vagy érdeklődni a másik munkássága felől.

Számos oldal létezik már az előadók mindennapi problémáinak megoldására, munkájának segítésére, viszont egy olyan oldal sem létezik, ami ezeket a hasznos funkciókat átfogóan, egy adott alkalmazásba foglalná, és ebből adódóan nem kellene több alkalmazást egyszerre használni.

# 2. Felhasznált technológiák

A webalkalmazásom elkészítése során igyekeztem minél jobban szem előtt tartani legmodernebb technológiák felhasználását, így mind kliens oldali, szerver oldali, valamint adatbázis szempontjából a következőket választottam.

## 2.1. HTML

A HTML (HyperText Markup Language) a legalapvetőbb építőeleme minden weboldalnak. Definiálja a tartalmat, és az elemek struktúráját a weben. Fel lehet úgy fogni, mint egy újság tartalmi felépítését, ahol külön ki van emelve a cím, a tartalom, alcímek, valamint az ahhoz tartozó egyéb elemek. A „Hypertext” alatt pedig azokat a linkeket értjük, amelyek összekötik a weboldalakat, így kialakítva a világhálót.

A HTML „markup” -ot használ, hogy a felhasználó számára is látható, olvasható tartalmat biztosítson a böngészőben. Ezek speciális elemek, mint:

<head>, <title>, <body>, <header>, <footer>, <article>, <section>, <p>, , , <img>, <aside>, <audio>, <canvas>, <datalist>, <details>, <embed>, <nav>, <output>, <progress>, <video>, <ul>, <ol>, <li>

Rengeteg létezik még ezeken kívül, de jelenleg csak a legfontosabbakat emeltem ki.

A HTML elemeket „tagekkel” választjuk el az azt körülvevő szövegtől. Egy tag nevét „<” nyitó és „>” záró jelölés alkalmazással emeljük ki. Ebből tudja a böngésző, hogy ezt az elemet tagként kell kezelni. Egy tagen belüli szöveg nem kis-és nagy betű érzékeny. Ez azt jelenti, hogy egy <title> elemet lehet <Title> és <TITLE>-ként is írni, nem változtat a jelentésén. Amennyiben egy elemet hibásan írunk meg, és például hiányzik egy záró tag, a böngésző ugyanúgy megjeleníti a tartalmat, de ezesetben hibásan fog megjelenni.

Ezen elemeknek lehetnek egyéb tulajdonságaik is, amelyek értékével állíthatjuk a viselkedésüket, ezzel is több szabadságot adva egy megfelelő struktúra felépítéséhez.

## 2.2. Blade Template Engine

A Blade [2] egy egyszerű, ugyanakkor rendkívül hatékony sablonmotor, amelyet a Laravel [3] tartalmaz. Néhány sablonmotortól eltérően, a Blade nem korlátozza az alap PHP [4] használatát. Minden Blade file végül PHP-ba fordul le, és a gyorsítótárba kerülnek amíg nem lesz bennük módosítás. Ez azt jelenti, hogy semmi plusz terhelés nem éri az alkalmazást. Ez a sablon a blade.php kiterjesztést használja, és a resources/views mappába tárolódik.

## 2.3. CSS

A Lépcsőzetes stíluslapok (Cascading Style Sheets [5]) egy nyelv, amely a leírja a HTML-ben vagy XML-ben [6] íródott dokumentum stílusát. Ez a nyelv megmutatja, hogyan jelenjenek meg az elemek a képernyőn, papíron, beszédben, vagy egyéb médiákon. A létrejöttének oka az volt, hogy a weboldalakon megnőtt az igény az egyre bonyolultabb, kifinomultabb

megjelenésekre és formázásokra, melyeket már egy egyszerű HTML ábrázolással nem lehetett elérni. Léteztek korábban HTML-en belül a szövegek színére, és kinézetére leíró tulajdonságok, azonban miután egyre komolyabb fejlesztések kerültek előtérbe, így azok már kevésnek bizonyultak, hiszen mindegyik oldalon ezeket külön-külön meg kellett adni, és nem lehetett egységesen kezelni, valamint korlátozottak voltak a lehetőségek is.

Ekkorra került igény a CSS kifejlesztésére, amely sokkal rugalmasabbá, és szabadon kezelhetőbbé tette a HTML dokumentumokat, a fejlesztés folyamatát is lerövidítette, valamint az eredményt is javította. A CSS segítségével beállíthatjuk a betűtípusokat, színeket, margókat, távolságokat, egyes elemek elhelyezkedését a weblapon belül és a felsoroltokon kívül még számos más stílussal kapcsolatos beállítást végezhetünk. Egy stíluslapot több oldalhoz is hozzá rendelhetünk, így rugalmasságot és időt spórolhatunk velük. Ha egy stílus lapon belül módosítunk például egy háttér szint, akkor az összes olyan HTML oldalon módosulni fog a háttér, amihez hozzá van csatolva a CSS fájl. Ha a böngészőben megnyitunk egy oldalt akkor az egy úgynevezett cache (gyorsítótár) mappába lementi a stíluslapot, és ha legközelebb az oldalra navigálunk, akkor nem kell megvárni, hogy letöltődjön a CSS fájl, mert a böngésző már a gyorsítótárban tárolja. Így időt spórol a felhasználóknak. Maga a nyelv már viszonylag régóta létezik; 1996. december 17-én készült el.

A CSS beletartozik a W3C [7] által specifikált nyelvek közé, amelyek a mai modern web elengedhetetlen részeit alkotják. Ez azt jelenti, hogy minden böngésző támogatja a CSS használatát. Jelenleg a CSS hármas verziója a legújabb, korábban létezett CSS1 és CSS2, sőt még CSS2.1 is. Ezek specifikálása viszont monolitikus módon történt, ami azt jelenti, hogy minden egyes részét befejezték, majd az egészet egy nagy dokumentációba összesítek. A CSS3 megjelenésével azonban már olyan komplex funkciókat fejlesztenek, hogy ezeket inkább modulokra bontják, és különböző szintekként publikálják, hiszen manapság már nincs egy konkrét vonal, amellyel befejezettnek lehetne nyilvánítani a CSS3-at. Jelenleg például a szelektorok a hármas szinten vannak, és jelenleg dolgoznak a négyes szint megjelenésén, amelyben új funkciók lesznek majd, miközben párhuzamosan folyik a munka más modulokon is, amelyek feltehetőleg később fognak megjelenni.

A lefordult és optimalizált CSS fájlok a public mappában lesznek megtalálhatóak, míg a forrás fileok a resources mappában, ez a folyamat a Laravel Mix segítségével indítható el.

### 2.3.1. Bootstrap

A Bootstrap [8] egy HTML, CSS, Javascript [9] együttes használatát igénylő könyvtár, amelynek célja, hogy egyszerűsítse az informatív weboldalak (webappok) építését. A Bootstrap beimportálása után tudunk válogatni az előre megírt osztályai közül, így nem kell időt tölteni a CSS egyedi megírásával, mivel a Bootstrap előre megírt osztályai, színei, valamint elrendezése jól működik együttesen használva, továbbá reszponzív felépítése mentesít minket a mobilbarát kinézet megírásától, ugyanis a bővítmény ezt biztosítja. Természetesen, amennyiben szükség van további osztályok megírására, erre is van lehetőség, valamint már meglévő osztályokat is felül definiálhatunk. Használata nem különbözik attól, mint mikor egy általunk megírt CSS

osztály alkalmazását hajtanánk végre, ugyanúgy a HTML elemnek kell megadni az osztályt. Jelenleg a 4.6-os verzió a legújabb, amelyet én is használtam, de az ötös verzió nemsokára elérhető lesz.

A Bootstrap alapból megtalálható a Laravelben. Ezt NPM [10] használatával lehet feltelepíteni. A package.json fájlban megtalálható a Laravel projekt indításakor.

## 2.4. JavaScript

A JavaScript (röviden JS) egy programozási nyelv, amelyet kifejezetten az internetre fejlesztettek ki. A legtöbb webböngésző szoftver, és a modern okostelefonok is mind támogatják a JavaScriptet.

A JavaScript-et elsősorban arra használják, hogy gazdagabb, felhasználóbarát élményeket teremtsenek vele az internetet böngészők számára, például dinamikusan frissülő weboldalakat, intuitív felhasználói felületeket, menüket, párbeszédpaneleket, 2D-s és 3D-s grafikákat, interaktív térképeket, videó lejátszókat, és számos egyéb elemet, illetve funkciót. A JavaScript a V8-as [11] motoron fut, amelyet a The Chromium [12] project fejlesztett ki a Chrome böngészőhöz 2008-ban.

A JavaScript az internetes háromszög egyik eleme, a másik kettő pedig a CSS, illetve a HTML. Ezen technológia nem esszenciális, de a fent említett hasznos funkciók miatt nagyon fontos eleme a webdesignnak, hiszen dinamikus weboldalakat lehet létrehozni vele, és lehetővé teszi a programozható elemek elhelyezését is ezeken a weboldalakon.

A Document Object Modelt [13] manipulálhatjuk vele. A DOM egy objektummodell, amire a HTML és az XML is épül. A modell egymással szülő-gyerek kapcsolatban álló objektumok rendszere, melyek segítségével módosíthatjuk a HTML dokumentumok elemeit, vagy a böngésző eseményeit. A DOM nem az a HTML, amit kézzel írunk, és nem is a weboldal forráskódja, hanem amit a böngésző fejlesztői elemeivel lehet megtekinteni.

A Webpack.mix.js fájlban lehet megadni azoknak a JavaScript fileoknak az útját, amelyeket szeretnénk, hogy a Laravel Mix lefordítson. A fordítást npm run dev paranccsal tehetjük meg, vagy az npm run watch paranccsal. A devnél egyetlen egyszer fordul le a JavaScript, míg a watch parancsnál folyamatosan figyeli a fájlokat változásokért és azonnal fordít.

A még le nem fordított javascript a resources/js mappában található, míg a lefordított fájlok tipikusan a public/js mappában találhatóak.

### 2.4.1. jQuery

A jQuery [14] nem más, mint egy gyors és tömör JavaScript nyelvű függvénykönyvtár, ami előre megírt függvényeket tartalmaz. A hagyományos programozási nyelvektől eltérően “csak” értelmez, nem pedig végrehajt. A jQuery megpróbálja leegyszerűsíteni a bonyolultabb JavaScript megvalósításokat (például az AJAX-hívások [15], DOM-manipulációk), így sokkal könnyebben, és gyorsabban használhatjuk azt a weboldalon.

### 2.4.2. Socket.io

A Socket.IO [16] szintén egy JavaScript könyvtár, viszont ezen technológia valós idejű webapplikációkhoz készült. Segítségével valós idejű kétirányú kommunikáció jöhet létre a web kliens és a szerver között. Két részből áll, a kliens oldali könyvtárból, amely a böngészőben fut, valamint a szerver oldali könyvtárból, amely Node.js-re [17] van írva. Ezek mind esemény vezéreltek.

### 2.4.3. Vue

A Vue [18] egy progresszív keretrendszer, amely felhasználói felületek építésére szolgál. A főkönyvtár csak a frontend módosítására szolgál, könnyen kezelhető, és egyszerűen integrálható más függvénykönyvtárak együttes használatával. Ennek ellenére a Vue nagyon könnyen használható Single-Page-Applikációk elkészítésére, csakúgy, mint más frontend keretrendszerek a megfelelő könyvtárak alkalmazásával.

A Laravelben alapból megtalálható a Vue könyvtár. Ezt NPM segítségével lehet feltelepíteni, és ez is a Laravel Mix által fordul le.

### 2.4.4. Node.js

Node.js egy olyan futtató környezet, amely lehetőséget nyújt JavaScriptben írt programok futtatására szervereken. JavaScriptet leginkább a kliens oldalon szokták használni, a böngészőn keresztül. Böngészőn keresztüli megvalósítására a böngésző applikáció fejlesztők számára olyan motorokat kell biztosítani (a Google Chrome esetén a V8 engine), amelyek a JavaScript kódot gépi nyelvként interpretálják majd le is futtatják. Alapvetően a Node.js is ugyanazt a V8 motort használja, némi módosítással.

Annyi könnyebbséget nyújt a Node.js, a többi szerveroldali nyelvvel szemben, hogy ha már dolgoztunk frontend oldalon, nem kell egy teljesen új nyelvet megtanulni ahhoz, hogy a backend-et is implementáljuk.

A másik előnye az, hogy a nonblocking természetéből adódóan alkalmas egyszerre több ezer bejövő „egyszerű” kérés kezelésére. Ezzel szemben egyetlen számításigényes utasítássorozat végrehajtására nem tűnik éppen ideális választásnak. Rengeteg fontos modullal rendelkezik, amelyek nagyon alacsony szinű hozzáférést tesz lehetővé a szerver különböző rendszereihez, viszont magasabb szintű rendszereket önmagában nem kínál. Ehhez külső modulokat használhatunk, amik JavaScript és C++ [19] komponensekből állhatnak (ahogy a beépített modulok is).

### 2.4.5. Node Package Manager

Az NPM (Node Package Manager, rövidítése angolul) egy JavaScript csomagkezelő, alapértelmezésben a node.js. Az NPM segítségével telepíthetjük és kezelhetjük a csomagjainkat az alkalmazásunkhoz.

A Node.js használatakor, ha új modulokat (könyvtárakat) kell telepítenünk, mivel a Node erősen moduláris rendszer szinten üres. Tehát a legtöbb művelethez további modulokat kell telepítenünk. Ez a művelet az NPM eszközzel nagyon egyszerűen elvégezhető. Ez hozzá létre a Laravelben is a node_modules mappát, amely a kliens oldali függőségeket tartalmazza.

## 2.5. PHP

A PHP egy általános szerveroldali szkriptnyelv dinamikus weblapok készítésére. Az első szkriptnyelvek egyike, amely külső fájl használata helyett HTML oldalba ágyazható. A kódot a webszerver PHP feldolgozómodulja értelmezi, ezzel dinamikus weboldalakat hozva létre. Rasmus Lerdorf 1995-ben indította útjára. Ma a The PHP Group tartja fenn és fejleszti. A PHP szabad szoftver, de licence nem csereszabatos a GNU licenccel [20], mivel megkötéseket tartalmaz a PHP név használatára.

A PHP születésekor csupán egy makrókészlet volt személyes honlapok karbantartására. Innen jön az eredeti név is: Personal Home Page Tools. A rövidítés jelentése később PHP: Hypertext Preprocessor lett, így rekurzívvá vált. Később a PHP képességei bővültek, így egy önállóan használható programozási nyelv alakult ki, amely képes nagyméretű webes adatbázisalapú alkalmazások működtetésére is.

A legújabb verziója PHP8, amely sokkal stabilabb, és gyorsabb, mint az elődei.

### 2.5.1. Laravel

A Laravel a PHP szerveroldali szkriptnyelvhez tartozó egyik keretrendszer, egy olyan nyílt forráskódú webes alkalmazás, amellyel gyorsan és egyszerűen lehetséges testre szabott webes alkalmazásokat tervezni.

A fejlesztők azért részesítik előnyben a Laravelt a többi keretrendszerhez képest, mert kiemelkedő teljesítményt nyújt, sok funkciót kínál, illetve skálázható is. Az MVC-t, vagyis a Model View Controllert követi, ezért hatékonyabb és biztonságosabb, mint a natív PHP. Megkönnyíti az olyan általános feladatok elvégzését, mint a hitelesítés (authentication), a munkafázis (sessions), a routing, vagy a gyorsítótárba másolás (caching).

Egyedülálló felépítéssel rendelkezik, melynek segítségével a fejlesztők számára lehetővé válik, hogy egy a specifikusan az alkalmazásukhoz igazított, saját infrastruktúrát hozzanak létre. A Laravel mind nagyobb, mind kisebb projekteknél is alkalmazható. A weboldal admin programozás tekintetében is könnyebbé válik ennek a keretrendszernek a használatával.

### 2.5.2. Composer

A Composer [21] egy alkalmazásszintű csomagkezelő a PHP programozási nyelvhez, amely szabványos formátumot biztosít a PHP szoftver és a szükséges könyvtárak függőségeinek kezeléséhez. Nils Adermann és Jordi Boggiano fejlesztették ki, akik továbbra is irányítják a projektet. A fejlesztést 2011 áprilisában kezdték el, és először 2012. március 1-én adták ki. A zeneszerzőt erősen ihlette a Node.js "NPM" és Ruby [22] "kötegje". A projekt

függőségmegoldó algoritmusa az openSUSE libzypp megoldójának PHP-alapú portjaként indult.

A Composer a parancssorból fut, és függőségeket (például könyvtárakat) telepít egy alkalmazáshoz. Ez lehetővé teszi a felhasználók számára a PHP-alkalmazások telepítését is, amelyek elérhetőek a "Packagist" oldalon, amely a rendelkezésre álló csomagokat tartalmazó fő tárolója.

Továbbá ez generálja le az alkalmazásnak a szerveroldali függéségeket, amelyek a vendor mappában találhatóak.

## 2.6. SQL

Az SQL [23] egy olyan nyelv, melynek célja az volt, hogy egységes lekérdező nyelvként működjön az egyes relációs adatbázis-kezelő rendszerek számára. Annyira sikeressé vált, hogy lényegében a ma használt összes relációs adatbázis-kezelő támogatja. Segítségével könnyen, életszerű mondatokat alkotva lehet lekérdezéseket készíteni, adatokat létrehozni, módosítani vagy törölni az adatbázisból.

Az SQL szabályai az angol nyelv szabályain alapulnak, így azokat viszonylag könnyű megtanulni. Manapság szinte minden vállalatnál szükség van valakire, aki ismeri az SQL-t.

### 2.6.1. SQLite

Az SQLite [24] egy C nyelven [25] íródott könyvtár, amely egy kicsi, gyors, és magas rendelkezésre állású SQL adatbázis motort biztosít nekünk. Jelenleg ez a legtöbbet használt adatbázis most a világon, beépülve mobiltelefonokba, és rengeteg számítógépes alkalmazásba.

## 2.7. Redis

A Redis [26] egy nyílt forráskódú adattároló. Adatbázisnak, gyorsítótárnak, és üzenetek váltására alkalmazzák. Az adatok, amelyekkel a Redis dolgozik többek között string-ek, hash-ek, listák és set-ek.

## 2.8. Git

A Git [27] egy nyílt forráskódú, elosztott verziókezelő szoftver, vagy másképpen egy szoftverforráskód-kezelő rendszer, amely a sebességre helyezi a hangsúlyt. A Gitet eredetileg Linus Torvalds fejlesztette ki egy kernel fejlesztéséhez. Minden Git munkamásolat egy teljes értékű repository teljes verziótörténettel és teljes revíziókövetési lehetőséggel, amely nem függ a hálózat elérésétől vagy központi szervertől.

## 2.9. Felhasznált eszközök

Miután megvizsgáltam és felmértem az interneten létező lehetőségeket, úgy döntöttem, hogy saját alkalmazás fejlesztése mellett döntök, mivel nem volt egy átfogó applikáció, amely kielégítene minden igényt, amire egy ma feltörekvő előadónak szüksége lenne.

A projektem szerveroldali komplexitására és méretére való tekintettel, egyértelmű, hogy PHP-t vagy egyéb szerveroldali nyelvet kell alkalmazni. Korábbi ismereteim révén a PHP-ra esett a választásom, viszont natív PHP-ben egy ekkora projekt megvalósítása nehéz és időigényes lenne. Biztonsági szempontból is körülményes minden egyes lehetséges aspektusra odafigyelni, amelyeket egy keretrendszer magában foglal, ezzel gyorsítva, és megbízhatóbbá téve a munkát. Így a keretrendszer alkalmazása egyértelmű választás volt számomra. A Laravel most a legnagyobb népszerűséget élvező PHP keretrendszer, ami nem is csoda, az egyszerűségét és átláthatóságát tekintve, így nem volt nehéz emellett döntenem.

Mivel webes alkalmazásról van szó, így mindenképpen használnom kellett egy kiszolgálót, amely által az futni tud a böngészőben. Szerencsére a Laravelhez nem kell letölteni semmilyen külső szoftvert, mivel abban ez már beépítve található. Egyszerűen egy php artisan serve paranccsal máris a böngészőnk localhostjára navigálva meg is láthatjuk az alapértelmezett Laravel kezdőlapot. Ezt a szolgáltatást a 8000-es porton teszi meg, így nagy valószínűséggel nem ütközik semmilyen, már futó alkalmazással.

Biztonsági szempontból is ajánlott a keretrendszer használata, mivel míg natív PHP-ben nekünk kell odafigyelni egy esetleges támadás (pl.: SQL Injection [28]) kivédésére, addig ezek a keretrendszerek fel vannak vértezve ennek elkerülési technikáival.

Adatbázis háttér tekintetében több lehetséges szolgáltatás is merülhet fel. A Prototípus időszakban az SQLite használata mellett döntöttem, mivel egy nagyon kicsi és könnyed adatbázisról van szó, prototípusozás esetében pedig nem beszélünk több ezer sornyi adat használatáról, viszont később, amennyiben tesztelésre, vagy használatra bocsátom az alkalmazást, mindenképpen váltok MySQL [29] alkalmazására.

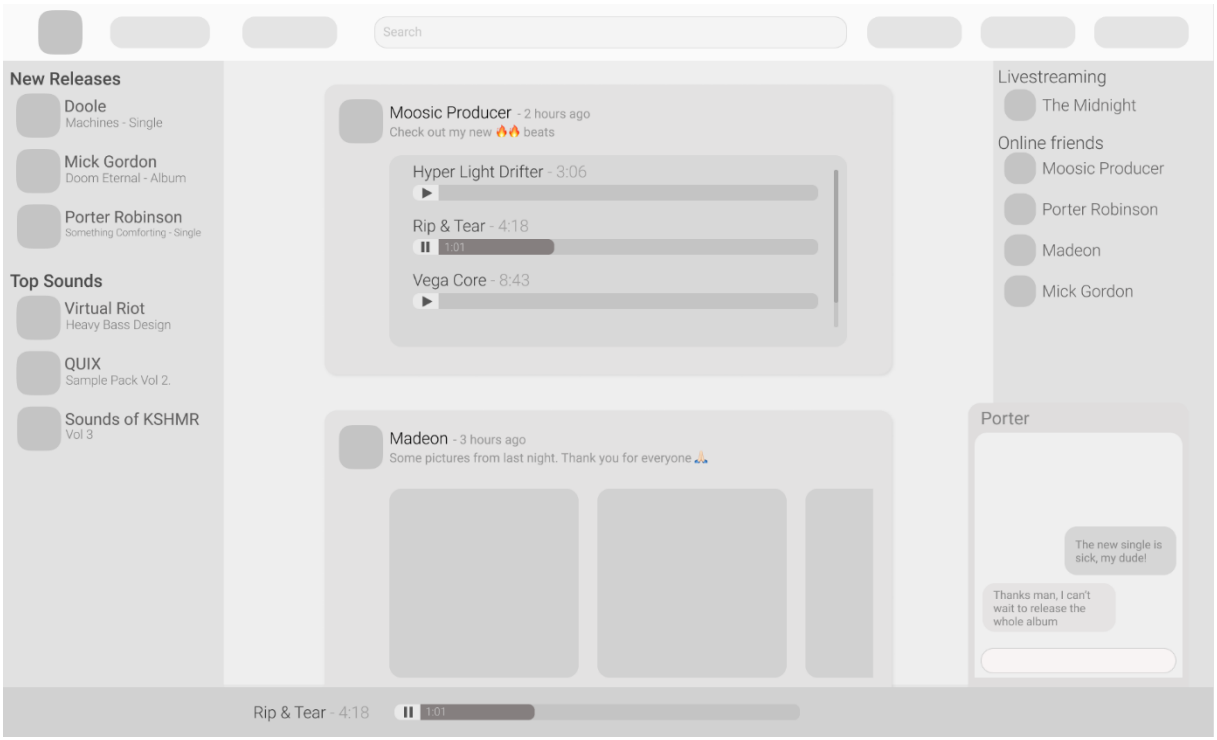
Fejlesztő környezet tekintetében, is rengeteg lehetőségről beszélhetünk. Kisebb méretű – natív PHP-ben megvalósítható – projektek esetében teljesen elégséges választás lenne egy egyszerűbb környezet, azonban nagyobb méretűeknél a fájlkezelés hiányából adódóan használhatatlan. A PhpStorm [30] a Jetbrains [31] által készített szoftver, hasonlít felépítésében, gyorsbillentyű készletében a többi általuk készített applikációhoz, valamint az egyetem licenccel rendelkezik hozzá.

Mivel munkám révén a PhpStorm alkalmazást használom, ezért a projektemben is előnyben fogom ezt részesíteni a további lehetőségek mellett. Amellett, hogy egy erős, megbízható cég áll mögöttük, a PhpStorm platform független. Elérhető több operációs rendszeren is, emellett pedig különböző bővítményekkel hozzá lehet kapcsolni Github [32] fiókunkat is. Így erre a fejlesztő környezetre esett a választásom.

A PhpStorm egy kereskedelmi, platformokon átívelő IDE (integrált fejlesztői környezet) a PHP számára, amelyet a cseh JetBrains cég épített. A PhpStorm egy szerkesztő programot nyújt a PHP, a HTML és a JavaScript számára, menet közbeni kódelemzéssel, hibaelhárítással és a PHP és a JavaScript kód automatikus javításával.

# 3. Felhasználói felület

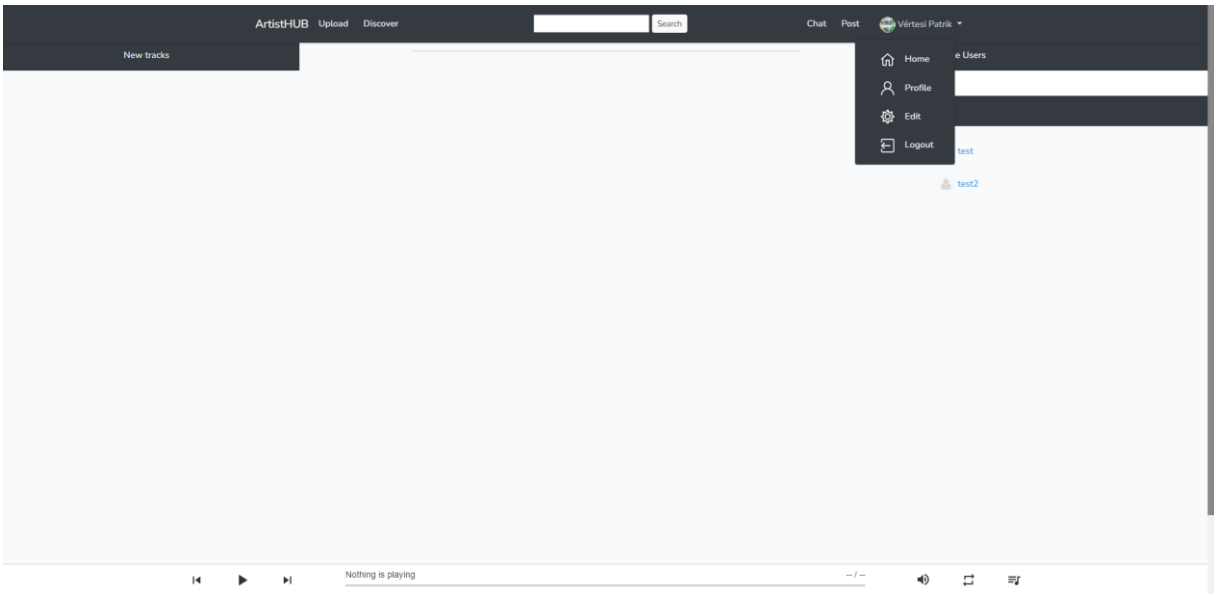
A design kialakításához szem előtt tartottam a modern reszponzív webdesignt, valamint egyszerű, minimalista árnyalatokat használtam. A kinézetet legfőképpen az határozta meg, hogy egy közösségi oldal általában letisztult, csak a lényeg látszik a frontenden, és nagyjából egy sablont is követnek ezek a webappok. Itt jött képbe a Bootstrap használata, mivel ezt rendkívül hatásosan fel lehet használni egy ilyen egységes designra.



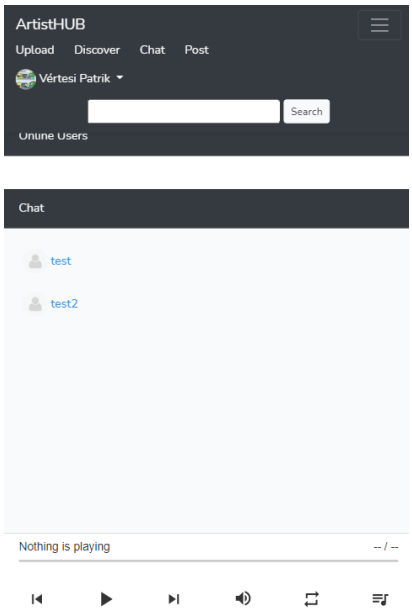

**1. ábra: Drótváz**

A weboldal eredeti terve itt látható. A drótvázat egy Figma [33] nevű webes alkalmazás használatával készítettem. Ahogy majd a későbbiekben látszani fog, az eredeti tervtől néhány helyen eltérhet az alkalmazás. Valamint szerepel rajta pár olyan funkció is, amire még jövőbeni tervként tekintek. A színekhez végül letisztult alapértelmezett Bootstrap színeket választottam. A bg-dark osztály színei a sötétek, valamint a text-light osztály színei használják a szöveget. Ahol világos a háttér, ott a bg-light hátteret használtam.

Az alkalmazásomhoz csak asztali tervet készítettem, ugyanis telefonon már korlátozottak az elrendezési lehetőségek, így a reszponzív designhoz ismét a Bootstrap segítségét használtam, ahol az előre megírt osztályokban eleve reszponzív módon jelennek meg az elemek. A későbbiekben belefutottam néhány egyedi elembe, amelyek nem úgy viselkedtek ahogy szerettem volna, így azokra megírtam a saját osztályaimat.




**2. ábra: Asztali kinézet**




**3. ábra: Telefonos kinézet**

Végül itt látható az elkészült asztali, valamint a mobil kinézet, jelenleg tartalom nélkül. A későbbiekben az összes aloldalra ki fogok térni. Jól láthatóak az egyszerű színek, amelyek még a drótvázban nem szerepeltek.

# 4. Adatbázis felépítése

Az adatbázis kialakítása egy döntő lépes az ilyenféle webalkalmazások esetében. Ebben a fejezetben fogom megadni az adatbázis felépítését, a tervezés folyamatát, a mezők és rekordok meghatározását, valamint a kapcsolatokat, amelyek ezen elemek között léteznek.

## 4.1. Adatbázis megtervezésének folyamata

### 4.1.1. Az adatbázis céljának a meghatározása

Először is fontos eldönteni, hogy mi a célunk a tárolandó adatokkal, valamint egyáltalán milyen adatokat érdemes eltárolni. Majd meghatározhatjuk ezeknek típusát, hosszát, valamint a tárolási formáját.

### 4.1.2. Táblák meghatározása

A meghatározott adatokot csoportokba kell sorolni, amelyek az adatbázis táblái lesznek. Meghatározhatunk akár kapcsoló táblákat is, amelyek a több-több kapcsolat kialakításának segítségére szolgálnak. De kezdő lépésként elég, hogyha csak a témaköröket rendezzük csoportokba, mint például felhasználó vagy üzenetek tábla. Fontos a redundancia kiküszöbölése a hatékonyabb adatbázis érdekében.

### 4.1.3. Mezők meghatározása

Az előbb felsorolt tábláknak meg kell határozni a mezőit, mint például egy felhasználó táblában van azonosító, felhasználónév. Ezeknek érdemes beszédesnek lenniük, hogy egyszerűbb legyen őket kezelni, viszont program szempontjából bárminek lehet őket hívni. Az elsődleges kulcsot szokás ID-nak nevezni, ami általában minden egyes új rekord beszúrásával egyel növekszik. Habár bármi lehet elsődleges kulcs, csak azt kell szem előtt tartani, hogy ennek egyedinek kell lennie, így két rekord nem kaphatja meg ugyanazt az azonosítót.

### 4.1.4. Kapcsolatok meghatározása

Az utolsó lépésben meg kell határozni, hogy a felvett táblák között milyen kapcsolat álljon fenn. Egy táblában szerepelhet idegen kulcs (FK), amely egy másik tábla elsődleges kulcsára (PK) hivatkozik, így ez a kapcsolat fennáll, és nagyban megkönnyíti a lekérdezésekben az adatkinyerést.

## 4.2. User tábla

A user tábla menti a legfontosabb adatokat a regisztráció során. Az ID lesz az elsődleges kulcs, amely automatán növekszik, minden újonnan regisztrált felhasználó egyel nagyobb számot kap az előtte lévőnél. Ez alapján vannak beazonosítva a felhasználók, például a böngésző is az ID-alapján tudja, hogy éppen melyik profilt kell megnyitnia. A név a felhasználó valódi nevére utal, amennyiben a felhasználó Oauth [34] belépést használ, úgy a felhasznált Facebook vagy Google fiókból automatikusan lekérdezésre kerül. Az e-mail cím szintén egy egyedi mező. A provider_id, az Oauthoz tartozó ID-t tárolja. A jelszó egy hash-ben titkosított mezőben kerül

eltárolásra, amennyiben Oauthot használ a felhasználó, úgy nem kell megadnia jelszót, és üresen maradhat a mező, hiszen akkor a token segítségével tud ki-és belépni az alkalmazásba. Viszont amennyiben Oauth nélküli felhasználót szeretne létrehozni, úgy mindenképpen kötelező megadnia a jelszót, az alkalmazás figyelmezteti is a felhasználót, hogyha ezt nem tette meg. Az appban lehetőség van jelszó megjegyzésére is, itt jön képbe a rememberToken, amely csak a felhasználó és a jelszó megjegyzéséhez szükséges tokent tárol el, amennyiben a felhasználó azt kéri.

| Users | | | |
|---|---|---|---|
| **Oszlopnév** | **Oszloptípus** | **Kötelező** | **Tartalom** |
| **id** | INTEGER | igen | Elsődleges kulcs (PK) |
| **name** | VARCHAR | igen | Felhasználó valódi neve |
| **email** | VARCHAR | igen | E-mail cím |
| **provider_id** | VARCHAR | nem | Oauth szolgáltató id |
| **password** | VARCHAR | nem | Jelszó |
| **rememberToken** | VARCHAR | nem | Jelszó megjegyzés |

**1. táblázat: Users tábla**

## 4.3. Profile tábla

A profile tábla a felhasználóhoz tartozó tábla, míg a user tábla az autentikációhoz szükséges adatokat tárolja, addig a profile tábla a módosítható, inkább személyesebb jellegű adatok tárolására való. Az ID itt is az azonosító. A user_id az idegen kulcs, amely a user táblára mutat. Az artistname a felhasználó előadó neve, ez nem egyedi, mivel több előadó is lehet ugyanazzal az előadói névvel. Amennyiben egy felhasználónak van előadó neve, úgy az fog megjelenni mindenhol, amennyiben nem adja meg, mert csak egy hétköznapi felhasználó, nem pedig előadó, úgy a felhasználó neve jelenik meg. A title egy leírás fölötti cím. Mivel a profilban adhat a felhasználó magáról személyes elírást, így, ha szeretné, annak megadhat egy figyelem felkeltő címet. A leírás értelemszerűen pedig egy kis összefoglaló a felhasználóról. Amennyiben rendelkezik a felhasználó webcímmel, úgy ezt meg tudja adni az URL-részhez. A profil és a bannerkép pedig a saját adatlapján megjelenő képek, amelyeket fel tud tölteni.

| Profiles | | | |
|---|---|---|---|
| **Oszlopnév** | **Oszloptípus** | **Kötelező** | **Tartalom** |
| **id** | INTEGER | igen | Elsődleges kulcs (PK) |
| **user_id** | INTEGER | igen | Felhasználó azonosítója (FK) |
| **artistname** | VARCHAR | nem | Felhasználó előadóneve |
| **title** | VARCHAR | nem | Profil leírás címe |
| **description** | TEXT | nem | leírás |
| **url** | VARCHAR | nem | Felhasználó weboldala |
| **profileimage** | VARCHAR | nem | Profilkép |
| **bannerimage** | VARCHAR | nem | Borítókép |

**2. táblázat: Profiles tábla**

## 4.4. Profile_user tábla

Ennek a táblának nincs más szerepe, mint a profil és a felhasználó tábla között egy kapcsolótábla. Ez a több-több kapcsolat miatt szükséges, tartalma csak a tábla saját azonosítója, valamint az idegen kulcsok. Mivel egy felhasználó követhet több profilt, valamint egy profilt követhet több felhasználó, így ezzel a módszerrel lett megoldva a követő rendszer, amely adatainak tárolásához ez a tábla játszik kulcsszerepet.

| Profile_user | | | |
|---|---|---|---|
| **Oszlopnév** | **Oszloptípus** | **Kötelező** | **Tartalom** |
| **id** | INTEGER | igen | Elsődleges kulcs (PK) |
| **user_id** | INTEGER | igen | Felhasználó azonosítója (FK) |
| **profile_id** | INTEGER | igen | Profil azonosítója (FK) |

**3. táblázat: Profile és Users kapcsolótábla**

## 4.5. Posts tábla

A posts táblában tárolódik minden post-hoz szükséges adat. Ezek a rekordok tartalmazzák a tábla azonosítására szolgáló ID-t, valamint szintén itt is megtalálható a user_id mező, amely idegen kulcsként szolgál a user táblához való összekötésre. A text itt maga a bejegyzés szövege. Legutolsó sorban pedig az image, ami a bejegyzés képe. A képet leszámítva a többi mezőt kötelező kitölteni a post létrehozásához.

| Posts | | | |
|---|---|---|---|
| **Oszlopnév** | **Oszloptípus** | **Kötelező** | **Tartalom** |
| **id** | INTEGER | igen | Elsődleges kulcs (PK) |
| **user_id** | INTEGER | igen | Felhasználó azonosítója (FK) |
| **text** | VARCHAR | igen | Bejegyzés szövege |
| **image** | VARCHAR | nem | Bejegyzéshez tartozó kép |

**4. táblázat: Posts tábla**

## 4.6. Music tábla

A music tábla felel az eltárolt zenék adataiért, ebben megtalálható a megszokott azonosító és az idegen kulcs, amely a felhasználóhoz társítja a zenét. Ezen felül megtalálható a feltöltött zene stílusa, az előadó neve, (ami akár lehet a saját nevünk is, de termesztésen más előadó névvel is feltölthetünk zenét a saját profilunkra) a zeneszám címe, valamint a hozzátartozó borító és maga a file.

| Music | | | |
|---|---|---|---|
| **Oszlopnév** | **Oszloptípus** | **Kötelező** | **Tartalom** |
| **id** | INTEGER | igen | Elsődleges kulcs (PK) |
| **user_id** | INTEGER | igen | Felhasználó azonosítója (FK) |
| **genre** | TEXT | igen | Zene stílusa |
| **artist** | TEXT | igen | Előadó neve |
| **songtitle** | TEXT | igen | Zeneszám címe |
| **image** | TEXT | igen | Zene borítóképe |
| **audio** | TEXT | igen | Zene file |

**5. táblázat: Music tábla**

## 4.7. Comments tábla

Ez a tábla tartalmazza a bejegyzéshez, vagy zenéhez készült hozzászólásokat. Itt csak a felhasználó azonosítója, valamint a hozzászólás saját azonosítója kötelező, ugyanis amennyiben zenéhez szól hozzá a felhasználó, úgy onnan kap egy azonosítót, és a bejegyzés része üres marad, vagy fordítva. Ezzel a tervezéssel kevésbé lesz redundáns az adatbázis, ugyanis nem kell a bejegyzés, és a zene hozzászólásait külön tárolni, hanem így egy táblából elérhető minden. Ezen kívül a hozzászólás tartalma található még itt.

| Comments | | | |
|---|---|---|---|
| **Oszlopnév** | **Oszloptípus** | **Kötelező** | **Tartalom** |
| **id** | INTEGER | igen | Elsődleges kulcs (PK) |
| **user_id** | INTEGER | igen | Felhasználó azonosítója (FK) |
| **post_id** | INTEGER | nem | Bejegyzés azonosítója (FK) |
| **music_id** | INTEGER | nem | Zene azonosítója (FK) |
| **body** | VARCHAR | nem | Hozzászólás szövege |

**6. táblázat: Comments tábla**

### 4.8. Likes tábla

A hozzászólásokhoz hasonlóan ugyanaz a koncepció itt is, azt az egy változást hozzáértve, hogy itt nincsen tartalom, csak azt tárolja a tábla, hogy az adott zene vagy bejegyzés kedvelve lett-e, vagy nem.

| Likes | | | |
|---|---|---|---|
| **Oszlopnév** | **Oszloptípus** | **Kötelező** | **Tartalom** |
| **id** | INTEGER | igen | Elsődleges kulcs (PK) |
| **user_id** | INTEGER | igen | Felhasználó azonosítója (FK) |
| **post_id** | INTEGER | nem | Bejegyzés azonosítója (FK) |
| **music_id** | INTEGER | nem | Zene azonosítója (FK) |

**7. táblázat: Likes tábla**

### 4.9. Messages tábla

A chat legfontosabb részéhez tartozó adatok itt tárolódnak, amely nem más, mint a tábla elsődleges kulcsa, valamint az üzenet tartalma.

| Messages | | | |
|---|---|---|---|
| **Oszlopnév** | **Oszloptípus** | **Kötelező** | **Tartalom** |
| **id** | INTEGER | igen | Elsődleges kulcs (PK) |
| **message** | TEXT | igen | Üzenet tartalma |

**8. táblázat: Messages tábla**

### 4.10. User_messages tábla

Az üzenetek és a felhasználók közötti kapcsolótábla. Az elsődleges kulcson kívül csak idegen kulcsok találhatóak a táblában, amelyek az üzenet azonosítója, a küldő, valamint a fogadó azonosítója.

| User_messages | | | |
|---|---|---|---|
| **Oszlopnév** | **Oszloptípus** | **Kötelező** | **Tartalom** |
| **id** | INTEGER | igen | Elsődleges kulcs (PK) |
| **message_id** | INTEGER | igen | Üzenet azonosítója (FK) |
| **sender_id** | INTEGER | igen | Küldő azonosítója (FK) |
| **receiver_id** | INTEGER | igen | Fogadó azonosítója (FK) |

**9. táblázat: Users és Messages kapcsolótábla**

# 5. Séma megtervezése

Ebben a fejezetben a korábban felvázolt táblák közötti kapcsolatokat [35] szeretném tárgyalni, az egyes mezők egymáshoz való viszonyulását, valamint a lehetséges kapcsolatok fajtáit, amik egy ilyen adatbázis használatakor felmerülhetnek.

## 5.1. Lehetséges kapcsolatok

A táblák mezőit úgy kell összehangolni, hogy ugyanazon rend szerint írják le a tárolt adatokat. Ezt az összehangolást a táblák között létesített kapcsolatok segítségével adjuk meg. A kapcsolat a kulcsmezők értékeit rendeli egymáshoz. A kulcsmezők általában a két táblában ugyanazzal a mezőnévvel rendelkeznek. A legtöbb esetben az egyik táblában szereplő kulcsrekord az ún. elsődleges kulcs, amely minden rekord szempontjából egyedi rekordazonosítást tesz lehetővé, a másik táblában pedig ez a külső kulcs.

### 5.1.1. Egy az egyhez (one-to-one)

Egy-az-egyhez kapcsolat esetén az A tábla minden egyes rekordjához legfeljebb egy rekord tartozhat a B táblában, és a B tábla minden egyes rekordjához is csak legfeljebb egy rekord tartozhat az A táblában. Ez a fajta kapcsolat nem túl gyakran használatos, mert a legtöbb információ, amelyet ilyen módon írunk le, leírható egyetlen táblán belül is. Az egy-az-egyhez kapcsolat akkor lehet hasznos, ha egy sok rekordból álló táblát több kisebb, könnyebben kezelhető táblára kívánunk felosztani, ha egy tábla valamely részét adatvédelmi megfontolásból külön kívánjuk tárolni, vagy ha az egyik táblában olyan adatokat szeretnénk tárolni, amely a fő táblában csak bizonyos rekordokra érvényes.

### 5.1.2. Egy a többhöz (one-to-many)

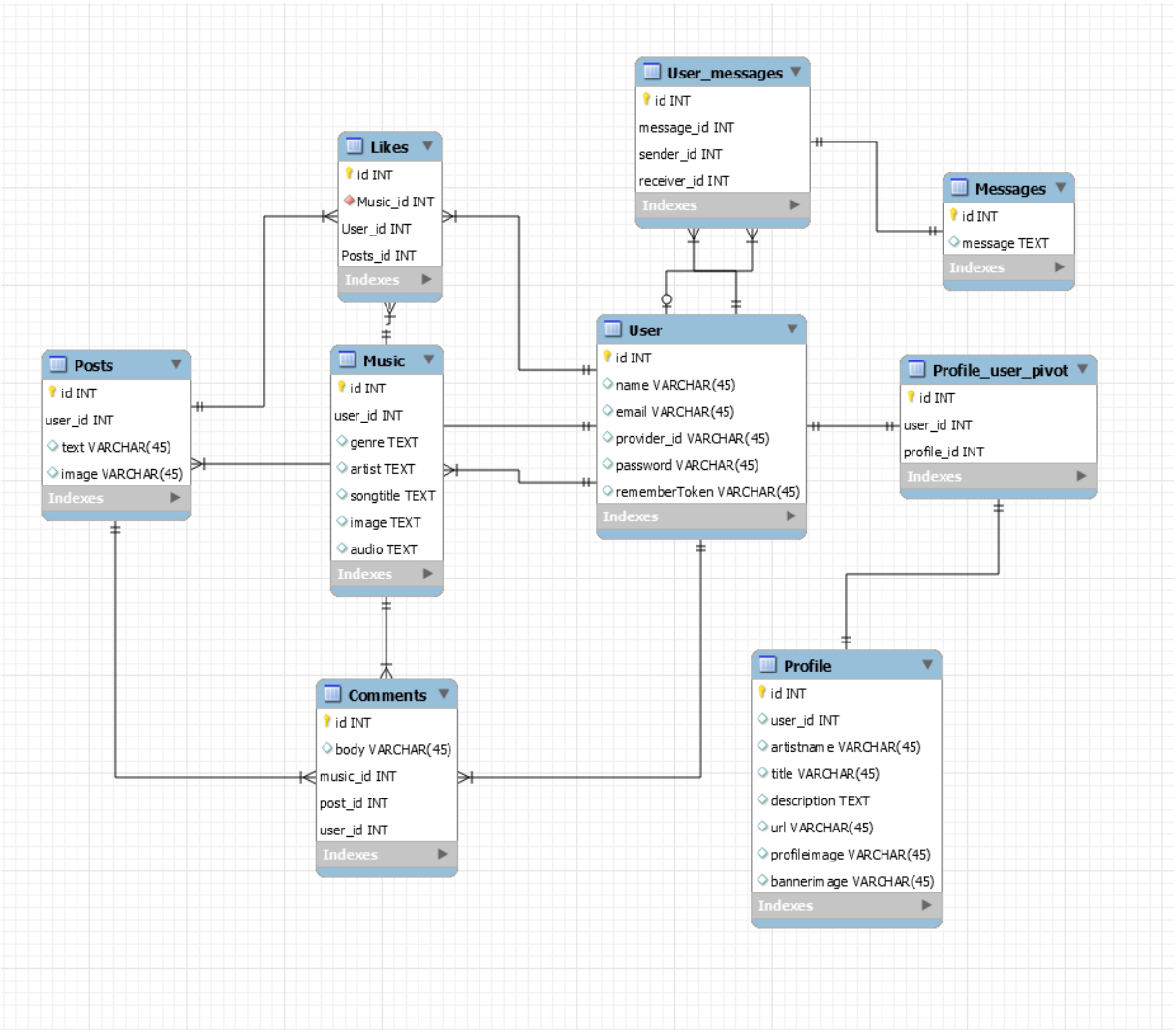
Az egy-a-többhöz kapcsolat a leggyakrabban használatos kapcsolati típus. Az egy-a-többhöz kapcsolatban az A tábla valamely rekordjához több rekord tartozhat a B táblában, de a B tábla valamennyi rekordjához csak egy-egy rekord tartozhat az A táblában.

### 5.1.3. Több a többhöz (many-to-many)

Több-a-többhöz kapcsolat esetén az A tábla valamely rekordjához több rekord is tartozhat a B táblában, és a B tábla valamely rekordjához is több rekord tartozhat az A táblában. Ez a fajta kapcsolat csak úgy lehetséges, hogy definiálunk egy harmadik táblát, az ún. illesztőtáblát, melynek elsődleges kulcsa két mezőt tartalmaz: az A tábla és a B tábla külső kulcsát. A több-a-többhöz kapcsolat tehát válójában két egy-a-többhöz kapcsolat egy harmadik tábla részvételével.

## 5.2. Séma

Az adatbázis sémáját és a táblák közötti kapcsolatokat a következő osztálydiagramon mutatom be. A terv a MySQL Workbench [36] használatával készült.




**4. ábra: Adatbázis séma**

# 6. Specifikáció és tervezés

Ebben a fejezetben fogok a webapplikáció felépítésével foglalkozni, felsorolom a funkcióit és kifejtem az egyes elemek működését. Meghatározom, hogy egyes felhasználók, milyen funkciókat érhetnek el, és milyen módon tudják használni a rendszert.

## 6.1. Általános elvárások

Egy rendszer biztonsága, használhatósága érdekében különböző megkötéseket kell alkalmazni. Ilyen megkötések általában az e-mail címekre, jelszavakra értendőek, de különböző beviteli mezőkre is érdemes ezt a módszert alkalmazni. Ezeket a megkötéseket általában JavaScript szkriptekkel végezzük el, de ebben az esetben a Laravel ezt alapból biztosítja nekünk.

Ezek a szkriptek jellegzetesen úgynevezett reguláris kifejezések alapján ellenőrzik, hogy a begépelt szöveg megfelel-e az elvárásoknak. Meghatározható, hogy adott beviteli mező kifejezetten milyen értékeket vehessen fel, például: egy név mezőbe ne írhassunk számot, egy e-mail mezőbe az e-mail általános elvárásainak megfelelő formátumú szöveg kerülhessen. Meghatározhatjuk, hogy a jelszó mezőben milyen kötelező karaktereknek kell szerepelniük, és melyek azok, amelyek nem szerepelhetnek.

## 6.2. Bejelentkezés nélküli funkciók

Akármilyen felhasználó kezelést felhasználó alkalmazás során biztosítani kell az oldalra érkező látogatóknak néhány regisztrálás nélküli funkciót. Néhány esetben ezek lehetnek olyan lehetőségek, amelyekkel megfogjuk a látogatót. Bizonyos tartalmakat megmutatunk neki, majd, ha többet akar látni ahhoz egy felhasználót kell létrehoznia. Az én webalkalmazásom esetében, csak a felhasználó létrehozásához, vagy a bejelentkezéshez szükséges lépeseket tudja megtenni amennyiben nincs belépbe.

### 6.2.1. Regisztráció

Az alkalmazásnak biztonságosan kell tudnia tárolni a felhasználók adatait és én is biztosítani akarom, hogy csak a regisztrált tagok láthassák egymást és léphessenek egymással kapcsolatba. A felhasználó neveknek egyedieknek kell lenniük, a sikeres regisztrációhoz pedig e-mail hitelesítés szükséges. Azért tartottam ezt a lépést szükségesnek, hogy valóban csak azok az emberek regisztráljanak, akik használni is akarják az alkalmazást. A jelszavak tárolásának felfedése még az adatbázisban is elfogadhatatlan, így szükséges a jelszavak eltárolásának a titkosítása.

### 6.2.2. Bejelentkezés

Miután a felhasználó sikeresen regisztrált, az általa megadott felhasználónév/e-mail cím és jelszó párossal tud belépni az oldalra. Ha valamelyik hibás a kettő közül, vagy üresen maradt a mező, jelezni kell a felhasználó felé. Elfelejtett jelszó esetén, egy erre kialakított linken keresztül lehet újat igényelni. Ennek működéséhez kell egy dedikált levelező szerver, vagy SMTP-n keresztül kell csatlakozni egy már létező szerverhez. Ez a modul úgy működik, hogy

miután beírtuk e-mail címünk és rákattintottunk a küldés gombra, generál egy „token” értéket, amit időbélyeggel ellátva elment az adatbázisba. Legenerálja az új, ideiglenes jelszót, majd ezt elküldi a felhasználónak e-mail formájában.

## 6.3. Bejelentkezés utáni funkciók

Miután létrejött a profili, vagy belépett láthatja magát a webalkalmazást. Itt már hozzáfér korlátok nélkül biztonságosan minden egyes funkcióhoz. Ezeket a lehetőségeket fogom bemutatni a következőkben.

### 6.3.1. Főoldal

Amennyiben a felhasználó be van jelentkezve, ezen a felületen fogja a legtöbb információt találni. A 2. és 3. ábrán ezt az oldalt láthatjuk. Itt a hírfolyamba láthatja majd más felhasználók posztjait, itt értesülhet a legújabb zenékről, hangokról, itt tud beszélgetést kezdeményezni, valamint erről az oldalról bármelyik funkciót elérheti.

### 6.3.2. Profil szerkesztése

Itt lehetősége van a felhasználónak megadni a személyes adatait, valamint profilképet és borítóképet is feltölthet. Amennyiben nem ad meg profilképet, úgy kap egy alapértelmezettet. Továbbá ebben a funkcióban is jelezni kell a felhasználó felé az esetleges hiányosságot és/vagy az adatok szerkesztésének sikerességét.

### 6.3.3. Azonnali üzenetváltás

A Chat fület megnyitva láthatja a felhasználó az összes további felhasználót, akivel beszélgetést kezdeményezhet. Amennyiben ír valakinek, azt a másik felhasználó azonnal megkapja.

### 6.3.4. Elérhető felhasználók

Itt az összes elérhető felhasználót láthatjuk, ha be vagyunk jelentkezve.

### 6.3.5. Posztolás

A bejelentkezett felhasználók tartalmakat, zenéket, hangokat tudnak megosztani egymással, amelyre a többi felhasználó reagálni tud, meg tudja osztani újra, vagy épp hozzászólni tud hozzájuk.

### 6.3.6. Zene lejátszás, feltöltés, letöltés

Hasonlóan, mint a posztolásnál, itt is reagálhatnak a hallgatók a zenékre, lementhetik őket.

### 6.3.7. Kedvelés

A bejelentkezett felhasználok tudnak reagálni egymás zenéire, bejegyzéseire. Minden felhasználó minden bejegyzést csak egyszer kedvelhet, amennyiben már kedvelve van az adott tartalom, csak levenni van lehetősége a kedvelést.

#### 6.3.8. Hozzászólás

A megvalósítás a kedveléshez hasonlít, azonban itt egy bejegyzéshez, vagy zenéhez bármennyi hozzászólást adhatunk, és ehhez termesztésen van tartalom is.

#### 6.3.9. Felfedezés

A felfedezés fülön találhatunk új felhasználókat, akiket érdemes követnünk, az összes legfrissebb zenét, valamint azokat a zenéket, amelyeket korábban kedvelt műfaj alapján kapjuk ajánlásba.

#### 6.3.10. Tartalom törlése

Amennyiben egy bejegyzés, hozzászólás vagy zene, már nem szükséges a felhasználó profilján, és úgy dönt törli, ezt megteheti. Minden saját tartalom mellett látható egy X, ami megnyomásával kitörlődik az adat az adatbázisból, és így az alkalmazásból is.

#### 6.3.11. Képek feltöltése

Saját profil szerkesztésekor profilképet, vagy épp más témájú képeket tölthetnek fel, és oszthatnak meg egymással a regisztrált felhasználók.

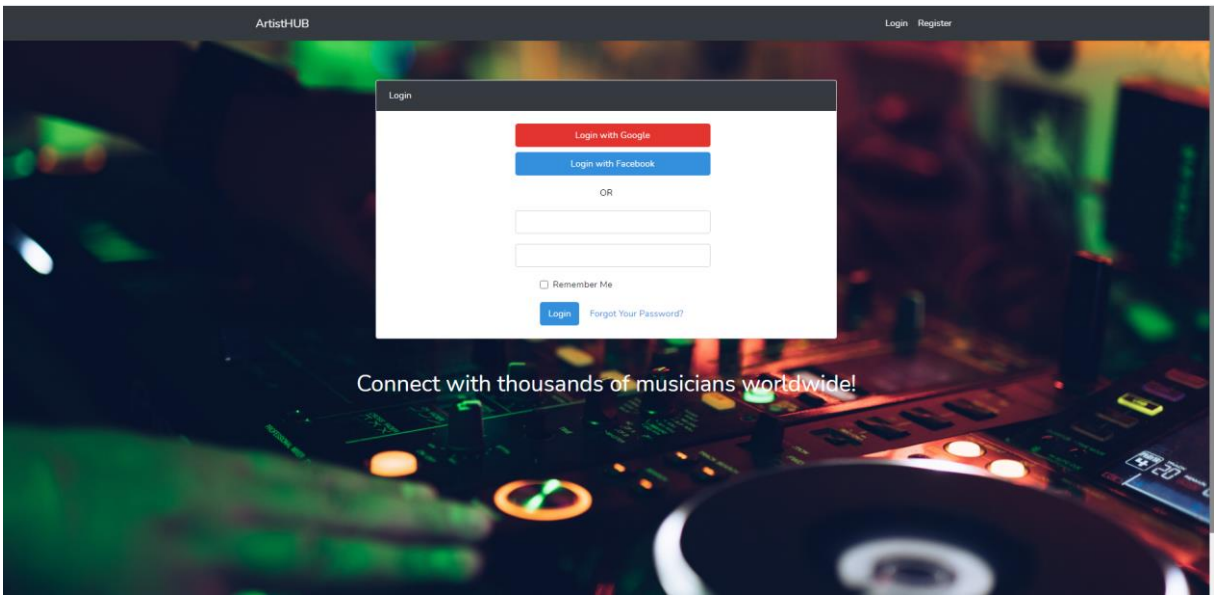
#### 6.3.12. Kijelentkezés

Értelemszerűen itt tudja magát kijelentkeztetni a felhasználó. Ha az ember bejelentkezik egy idő után ki is akar lépni, ha nem kíván tovább az oldalon maradni és nem akarja, hogy más emberek ugyanazon a számítógépen bejelentkezve lássák a személyes profilját. Erre minden esetben lehetőséget kell biztosítani egy ilyen jellegű alkalmazásnál.

# 7. Implementáció

A korábban leírt funkciók fejlesztését, és a felhasználásuk lehetőségeit fogom bemutatni ebben a fejezetben.

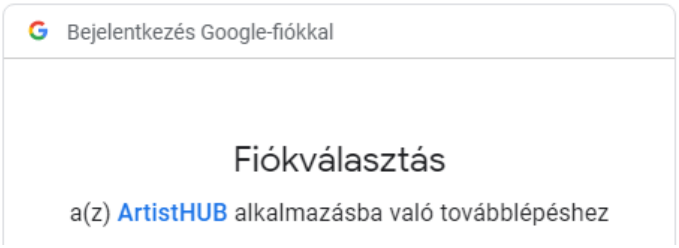
## 7.1. Regisztráció és bejelentkezés




**5. ábra: Bejelentkező felület**

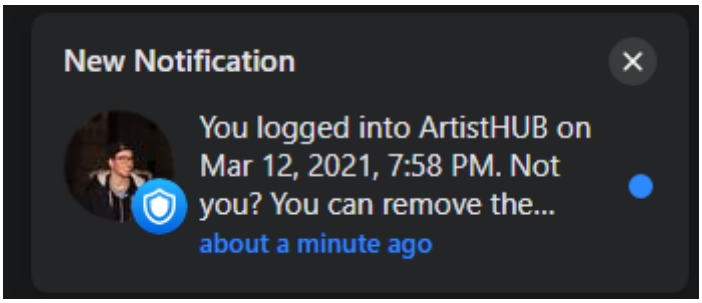
A belépési fülre érkezve a következő felület vár minket. Mivel az alkalmazást csak olyan felhasználók használhatják, akik regisztrálva vannak, ezért kötelező profillal rendelkezni. Amennyiben a felhasználó megpróbálna egy létező linket beírva (például egy felhasználó profiljának linkjét) elnavigálni a kezdőoldalról, az alkalmazás automatikusan visszairányítja, ugyanis bejelentkezés nélkül, nincs jogosultsága ezen linkek megtekintésére. Erre legfőképpen a biztonság miatt is szükség van. De nagyban hozzájárul ahhoz is, hogy így, ha valakit érdekel az alkalmazás, mindenképpen regisztrálni fog, ezzel elősegítve az alkalmazás növekedését.

Lehetőség van Oauth belépésre, valamint hagyományos profillal való belépésre. Amennyiben a felhasználó az Oauth mellett dönt, úgy Google [37] és Facebook [38] profiljával egy kattintás alatt már tudja is az alkalmazást használni. Az alkalmazás nem fogja lekérni és eltárolni a felhasználó jelszavát, Oauth használata közben, ezért hagyható üresen a táblában a jelszó mező, ahogyan már korábban említettem. A felhasználó név ezesetben a valódi név lesz, amit később meg lehet változtatni. Amennyiben Google fiókkal lépne be a felhasználó, úgy az alkalmazás a következő felületre irányítja, ahol meg kell adnia melyik Google fiókját szeretné használni az alkalmazáshoz.



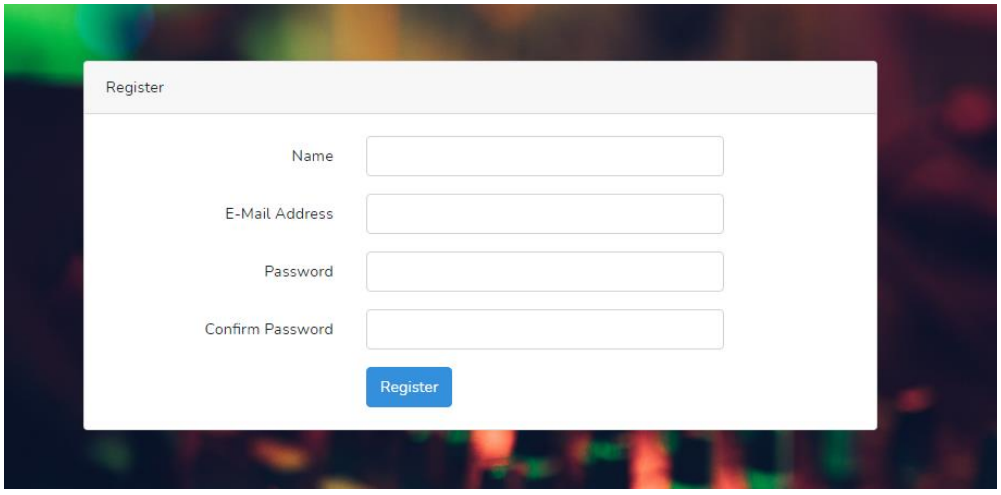

**6. ábra: Google bejelentkezés**

Facebook-os használatot követően szintén meg kell adni a felhasználót, amelyikkel be szeretnénk lépni, majd a saját Facebook profilunkon keresztül kapunk egy értesítést, hogy az alkalmazás hozzáfért a Facebook fiókhoz.




**7. ábra: Facebook bejelentkezés**

Amennyiben a felhasználó hagyományos regisztrációt szeretné használni, úgy ez a felület fogja várni a regiszter fülön.

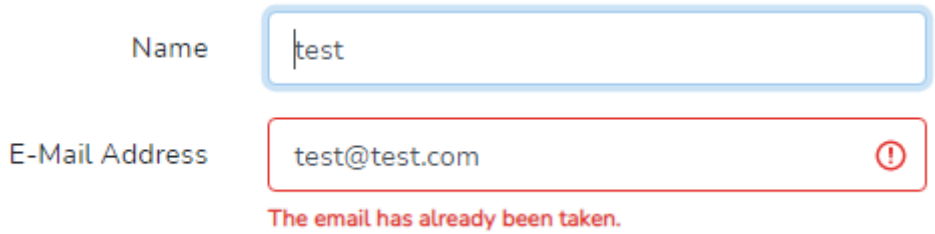



**8. ábra: Regisztrációs felület**

A következőkben a regisztrációhoz szükséges adatokat, és a regisztráció sikerességéhez szükséges megkötéseket szeretném tárgyalni. Fontos, hogy egy felhasználónak legyenek egyedi sorai is a táblában, amivel könnyen elkerülhető az, hogy esetleg két vagy több felhasználó a későbbiekben összeakadjon esetleg egy keresés, vagy bármilyen jellegű lekérés során.

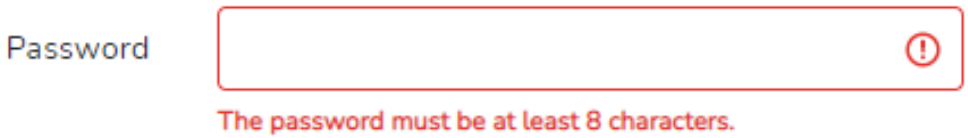
A Name nem egyedi, mivel nagyon sok két ugyanolyan nevű ember létezhet, emiatt a megkülönböztetésre az azonosítót és az e-mail címet használjuk. Mivel az e-mail cím egyedi,

ezért, ha már foglalt, az alkalmazás figyelmeztet minket, hogy ezt nem használhatjuk, és módosítsunk rajta. A felhasználó nevén felül egy user létrehozásakor az alkalmazás hozzárendel mindenkihez egy userID-t ahogyan azt korábban az adatbázis tervben láthattuk. Ez egy inkrementális szám, ami minden egyes felhasználó regisztrációjakor szimplán az előző regisztrált felhasználóhoz képest az egyel nagyobb számot kapja meg. Természetesen mivel a felhasználónevünk nem egyedi, ez így egy fontos lépése a felhasználó létrehozásának, valamint a profil URL-jében a későbbiekben az ID alapján hivatkozunk egy felhasználóra, nem pedig a user neve alapján.



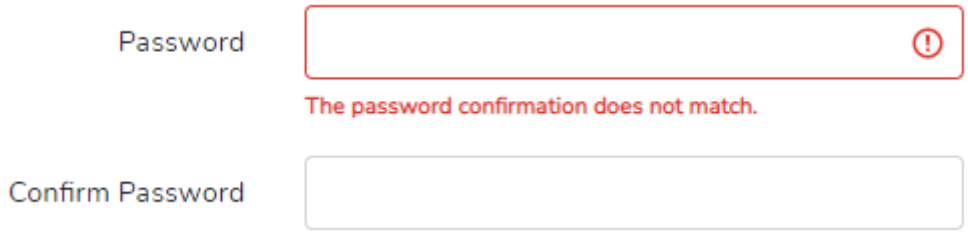

**9. ábra: Felhasználó adatainak megszorításai**

Amint láthatjuk a névből lehet több is, e-mail címből viszont nem. Az alkalmazás figyelmeztet amennyiben már korábban regisztrált valaki ezzel az e-mail címmel. Mivel Oauth bejelentkezés esetén is eltárolásra kerül az e-mail cím, így akkor is figyelmeztet minket a program, hogyha Oauthos regisztrációkor lett lefoglalva az e-mail cím. Viszont, ha a felhasználó élt az Oauth lehetőséggel, úgy mindig azt kell használnia a belépéshez, mivel az alkalmazáson belül nem létezik jelszava amellyel tudná azonosítani a program, ezt teszi meg helyette a Facebook vagy a Google.



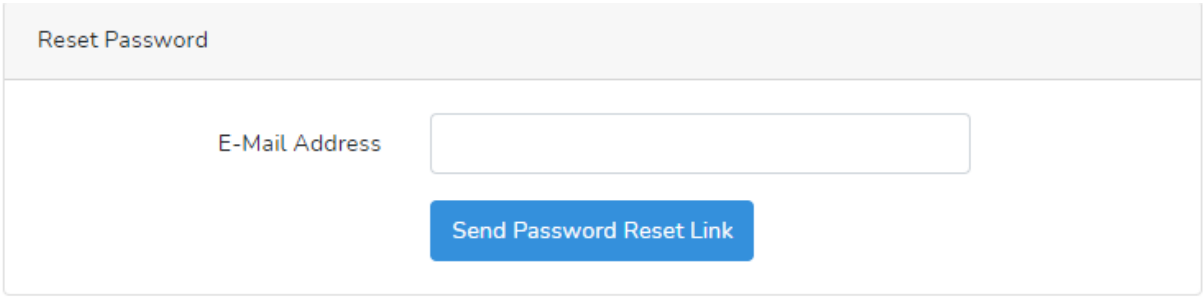

**10. ábra: Felhasználó jelszava**

A megfelelő biztonság érdekében ahogyan a fenti ábra is mutatja, a jelszónak legalább 8 karakternek kell lennie. Ezzel ki lehet kerülni, hogy a felhasználó 1-2 karakteres jelszavakat adjon meg, amit később könnyen fel lehetne törni. Manapság nagyon sokan próbálkoznak személyes adatokat, főleg bankkártya adatokat szerezni. Nagyon sok SQL injection, és sok más féle támadás éri manapság a webapplikációkat. Szerencsére a Laravel keretrendszernek van ezek ellen automatikus védelme. Az eloquent model [39] segítségével ez könnyen kezelhető. Paraméter kötést használ a háttérben figyelmen kívül hagy bármilyen inputot ami a where()-en belül kerül beadásra külső forrásból.




**11. ábra: Felhasználó jelszó egyezése**

Az utolsó hibaüzenet, amit kaphatunk egy profil létrehozásakor, természetesen az, hogyha a jelszó, és a jelszó megerősítés mező nem egyezik, a program szól a felhasználónak, hogy elgépelt valamit, és próbálja újra. Ez mindenképpen szükséges, elvégre előfordulhat, hogy valaki elgépel valamit a jelszavában.

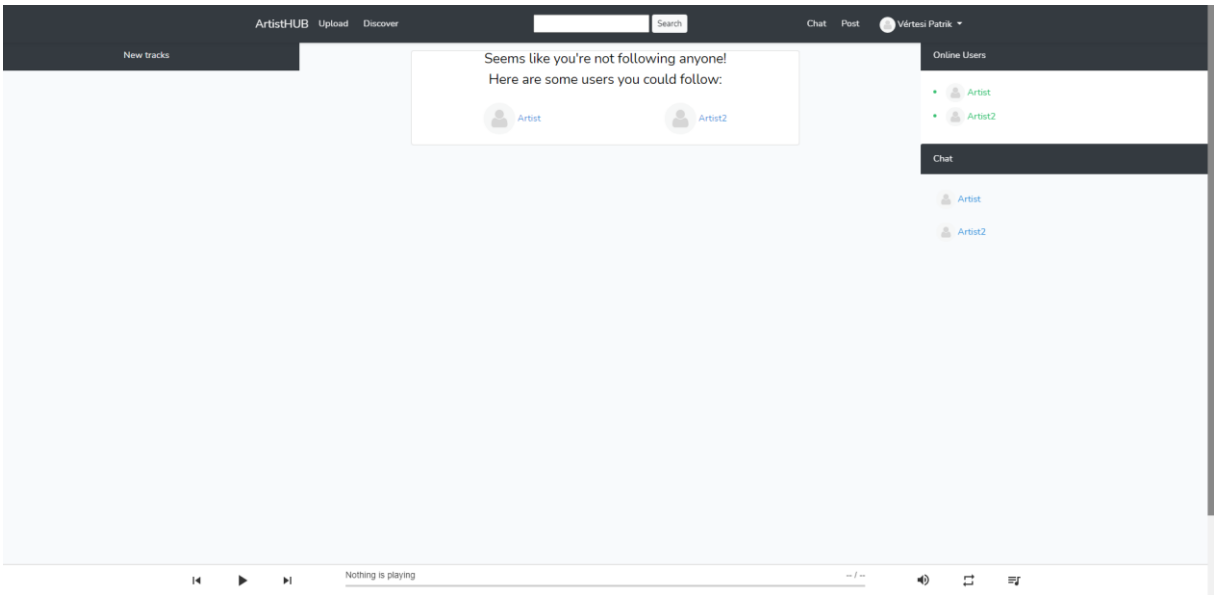



**12. ábra: Elfelejtett jelszó**

A feljebb említett esetben a felhasználó minden probléma nélkül tud kérni egy jelszó újra küldést. A megadott e-mail címre fog érkezni, és amennyiben az e-mail cím megfelel a felhasználónévhez tartozó e-mail címnek, megadhat egy új jelszót, majd a későbbiekben ezzel fog tudni bejelentkezni az alkalmazásba.

Amennyiben minden megszorításnak eleget tettünk, a felhasználó profilja létrejött. Ezzel a művelettel a user mellé automatikusan létrejön a hozzá tartozó profil, a user tartalmazza az azonosításhoz tartozó adatokat, mint a korábban említett ID, a profilhoz pedig az olyan adatok tartoznak, mint a felhasználó profilképe, előadó neve, vagy a leírása. Ezt fogják más felhasználók látni publikusan a szóban forgó előadóról, amennyiben meg vannak adva. Amennyiben nem ad meg előadó nevet, csak átlag felhasználóként szeretné használni az alkalmazást, a valódi nevét fogja használni az alkalmazás. Ezeket az adatokat bármikor megváltoztathatja, kivéve a valódi nevét.

## 7.2. Bejelentkezés után



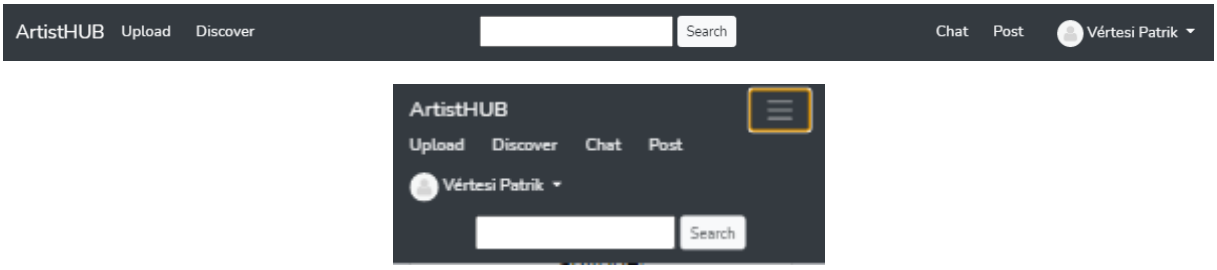

**13. ábra: Belépés utáni kezdőkép**

A bejelentkezés után ez a kép fogad bennünket. Amennyiben még nem követünk senkit, úgy az alkalmazás ajánl nekünk felhasználókat, akiket bekövethetünk, hogy elkezdjük a platform használatát. Erre azért van szükség, mert amennyiben valaki nem ismeri az alkalmazást, úgy elég nehezen tudna elindulni a használatával, hogyha azt se tudja mit keressen. Ennek segítségére hoztam még létre a Discover aloldalt, amelyet a későbbiekben még kifejtek.

A New tracks alatt az általunk követett felhasználók legújabb zenéit láthatjuk, középen ugyanígy a felhasználók posztjait, akiket követünk. Jobb oldalt láthatjuk az éppen aktív felhasználókat, valamint alatta azokat a felhasználókat, akikkel tudunk beszélgetni. Ezeket a funkciókat bővebben kifejtem a következő fejezetekben.

## 7.3. Navbar

A funkciók nagy részét a navigation bar segítségével érhetjük el. Mint minden modern weboldalnak, ennek is szükséges a használata. Jelenleg a Bootstrap alapértelmezett navigation barjat használom, amihez hozzáadtam az alkalmazáshoz szükséges funkciókat, amelyek útvonalát a Laravel speciális routing fájljának segítségével állítom be.



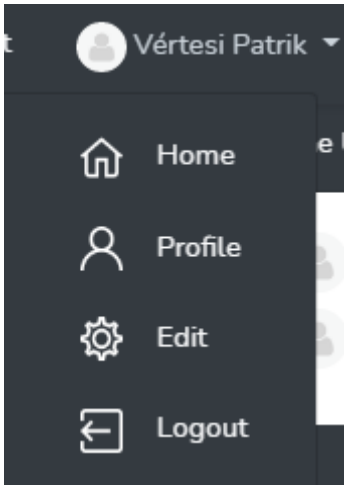

**14. ábra: Navigation bar**

Az útvonalak, és funkciók ezen a részen a következők:

- Az ArtistHUB kattintásával visszakerülünk a kezdőlapra.
- Az Upload alatt tudunk új zenét feltölteni.
- A Discover fül alatt látunk új felhasználókat, akiket érdemes követnünk, az összes zenei újdonságot, valamint szűrve azokat az ajánlásokat, amelyek az általunk kedvelt műfajba tartoznak.
- A keresővel átnavigálhatunk olyan előadók profiljára, akiket még nem követünk.
- A Chat fülön megjelenik a beszélgető felület, ahol minden felhasználó elérhető üzenetváltásra.
- A Posttal létrehozhatunk új bejegyzést a profilunkon.

Ezeket mind kifejtem a dokumentumom további részeiben, egy-egy fejezetet szentelek minden fontosabb funkciónak.

A felhasználó specifikus útvonalakat a lenyíló menü segítségével érjek el a navigation bar jobb oldalán. Az itt megtalálható funkciók minden felhasználónként egyedi, saját beállításaikat figyelembe véve változhat.




**15. ábra: Lenyíló menü**

A Profilnév természetesen az lesz a lenyíló menü felett, amit megadtunk a regisztrálásnál, ezesetben, ha például test lenne a felhasználónk neve, úgy tesztet látnánk itt.

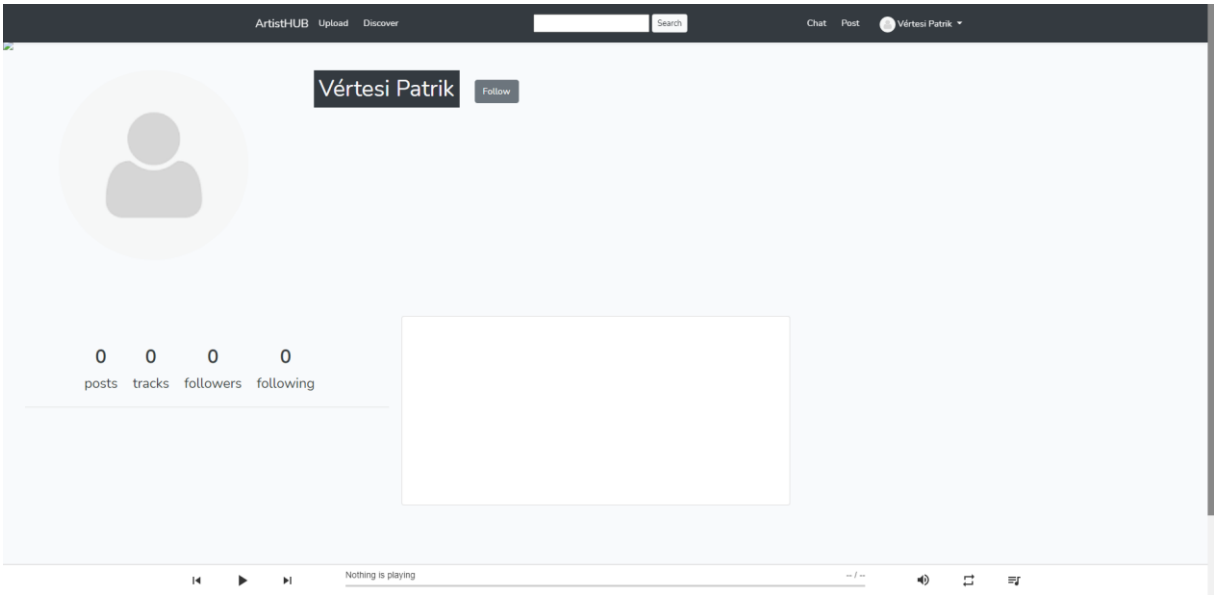
- A Home itt is a bejelentkezés utána kezdőoldalra navigál vissza.
- A Profile a saját profilunkat nyitja meg
- Az Edit alatt tudjuk a saját profilunkat szerkeszteni
- A Lougout-tal pedig kiléphetünk, és visszakerülünk a regisztrálás előtti kezdőoldalra.

## 7.4. Profil

Egy új profil létrehozása után ezt a képet látjuk amint megnyitjuk a sajátunkat. Jelenleg elég üres, hiszen nekünk kell felvinnünk a saját adatainkat, mint bármely közösségi oldalon. A felvitt adatokat természetesen később bármikor módosíthatjuk, viszont fontos megjegyezni, hogy

ezeket az adatokat bárki láthatja rólunk, így ennek tudatában érdemes ezeket a képeket, bejegyzéseket felvinni. A háttérben a profiles táblához kapunk egy alapértelmezett rekordot, aminek egyedül két rekordja van kitöltve, a saját ID, és az idegen kulcs, ami a user ID-je.

Minden más rekord nullázható, hiszen regisztráláskor nem kell például megadni a profilképet vagy a bannerképet. A következőkben egy profil feltöltésén fogok végig menni, majd felvázolom a működését az egyes elemeknek.



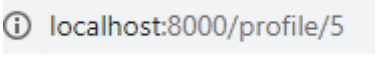

**11**

**16. ábra: Üres profil**

Kapunk egy alapértelmezett profilképet, amit természetesen később lecserélhetünk. A nagy üres rész a profilkép mögött a banner helye, amit hamarosan szintén beállíthatunk.

Természetesen a felhasználónk még nem rendelkezik se poszttal se zenével, nem követi őt senki és ő sem követ még senkit, ezért minden üres és minden nullázva van. A későbbiekben a profilkép alatti számlálók dinamikusan fognak nőni, ahhoz mérten, hogy mennyi feltöltött tartalma van. Ezt minden felhasználónál külön-külön a saját tartalmához igazítva láthatjuk, így, ha rámegyünk egy másik profilra ott más számok fognak megjelenni.

Minden profil kap egy ID-t, amit már korábban említettem, hogy a tábla egyetlen olyan rekordja, amit már az inicializálásnál megkapunk, a user tábla idegen kulcsa mellé. Mi ezt az újonnan létrehozott profilunkat az alábbi linken találhatjuk.



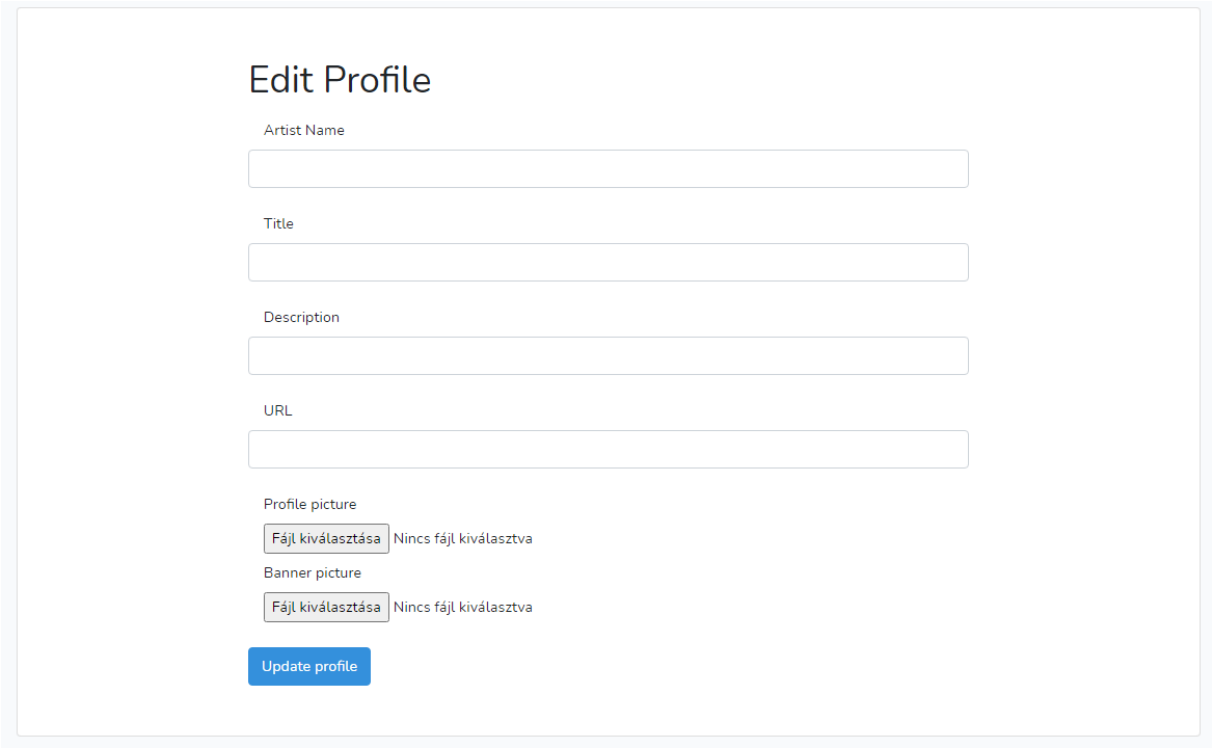

**17. ábra: Profil link**

Ez azt jelenti, hogy mi vagyunk az ötödik felhasználója az alkalmazásnak. Amennyiben valaki beírná, hogy profile/3, abban az esetben visszakapná az ahhoz az ID-hoz tartozó felhasználó profilját, adatait és posztjait. A háttérben ezt a web.php file biztosítja, ami várja a beadott ID-t,

tovább küldi a ProfileController kontrollerjébe, ott a profile funkció megkapja ezt az ID-t majd a visszaadja az ID-hoz tartozó profilt, amit meg is jelenít a böngésző ablakában.

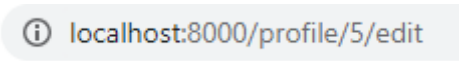
### 7.4.1. Profil szerkesztése

A lenyíló menüből az előbb említett Edit részre való kattintás után ez a lehetőség fogad bennünket.




**18. ábra: Profil szerkesztése**

Ennek a linkje szorosan kapcsolódik a profil ID-hoz, ugyanis az aktuális azonosított profilt akarjuk szerkeszteni, és nem másét. Minden esetben az éppen bejelentkezett felhasználó saját felületére navigál át minket. Ezt az általam megírt edit funkció biztosítja a profilecontrolleren belül, ahol annyi történik, hogy megkapja a bejelentkezett felhasználó ID-ját, és a hozzá tartozó Edit view-val tér vissza. A hozzátartozó link a következő módon néz ki:




**19. ábra: Szerkesztés linkje**

Itt adhatja meg a felhasználó a saját előadói nevét, magáról a leírást, a weboldalát, a profilképét valamint a bannerjét. A title-hez pedig megadhat egy mottót.

Amennyiben valamit üresen hagyunk az alkalmazás szól, hogy ki kellene tölteni azt a blokkot. Ez egy alapértelmezett Laravel funkció, ahol csak megadtam, hogy kötelező a mezőt megadni, így az automatikusan jelezni fog, hogyha üresen lett hagyva. Úgy gondoltam ez szükséges, mivel az egész alkalmazás más emberek felfedezéséről szól, így ki akartam küszöbölni azt, hogy legyen olyan user, akinek üres a profilja. Hagytam lehetőséget, hogy később, amennyiben

esetleg bővíteni szeretném a funkciót még több mezővel, így egy sor hozzáadásával már meg is jelenik. A probléma csak annyi, hogy minden ilyen módosítás alkalmazásakor újra kell migrálni az adatbázist, amely néhány adat elvesztésével járhat, de a Laravel tinker [40] alkalmazásán belül akár ellenőrizhetjük az adatokat manuálisan egy-egy ilyen parancs lefuttatása után.

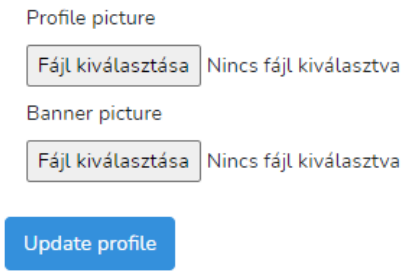
Amennyiben a felhasználó nem linket adott meg az URL-hez, értesítve lesznek róla, hogy ide mindenképpen linket kellene megadni. Ezzel könnyen ki lehet küszöbölni azt, hogy esetleg nem weboldalhoz tartozó információkat akar megadni egy felhasználó. Persze hogyha valakinek nincs weboldala, ezt a részt üresen hagyhatja, hiszen nem mindenki rendelkezik weboldallal. Az előadók elég nagy része csak évek után készíttet weboldalt, hiszen ez náluk jóformán csak információ közlésre szolgál.

Miután a felhasználó frissítette profiljának leírását, az azonnal meg is jelenik a profiljában. Ahogyan már korábban említettem, ez profilról profilra változhat, így dinamikus a tartalma. A profile view-ban annyi történik, hogy egyszerűen meg van hívva a HTML tagek közé a Laravel specifikus kiíratással, hogy a profilhoz tartozó tábla melyik rekordjához nyúljon vissza, és jelenítse meg, és ez persze a view-ba jelenleg átadott profile ID-tól függ, nem pedig a bejelentkezett felhasználó ID-tól.


**20. ábra: Előadó leírása**

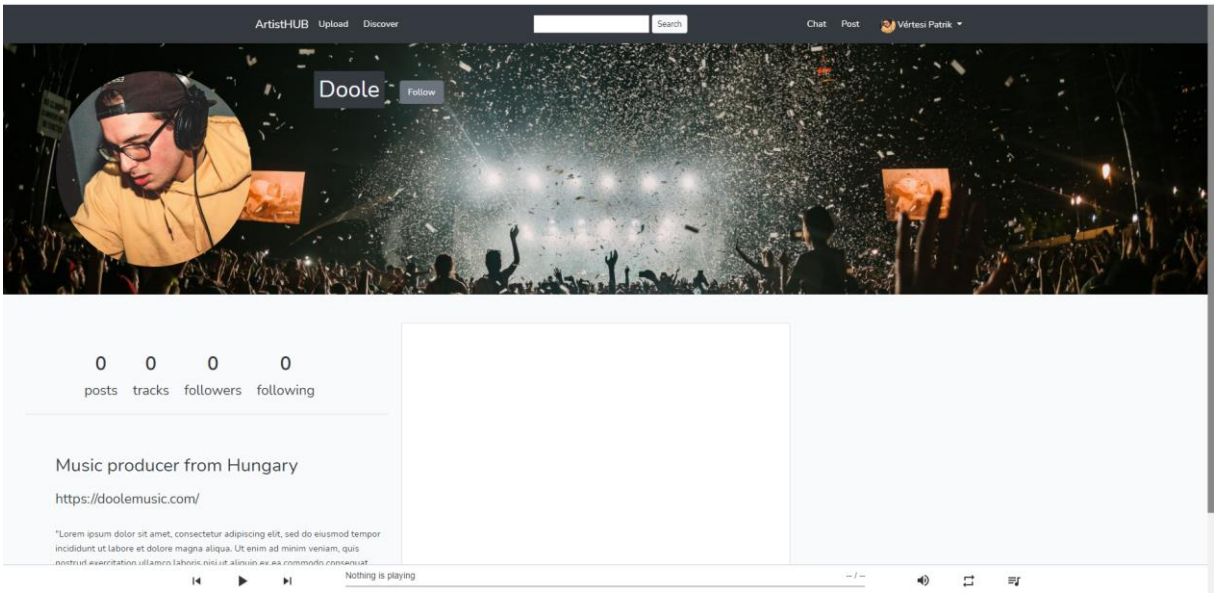
A következő, amit tehetünk ebben az ablakban, a képek frissítése. Az Update profile funkció lefutásakor megnézi, hogy a képhez van-e valamilyen fájl megadva, és ha igen, csak akkor fut le, ha az valamilyen kép formátum. Amennyiben bármi mást ad meg a felhasználó, hibát fog jelezni a rendszer. A háttérben ezt a request funkcióval valósítom meg. Majd ha átment a fájl az ellenőrzésen, a megadott útvonalba lementi a program a fájlt, és az Image:fit egy fix méretbe lekicsinyíti, hogy egységesek legyenek a képméretek az alkalmazásban. A profilkép és a banner is egy külön tömbbe mentésre kerül alapértelmezetten. Ezt követően egy array_merge

paranccsal a leírás tömbjével összeolvasztom a két tömböt, majd ezt a kombinált tömböt fogja a view megkapni, amely minden adatot tartalmaz, ami a view megjelenítéséhez szükséges.




**21. ábra: Kép feltöltés**

Miután megadta a felhasználó azokat a képeket, amiket fel akar tölteni magáról, csak rá kell nyomnia az update profile-ra, és megjelenik a profilban. Ezeket a képeket, adatokat természetesen ahogyan már korábban is említettem, bármikor lehet frissíteni. A funkció többször is felhasználható, és amennyiben új adatokat kap, felülírja a régieket. Ezt részletesen ismételten a Laravel tinker alkalmazásával lehet nyomon követni, esetleg a Laravel telescope-val, de jelenleg még nem telepítettem fel, ugyanis a tinkerrel pontosan annyit tudtam kommunikálni az alkalmazásommal, amennyire szükségem volt.




**22. ábra: Frissített profil**

A profilképet több helyen is felhasználja az alkalmazás, például a posztoknál is, az aktivitás jelzésénél, mostantól mindenhol ez a kép fog megjelenni, mint például itt is. A háttérben többször van felhasználva ugyanaz a funkcionalitás, amelyben az alkalmazás lekéri a felhasználóhoz tartozó profilt, és a hozzátartozó profileImage funkciót. Azért funkciót használtam, mert ez dönti el, hogy amennyiben nincsen megadott kép a profilban, úgy az alapértelmezett képet fogja felhasználni az alkalmazás, mivel, ha nem lenne ott semmi, úgy csak egy útvonal nem elérhető hibát kapnánk. Ami persze nem mutat szépen. Természetesen

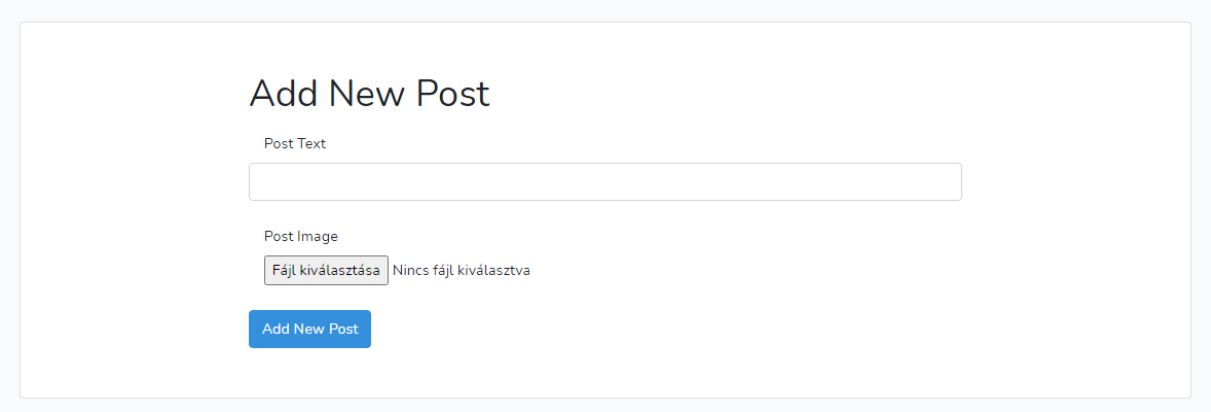
igyekszem a legtöbb ilyen hibát lekezelni. Az update metódus a Laravel ’patch’ funkcióját használja, amely felülírja a már meglévő tartalmat a táblában a funkció meghívásakor.

Az MVC programtervezési minta nagyon megkönnyíti az ehhez hasonló problémák kezelését. Ha nem ezt a programozási folyamatot használnám, már rég nagyon meg lennének keveredve a kódsorok, de így egy egyszerű, átlátható és konzisztens kód a végeredmény. Ezáltal a nagyobb méretű projekteknél sem kell attól félni, hogy hamar átláthatatlan lesz, egy rosszul felépített kód, hiszen a Laravel nem is engedi máshogyan a munkafolyamatot.

Az újra felhasznált kód és a tartalom dinamikus változása egy ekkora nagyságú projektben elengedhetetlen, hiszen nem lehetne minden felhasználóra külön-külön készíteni egy view-t, nagyon gyorsan elfogyna a rendelkezésre álló hely es nagyon átláthatatlan lenne a fájlok rendszere.

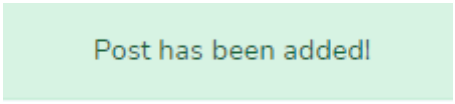
## 7.5. Posztolás

A korábban említett Post-ra kattintva adhatunk meg egy új posztot. A megjelenés nagyon hasonló az edit ablakhoz, hiszen a bootstrap alapértelmezett card osztályát használtam ennek a megjelenésnek a megvalósításához. Nagyon sok helyen használtam fel a bootstrap adta alapértelmezett lehetőségeket.




**23. ábra: Új bejegyzés**

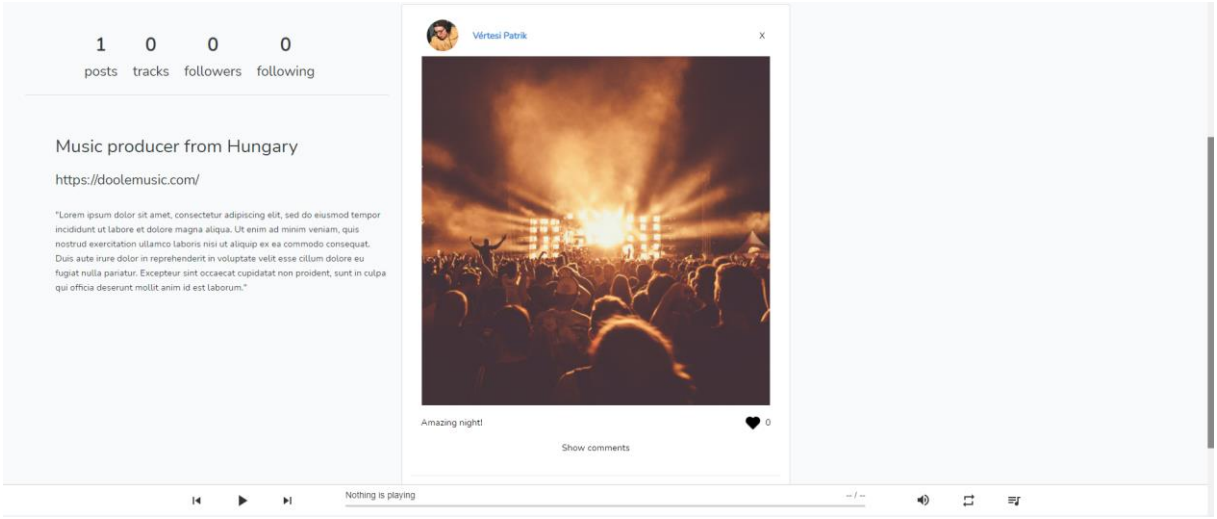
Szöveget, valamint képet tudunk hozzáadni egy posztunkhoz. Itt úgy gondoltam, hogy kötelező megadni a képet egy felhasználónak, mivel az oldal jelenleg eszerint van formázva. Ezért abban az esetben, hogyha egy felhasználó nem ad meg képi anyagot, úgy az alkalmazás figyelmezteti az Add new post gomb megnyomása után, hogy valamit elfelejtett. Ez a döntés kinézet miatt született, ugyanis jelenleg nincs még funkció arra, hogy amennyiben nincs kép a poszton belül, milyen formában jelenjen meg a poszt, ezt a következőkben részletesebben körbejárom majd, amikor azt tárgyalom, hogy egy külön poszt hogyan néz ki megnyitva a saját linkjén belül. Működésre nagyon hasonlóan működik, mint a profil szerkesztése. A beírt szöveg mentésre kerül a megfelelő helyre, valamint a kép request segítségével kerül elmentése a megfelelő mappába. Ezt a Laravel post metódusával teszem lehetővé.




**24. ábra: Sikeres poszt feltöltés**

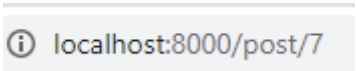
Amennyiben a poszt hozzáadása sikeres volt, az alkalmazás visszatér egy siker üzenettel. Sikertelen üzenet nem jelenik meg a profilban, ugyanis az esetben, hogyha egy poszt hozzáadása sikertelen volt, a felhasználó még a post pagen belül lesz értesítve a hozzáadás sikertelenségéről. Ez a megoldás azért született, mert így azonnal tudja korrigálni a hibáit, és nem kell vissza navigálnia a poszt hozzáadása oldalra, így el lehet kerülni a felesleges extra lépések megtételét.

A felhasználó által hozzáadott poszt most már megjelenik a profilban a posztok alatt. Valamint a posztok száma, ha jobban megnézzük növekszik egyel. Ez minden profilban a saját felhasználója által feladott posztok száma által növekszik. Ezt egészen egyszerűen a Laravel count metódusával oldottam meg, amely megszámolja, hogy hány darab létezik a megadott rekordból egyes felhasználónál. A poszt formázása is jól látszik itt. Felül van a felhasználó profilképe, valamint mellette a neve. Jobb oldalt található egy X, ami csak akkor jelenik meg, hogyha a bejegyzés a felhasználóhoz tartozik. Ez lehetőséget ad arra, hogy amennyiben már nincsen szükség a bejegyzésre, a felhasználó törölni tudja azt. Ez az adatbázisból törli szintén a hozzátartozó adatokat, mint a hozzászólások, valamint a kedvelések. Alul látható a bejegyzés szövege, valamint mellette a kedvelések száma.



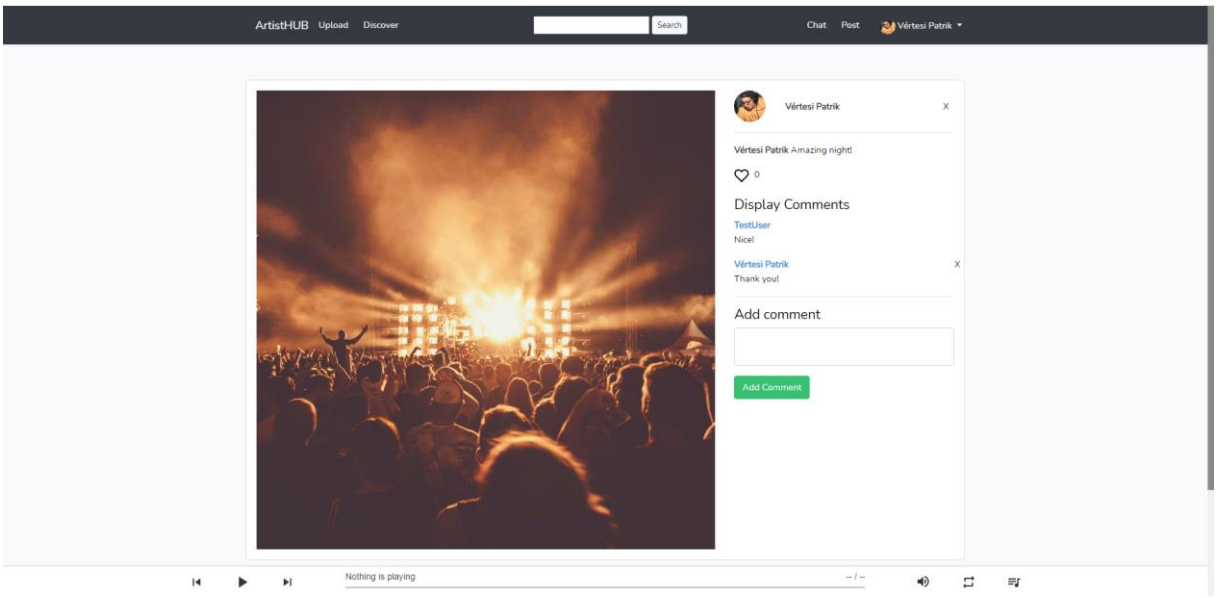

**25. ábra: Profil poszttal**

Egy posztot meg lehet jeleníteni önmagában is a post ID segítségével, amennyiben valaki rákattint a poszt képére a következő linkre fogja irányítani az alkalmazás.




**26. ábra: Post ID**

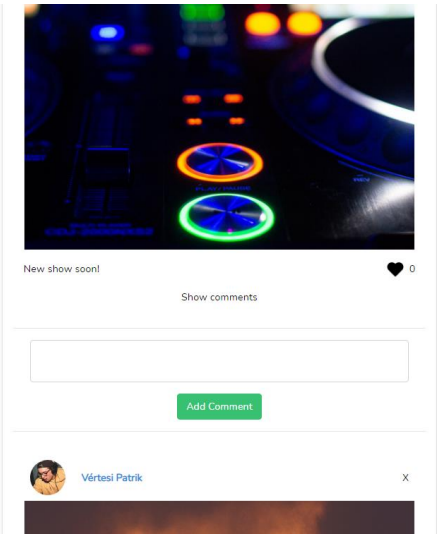
A szám természetesen az aktuális poszt ID-jától függ. Amint látjuk, ez nincsen hozzákötve a bejelentkezett felhasználóhoz, értelemszerűen a háttérben a link lekéri, hogy a poszt melyik felhasználóhoz tartozik. Így láthatjuk a kép mellett és a szöveg felett a felhasználó profilképét, és nevét, ugyanis a user nevű változót megkapja minden poszt, ami, mint láthatjuk azon felhasználót tartalmazza, aki a posztot feladta. Ezt fontos feltüntetni, hiszen így az, aki böngészi az alkalmazást, azonnal tudni fogja, hogy éppen a kedvenc előadója adta-e fel a posztot vagy más. A következőkben pedig kitérek arra, hogy hogyan néz ki maga a poszt, a saját ablakában amikor meg van nyitva.




**27. ábra: Poszt ablak**

Jelenleg ez a kép vár minket, hogy ha meg akarunk nézni egy különálló posztot. A komment szekció, valamint a kedvelések száma is látható itt is, ezen az ablakon, viszont ezekre később térnék ki egy külön fejezetben. Ránézésre természetesen nem tér el a profilon belül megjelent poszt formázásához képest, hiszen itt is a Bootstrap által használatra bocsájtott alapértelmezett osztályokat használtam fel. Amennyiben tetszik a kép, amit a posztnál látunk, és szeretnénk megnézni nagyobban, erre mindenképpen jó megoldás lehet akár már most is megnyitni a poszt ablakot, hiszen mint a példán látható képen, sokkal jobban mutat. Itt is a kép méretét a Laravel ’fit’ funkciójával oldottam meg, így nem vágja le a képet, csupán beleerőlteti egy négyzetbe, amit majd, ha esetleg később egy lightbox-al például szeretnénk megnyitni, vagy hogyha esetleg a kép megnyitása új lapon lehetőséget választanánk, úgy az eredeti képet kapnánk vissza, nem pedig egy levágott torzított képet.

Amennyiben a posztoló nevére kattintunk természetesen át leszünk irányítva a felhasználó profiljára. Ez is egy újra felhasznált kód korábbról, amit az átláthatóság és a hatékonyság miatt nagyon fontosnak találok, hogy minél több funkció többször is fel legyen használva.

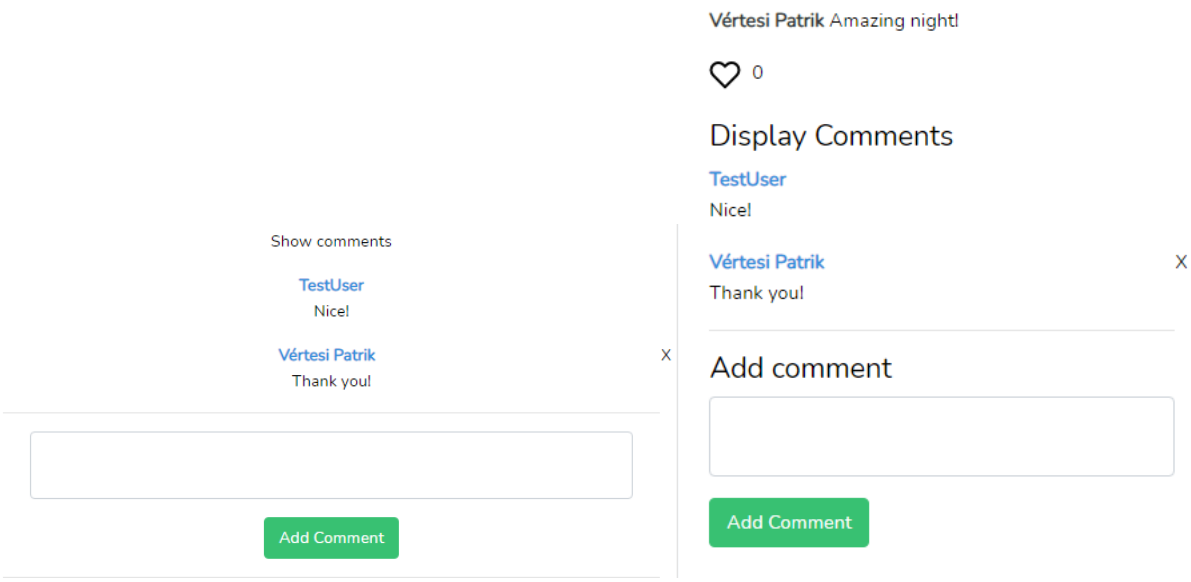



**28. ábra: Több poszt**

Ha a felhasználó hozzáad még egy posztot, természetesen a legutolsó mindig felűre kerül. Ezt a Laravel latest metódusával valósítom meg. Amely végig iterál a felhasználó posztjainak fenttartott mezőben a táblában, majd kilistázza mindet, de a legfrissebbet rakja előre. Nem csak egy közösségi oldalon, de bármilyen portálon is az emberek a legfrissebb posztokra kíváncsiak, így ennek a funkciónak a használatát evidensnek gondoltam. Szerencsére a Laravel nagyon sok közösségi oldalhoz használható beépített funkcionalitással érkezik.

## 7.6. Hozzászólások

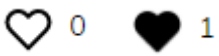
Minden bejegyzés alatt pedig a show comments megnyomásával láthatjuk a bejegyzéshez készült hozzászólásokat. Ezt a jQuery slideToggle() funkciójával valósítottam meg, és azért volt rá szükség, mert amennyiben rengeteg komment lenne egy-egy bejegyzéshez, úgy hatalmas hely lenne a bejegyzések között a feed-en. Az X itt is megjelenik, amennyiben a felhasználó a korábbi hozzászólását törölni szeretné. Ezen felül még hogyha maga a bejegyzés van megnyitva is láthatóak oldalt a hozzászólások. Minden felhasználó tudja kezelni a saját hozzászólásait. Mivel a hozzászólás azonosítójához vannak kapcsolva a hozzászólások, így minden bejegyzésnél láthatóak a hozzászólások, amelyek hozzá tartoznak. Látszik, hogy ki írta a hozzászólást, valamint a felhasználó profilját is megtudjuk nyitni a felhasználó névre való kattintás után.




**29. ábra: Hozzászólások**

## 7.7. Kedvelések

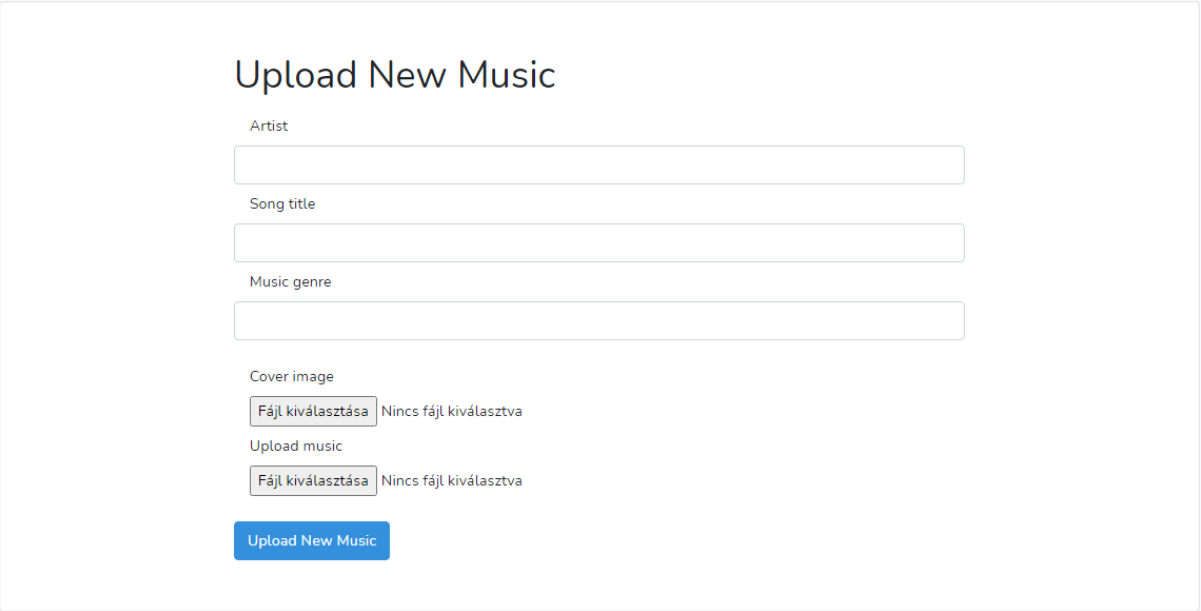
A kedvelések a hozzászólásokhoz hasonlóan működnek, valamint ugyanott találhatóak. A kapcsolatok is megegyeznek, annyi különbséggel, hogy a kedvelésekhez nem tartozik szöveg. Viszont az alkalmazás érzékeli, hogy a felhasználó már kedvelte-e az adott bejegyzést, vagy zenét, és aszerint mutatja a kedvel ikonját. Ha üres az ikon, még nem kedveltük a bejegyzést, amennyiben pedig teli szív van ott, úgy az már kedvelve van. Mellette található egy szám, amely a korábban már említett count metódussal összeszámolja, hogy az adott bejegyzést eddig hány felhasználó kedvelte. Egy bejegyzést vagy zenét természetesen csak egyszer enged kedvelni, ha már kedvelve van, csak visszavonni engedi a kedvelést. Kedvelni csak a bejegyzés ablakában tudunk.




**30. ábra: Kedvelések**

## 7.8. Zene felöltés

A korábban említett Upload gombra kattintva jutunk el a zene feltöltés részhez. Itt szintén látható, hogy ugyanazt a formátumot használom, mint a post page-en. Fontosnak tartom, hogy egy weboldalon a design egységes legyen, és habár ez még nem a végső design, próbáltam törekedni az egyszerű letisztult kinézetre, és persze a Bootstrap alapértelmezett osztályokat minél jobban előnyben részesíteni.

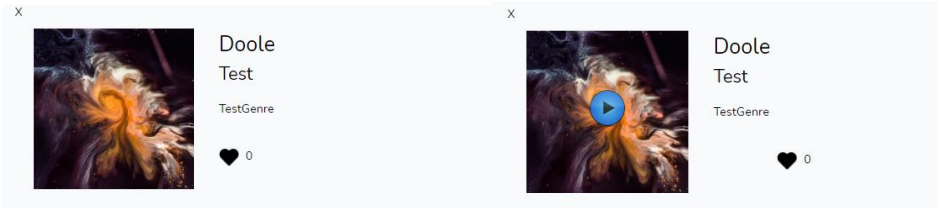



**31. ábra: Zene feltöltés**

Az alkalmazás vár egy előadónevet, hiszen egy felhasználó feltölthet más előadóktól való zenét vagy közös zenét is. Egy zenecímet, egy műfajt (ez a későbbiekben fontos lesz), valamint egy borítóképet, és magát az mp3 fájlt. Ez biztonságos, hiszen az alkalmazás szól, amennyiben nem egy audio fájlt töltöttünk fel, az pedig vírust nem tartalmazhat, legrosszabb esetben nem játssza le, de még ha letöltené egy felhasználó, akkor sem tudna egy rossz mp3 fájl kárt tenni más gépén.

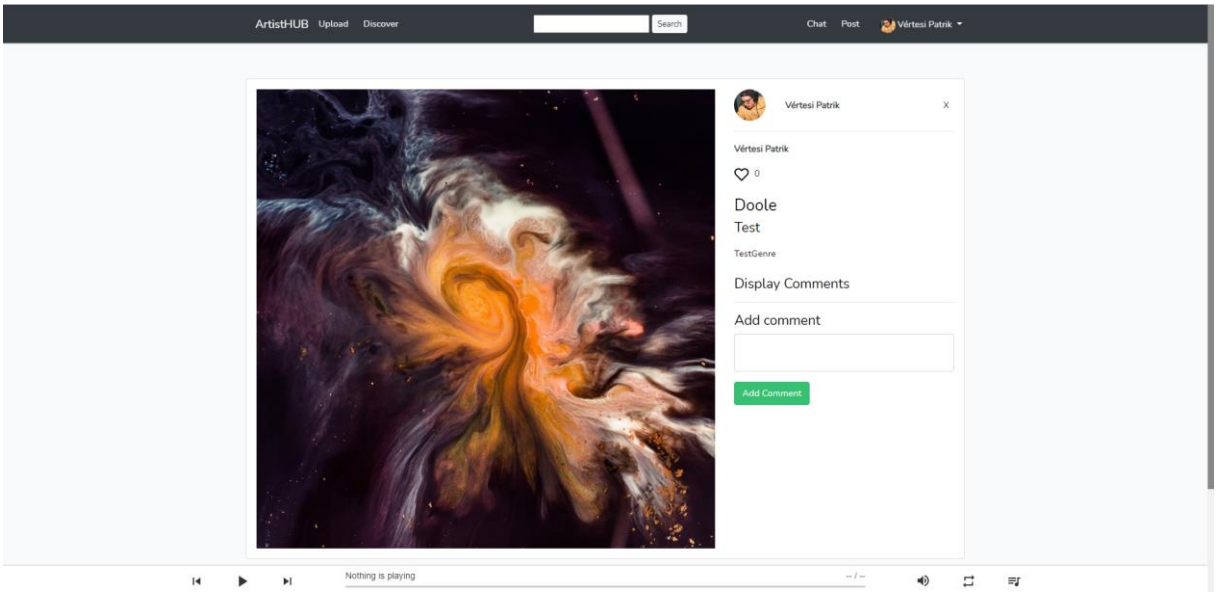

**32. ábra: Zene hozzáadása**

A program visszatér egy sikert jelző üzenettel, majd a zene megjelenik a profilunkon, és a track számláló pedig megnövekszik eggyel. Itt is szintén a count metódust használtam, és ugyanúgy kapunk egy siker üzenetet, mint a poszt hozzáadásánál, viszont azt az előbbiekben nem említettem, hogy ezt úgy valósítom meg, hogy a Laravel session funkcióját meghívom, amiben ez a sikeres üzenet blokk található. Ez pedig csak akkor fut le, amennyiben a sessionnek van üzenete, és azt pedig egy get metódussal lekérem, majd megjelenítem a profilon. Ezt egy alert-success bootstrap osztályban teszem meg.



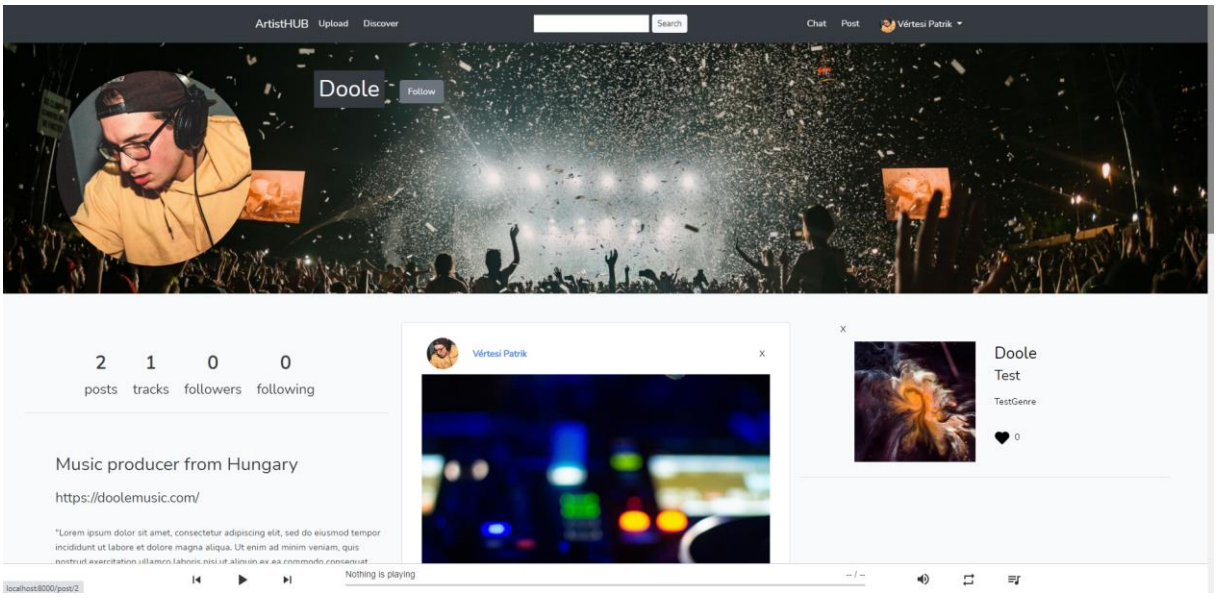

**33. ábra: Zene kilistázva**

Feltöltés után jobb oldalt meg fog jelenni ez az új blokk. Bal oldalt halálható az album borítója. Jobb oldalt az előadó neve (amit megadott a felhasználó feltöltéskor, nem pedig a saját profilneve). Alatta pedig a szám címe, műfaja, valamint a kedvelések száma. A profilnak ezen a részén a bejegyzésekhez hasonlóan, amennyiben több zeneszámot tölt fel egy adott felhasználó, úgy egyre több jelenik meg egymás alatt, mindig a legújabb legfelül, és a számláló is mindig növekszik, minden egyes zene hozzáadásánál. Látható itt is a törlés gomb, amely csak akkor látszik, ha egy felhasználó a saját maga által feltöltött zenét nézi. Ha törli, itt is töröl mindent, ami hozzá tartozik az adatbázisban, és a számláló is visszaesik egyel. Ahogyan pedig a képen is látható, amennyiben a felhasználó ráhúzza az egerét a borítóképre, úgy jelenik meg egy kis lejátszó gomb, amire ha rákattint, el fog indulni a lent látható zenelejátszón a zene.




**34. ábra: Zene aloldala**

A zenét megnyitva a következő oldal várja a felhasználót. Pontosan ugyanazt az elvet követi, mint a bejegyzések, annyi különbséggel, hogy itt nem a post ID hanem a music ID alapján nyitja meg a hozzátartozó aloldalt. Itt is látható a borító, amire, ha rákattintunk természetesen innen is elindul a zene, látható a posztoló, kedvelések, hozzászólások, valamint a szám előadója, címe és műfaja.




**35. ábra: Feltöltött profil**

Egy feltöltött profil a következőképpen néz ki: jelenleg ezt a felhasználót nem követi senki, és nem is követ senkit. Azt a következőkben szeretném vázolni, hogy ez hogyan is működik, de előtte még egy fontos dolog maradt hátra, hogy hogyan fog a zene lejátszása működni.

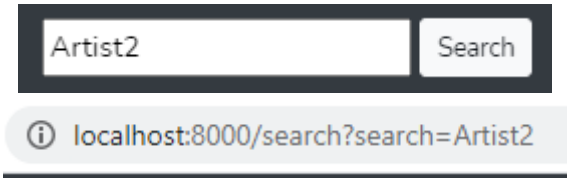
## 7.9. Zenelejátszás

Korábban már sokszor látható volt a képek alján. Miután rákattintott a felhasználó a lejátszani kívánt zenére, a Laravel kattintás eseményre átadja a zene helyét a lejátszót elindító funkcióba, amennyiben korábban már ott játszódott egy zene, azt kitörli onnan. Az előadó neve és a szám címe megjelenik a csúszka felett, valamint a hangerő szabályzásra, és a zene ismétlésére is lehetőség van.


**36. ábra: Zenelejátszó**

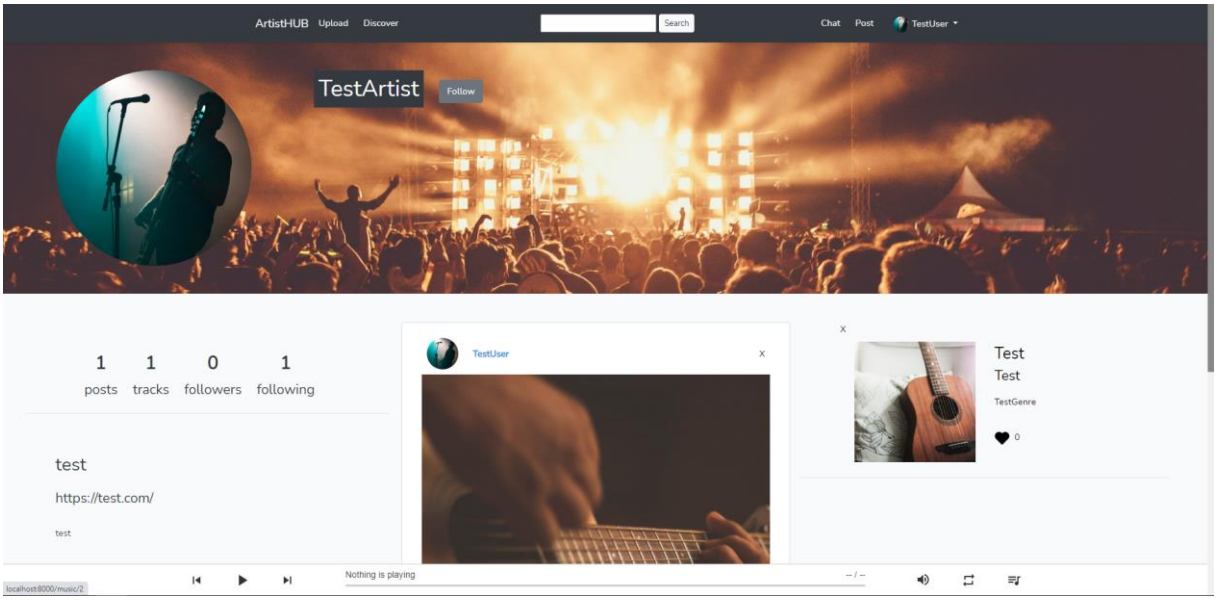
## 7.10. Felhasználó keresés

Amennyiben egy felhasználó szeretne más felhasználót megtalálni, nincs semmi más dolga, mint az előadó nevét beírni a keresőbe, és amennyiben van találat, automatikusan át lesz irányítva a profiljára. Ezt egy saját kereső funkcióval valósítom meg, amely egy form segítségével bekéri a mezőben található artist nevet, majd átadja a profilescontroller-ben található funkciónak egy request változóba mentve, és végeredményben hogyha van egyezés a beadott szöveg, és a tábla egy rekordja között, annak az ID-ját visszaadja a nézetbe. Rákereshetünk a felhasználó igazi nevére is, amennyiben nincs előadó neve, de hogyha van, úgy az alapján is meg tudjuk találni.




**37. ábra: Kereső funkció**

És meg is találta az előadó profilját, amit most nem az a szokásos ID alapú linkel kapunk meg, hanem a kereső paraméter eredményeként kapjuk vissza. A paraméterként átadott link ugyanúgy működik, mintha csak az ID alapján nyitnám meg a profilt, mivel a funkcióba meg van írva, hogy az ID alapján adja vissza a szóban forgó profilt, így attól függetlenül, hogy más a link, funkcionalitás szerint megegyezik. Amennyiben adott meg adatokat, itt láthatnánk azokat. Elolvashatjuk a leírását, láthatjuk a zenéit, hozzászólhatunk a tartalmához. Valamint persze be is követhetjük, amely funkciót a következőkbe fogom tárgyalni.




**38. ábra: Másik felhasználó profilja**

## 7.11. Követés

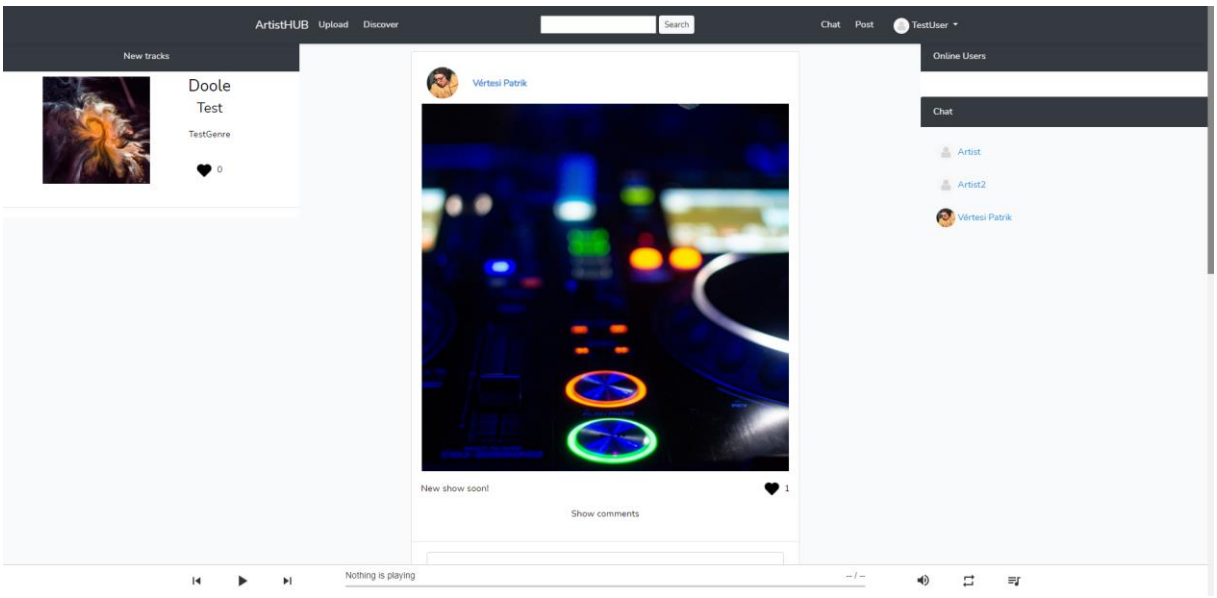
A követéshez nincsen más dolga a felhasználónak, mint a Follow gombra nyomni, és innentől látni fogja a bekövetett felhasználó legújabb bejegyzéseit és zenéit a főoldalon.

A Follow gomb benyomása után természetesen automatikusan átvált Unfollow-ra, indikálva azt, hogy már követjük az alábbi felhasználót, és ha már nem vagyunk rá kíváncsiak, akár ki is követhetnénk. Ezt a funkciót egy Vue.js komponenssel valósítom meg, amely talán az egyetlen olyan funkció ebben a projektben, amely ezt a keretrendszert használja, de a megvalósítása jelentősen egyszerűbb volt ebben a formában.


**39. ábra: Ki és be követés**

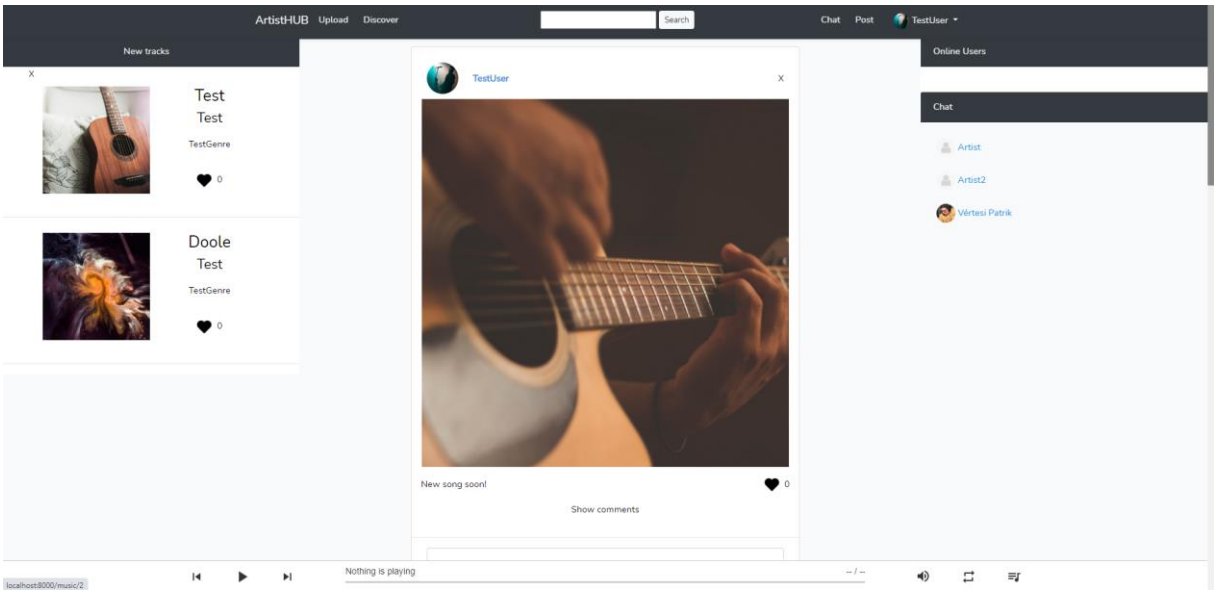
Az oldal frissítése után láthatjuk, hogy az előbbi 0 követő, most már feljebb lépett 1 re, aki ez esetben mi vagyunk. A saját profilon természetesen a following most eggyel megnőtt.

Amennyiben visszamegyünk a kezdőoldalra, most már nem üres, hiszen van egy felhasználó, akit követünk, ezáltal látjuk a tartalmat, amelyet megoszt. Demonstráció céljából átléptem egy teszt felhasználó profiljába, és az előző felhasználó profilját követtem be, mivel az már tartalommal feltöltött, így látható, hogy a korábban beállított tartalmakat fogja kilistázni a kezdőoldalra.




**40. ábra: Hírfolyam**

Hogyha kikövetnénk ezt a felhasználót természetesen eltűnne a tartalma innen. A home nézet funkciói úgy működnek, hogy a homecontroller nevű controllerben definiálom a szükséges változókat, majd azt visszaadom a nézetnek, és az összes felhasználón végig megyek. Majd azoknak a felhasználóknak a zenéit és posztjait fogja kilistázni, amelyeket követek.




**41. ábra: Hírfolyam több követett felhasználóval**

Ha bekövetünk még egy felhasználót, annak a profilja is megjelenik a kezdő oldalunkon, és így tovább, mindig a legutóbbi fog megjelenni nekünk. Ennek az opciónak a beállításához is a Laravel latest metódusát használtam.

## 7.12. Online felhasználók

A kezdőlap utolsóként következő tartalma pedig az aktív felhasználók rész. Ez a rész a követéstől független, az összes felhasználót láthatjuk, aki fent van, és meg is nyithatjuk rájuk kattintva a profiljukat.

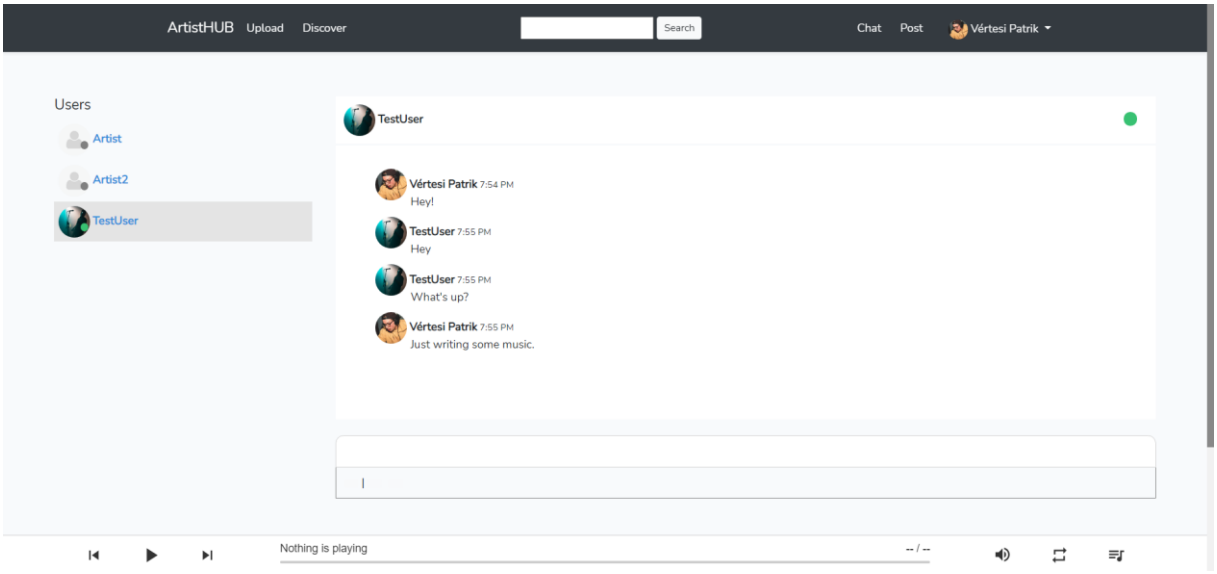
Nem a bejelentkezéstől függ az, hogy egy profil megjelenik itt, hanem az utolsó aktivitástól számított egy percig számít online-nak egy felhasználó. Tehát ha folyamatosan használjuk az alkalmazást, online vagyunk, amennyiben nem jelentkezünk ki, de nem kattintunk 10 percig, már nem leszünk aktívak. Ez később a chat felület implementálásakor lesz nagyon hasznos, hiszen ott sem az a lényeg, hogy ki van bejelentkezve, hanem hogy ki használja éppen az alkalmazást, azaz ki elérhető.


**42. ábra: Elérhető felhasználók**

## 7.13. Chat

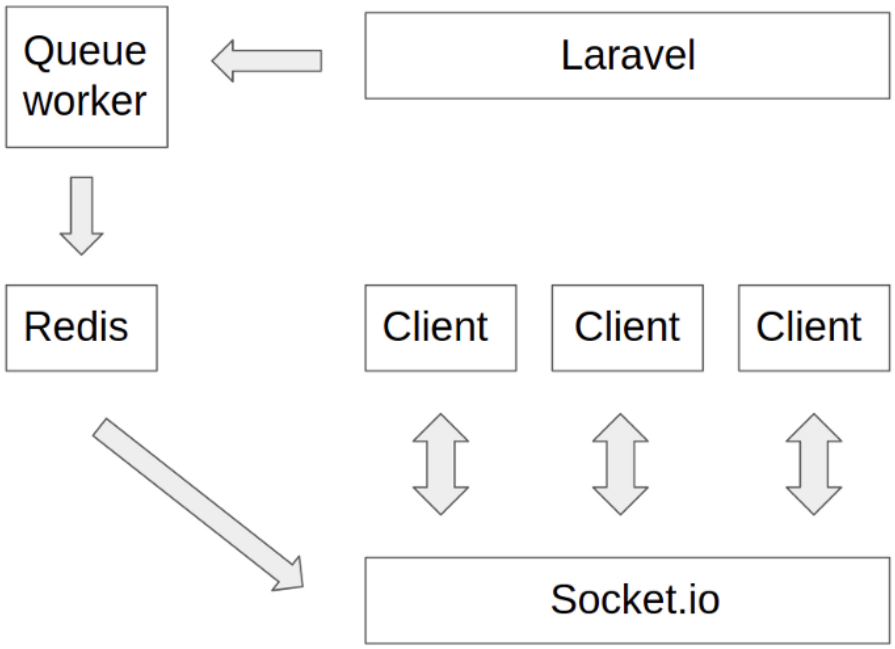
Az aktív felhasználók alatt található a chat rész, itt láthatunk minden felhasználót, akikkel beszélgetést tudunk kezdeményezni. Ezt meg tudjuk nyitni a korábban említett navbarban található Chat funkció segítségével, vagy itt. Ha a kezdőlapról nyitja meg egy felhasználó, egy

másik felhasználóval történő beszélgetését, úgy rögtön arra a beszélgető ablakra navigál az alkalmazás. Amennyiben a chat fülről nyitja meg, úgy egy általános chat ablakot kap csak.




**43. ábra: Chat ablak**

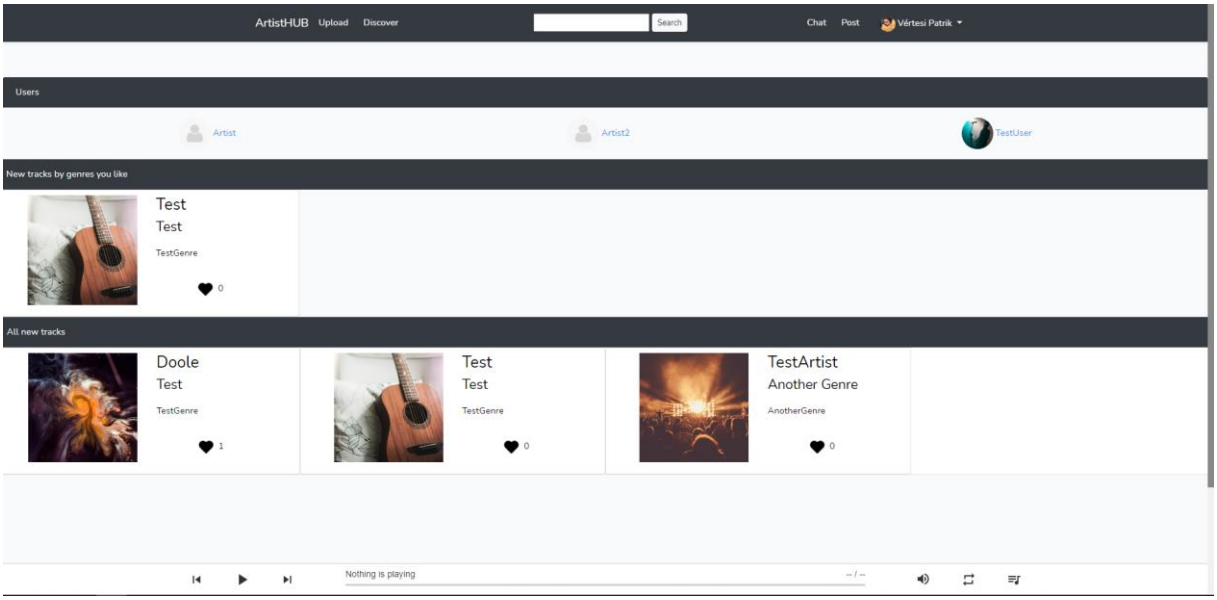
A Chat ablakban található a talán legkomplexebb része a projektnek. Mivel azonnali üzenetváltásról van szó, így a Socket.io-t használtam, ami websocketek [41] használatával duplex kapcsolatot hoznak létre a kliens és a szerver között.




**44. ábra: Közvetítési architektúra[42]**

Működése a következő: amikor egy Laravel esemény elküldésre kerül, ami implementálja a ShouldBroadcast interfacet [43], a parancs bekerül a queue worker [44] sorába. Ez azt jelenti, hogy egy másik php funkció fogja közvetíteni a parancsot. Esetünkben a közvetítést a Redis közvetítő vezérlője fogja intézni. Ez fogja közzétenni az üzenetet a publish parancsával a csatornára, ahol az üzenetet minden feliratkozott felhasználó meg fogja kapni. Itt fogja az alkalmazás kihasználni a Socket.io és a Node.js adta lehetőségeket. A Node.js köti össze a Redist a Socket.io-val. Végül pedig a Socket.io fogja a Redis-től megkapott üzenetet továbbítani a kliens oldalra, hogy azt a felhasználó megkapja. Ez az egész folyamat a böngésző frissítése nélkül működik, így lehetővé téve az azonnali üzenet váltást az alkalmazásban.

## 7.14. Felfedezés




**45. ábra: Felfedezés ablak**

A Discover fülön láthatóak a felhasználók, akiket követhetünk, valamint zenei ajánlások.

A felhasználóknál, valamint a legújabb zenéknél az összes kilistázásra kerül, azonban a stílus szerinti ajánlások csak bizonyos zenéket fog mutatni. Minden zenének van egy stílusa, amit már korábban említettem, a feltöltésnél lehet megadni. Amennyiben a felhasználó kedvel egy zenét, jelen esetben a képen látható TestGenre stílusú zenét kedveli, úgy csak azok a zenék kerülnek kilistázásra, amelyek ugyan ehhez a stílushoz tartoznak. A funkció a háttérben megnézi a felhasználó kedvelt zenéit, összesíti a kedvelt műfajokat, majd véletlenszerűen visszaad ezekből a műfajokból olyan zenéket, amelyet a felhasználó még nem kedvelt.

# 8. Tesztelés

Minden szoftver beüzemelése előtt a legfontosabb lépés a tesztelés. Ez a lépés képes felfedni az alkalmazás hibáit, de azt nem, hogy nincsen hiba. Azonban mindenképpen növeli a szoftver minőségét és megbízhatóságát. [45]

A tesztelés szintjei a következők:

- komponens teszt
- integrációs teszt
- rendszerteszt
- átvételi teszt

A komponens tesztben az egyes elemeket teszteljük külön-külön. Az integrációs tesztben az egyes elemek együttes működését tudjuk letesztelni. A rendszertesztben az egész rendszert, valamint az átvételi tesztben a kész rendszer kerül teszt alá.

A következő tesztelési technikák léteznek:

- Feketedobozos
- Fehérdobozos

A feketedobozos technikában nem látható a forráskód, csak a specifikáció alapján dolgoznak. Én fehérdobozos tesztelést alkalmaztam, amelyben használható a forráskód is, így hatékonyabb tesztelést tudtam végrehajtani és azonnal javítani a hibákat.

Fontos megjegyezni ezeket a teszteket általában csapatokat végzik, de mivel itt egyedül dolgoztam, előfordulhatnak még hibák.

Azt is érdemes figyelembe venni, hogy ezekre a tesztekre mennyi a rendelkezésre álló idő, valamint a kapacitás. Esetemben leginkább az általános felhasználói szolgáltatásokhoz kapcsolódó hibákat próbáltam kiküszöbölni, mint hogy esetleg a követés nem megfelelően működik, esetleg rosszul jelenik meg egy bejegyzés, vagy a felhasználó nem a megfelelő tartalmat látja. A rendkívül szélsőséges esetekre nem fordítottam annyi figyelmet.

Az első három tesztelési szintet fejlesztői tesztelésnek hívják, míg az átvételi tesztet a felhasználók tesztelik. Az én esetemben csak az első három tesztelési fajtát alkalmaztam.

Külön-külön teszteltem a keresőt, a követés funkciót, a bejegyzés létrehozását, zenefeltöltést, és az összes lehetséges funkciót, amely előfordul az alkalmazásban. Fontos volt azt is tesztelni, hogy az alkalmazás jól kezeli-e, hogyha esetleg nem audio fájlt, vagy képet ad be egy felhasználó. Így az ebből adódó hibákat, valamint legfőképpen a vírusok megjelenését ki tudtam küszöbölni. Különös figyelmet kellett arra fordítanom, hogy egy felhasználó ne tudja szerkeszteni másik felhasználó profilját. Például volt engedély, ami nem volt megfelelően kezelve, így hogyha az URL-ből át írtam a jelenlegi felhasználó ID-ját az edit ablakban, úgy lehetőségem nyílt egy másik felhasználó profiljának szerkesztésére. Természetesen ezt azonnal ki kellett küszöbölni.

Ilyen és ehhez hasonló hibák kiküszöbölése miatt nagyon fontos a tesztelés. Habár egy személyes fejlesztői és tesztelői munkát végeztem, igyekeztem szem előtt tartani a tesztelés különböző fokozatait, így például egy bejegyzés létrehozásánál figyelembe venni, hogy először a funkció egyáltalán elmenti-e a tartalmat. Majd később, hogy az megfelelően jelenik-e meg, így ez a két komponens együtt megfelelően fut-e le. Valamint, hogy később az egész rendszer teljes egészében működik-e.

# 9. Összefoglalás és további tervek, tovább lepési lehetőségek

Összefoglalva a fent leírtakat, egy általam kitalált web applikáció fejlesztését mutattam be, amely a feltörekvő producerek/zenészek karrier építését és kapcsolataik kiépítését segítené elő. Bemutattam az ehhez felhasznált technológiákat, az adatbázis sémáját és funkciókat.

Ez egy közösségi oldal, amely rengeteg általános ilyen témakörben elvárható funkcióval és résszel rendelkezik, mint például a felhasználok követése, keresése, hírfolyam, profil megtekintése, beszélgetés. Az én alkalmazásomban ezen felül még található zene feltöltési lehetőség, ugyanis leginkább erre a hangsúlyra építettem az alkalmazást. Fontosnak tartom a feltörekvő előadók közötti kapcsolattartást, és a munkájuk segítését. Bemutattam ezeknek a funkcióknak a működését, kitértem a technológiák sajátosságaira, valamint azok használatának menetére.

Elektronikus zenei producer hobbimból adódóan több portálon is jelen vagyok a karrierem építése érdekében, ezért szerettem volna ezt az applikációt elkészíteni, hogy ezzel is könnyítsem a zenészek karrierépítését. A modern kor nagyon sok lehetőséget ajánl a feltörekvő zenészeknek, sokkal egyszerűbb feltörni manapság, mint 30-40 évvel ezelőtt, viszont még így is maradtak kiaknázatlan lehetőségek a piacon.

A dolgozatom kezdete óta bővítettem a témával kapcsolatos tudásom, és ez biztosan hasznos lesz a jövőben a webfejlesztő pályámon.

A projekt folytatásában tervezek még egy értesítési rendszert kialakítani, történetek hozzáadásának lehetőségét, amik olyan bejegyzések, amik csak 1 napig megtekinthetőek. Tervezem ezen felül még a zenelejátszó funkció bővítését, valamint a fájl megosztás lehetőségét.

Továbbá a frontend továbbfejlesztése a működő képes oldalon túl is, mivel egy ilyen közösségi oldalt a végtelenségig lehet fejleszteni.

# 10. Ábrajegyzék

# 11. Irodalomjegyzék


[1] Devdocs HTML
https://devdocs.io/html/ (Utolsó megtekintés: 2021. 04. 05.)

[2] Blade Template Engine
https://laravel.com/docs/8.x/blade (Utolsó megtekintés: 2021. 04. 24)

[3] Laravel
Laravel: Up & Running: A Framework for Building Modern PHP Apps 2nd Edition, Kindle Edition by Matt Stauffer

[4] PHP
https://www.php.net/manual/en/intro-whatcando.php (Utolsó megtekintés: 2021. 04. 06.)

[5] Devdocs CSS
https://devdocs.io/css/ (Utolsó megtekintés: 2021. 04. 05.)

[6] XML
http://www.w3c.hu/forditasok/XML_10_pontban.html (Utolsó megtekintés: 2021. 05. 05.)

[7] W3C
https://www.w3.org (Utolsó megtekintés: 2021. 05. 05.)

[8] Bootstrap
https://getbootstrap.com/docs/5.0/getting-started/introduction/ (Utolsó megtekintés: 2021. 05. 05.)

[9] JavaScript
https://developer.mozilla.org/en-US/docs/Web/JavaScript?retiredLocale=hu/ (Utolsó megtekintés: 2021. 05. 05.)

[10] NPM
https://www.npmjs.com/ (Utolsó megtekintés: 2021. 05. 05.)

[11] V8
https://v8.dev/ (Utolsó megtekintés: 2021. 05. 05.)

[12] Chromium
https://www.chromium.org/ (Utolsó megtekintés: 2021. 05. 05.)

[13] Document Object Model
https://developer.mozilla.org/en-US/docs/Glossary/DOM (Utolsó megtekintés: 2021. 05. 05.)

[14] jQuery
https://jquery.com/ (Utolsó megtekintés: 2021. 05. 05.)

[15] AJAX
https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX/Getting_Started (Utolsó megtekintés: 2021. 05. 05.)

[16] Socket.IO
https://socket.io/ (Utolsó megtekintés: 2021. 05. 05.)

[17] Node.js
http://www.inf.u-szeged.hu/~tarib/javascript/nodejs.htm (Utolsó megtekintés: 2021. 04. 06.)

[18] Vue JS
https://vuejs.org/v2/guide/ (Utolsó megtekintés: 2021. 04. 06.)

[19] C++
https://www.cplusplus.com/ (Utolsó megtekintés: 2021. 05. 05.)

[20] GNU
http://www.gnu.hu/gpl.html (Utolsó megtekintés: 2021. 05. 05.)

[21] Composer
https://getcomposer.org/doc/(Utolsó megtekintés: 2021. 05. 05.)

[22] Ruby
https://www.ruby-lang.org/en (Utolsó megtekintés: 2021. 05. 05.)

[23] SQL
https://www.dataquest.io/blog/sql-basics (Utolsó megtekintés: 2021. 05. 05.)

[24] SQLite
https://www.sqlite.org/index.html (Utolsó megtekintés: 2021. 04. 06.)

[25] C
https://devdocs.io/c/ (Utolsó megtekintés: 2021. 05. 05.)

[26] Redis
https://redis.io/ (Utolsó megtekintés: 2021. 04. 06.)

[27] Git
http://math.bme.hu/~balazs/git/gitcml.html (Utolsó megtekintés: 2021. 05. 05.)

[28] SQL Injection
https://portswigger.net/web-security/sql-injection (Utolsó megtekintés: 2021. 05. 05.)

[29] MySQL
https://www.mysql.com/ (Utolsó megtekintés: 2021. 05. 05.)

[30] PhpStorm
https://www.jetbrains.com/phpstorm/ (Utolsó megtekintés: 2021. 04. 24.)

[31] Jetbrains
https://www.jetbrains.com/ (Utolsó megtekintés: 2021. 05. 05.)

[32] GitHub
https://github.com/ (Utolsó megtekintés: 2021. 05. 05.)

[33] Figma
https://www.figma.com/ (Utolsó megtekintés: 2021. 05. 05.)

[34] OAtuh
https://oauth.net/2/ (Utolsó megtekintés: 2021. 05. 05.)

[35] Adatbázis kapcsolatok
http://www.sze.hu/~szorenyi/sz03/htm/doc/kapcsola.htm (Utolsó megtekintés: 2021. 04. 14.)

[36] MySQL Workbench
https://www.mysql.com/products/workbench/ (Utolsó megtekintés: 2021. 05. 05.)

[37] Google
http://www.google.com (Utolsó megtekintés: 2021. 05. 05.)

[38] Facebook
http://www.facebook.com (Utolsó megtekintés: 2021. 05. 05.)

[39] Eloquent model
https://laravel.com/docs/8.x/eloquent(Utolsó megtekintés: 2021. 05. 05.)

[40] Laravel tinker
https://laravel.com/docs/8.x/artisan (Utolsó megtekintés: 2021. 05. 05.)

[41] Websocket
https://developer.mozilla.org/en-US/docs/Web/API/WebSocket (Utolsó megtekintés: 2021. 05. 05.)

[42] Laravel Broadcasting with Redis and Socket.IO
https://medium.com/@jan.kulma/laravel-broadcasting-with-redis-and-socket-io-51ce2660633d (Utolsó megtekintés: 2021. 04. 17.)

[43] ShouldBroadcast
https://laravel.com/docs/8.x/broadcasting (Utolsó megtekintés: 2021. 05. 05.)

[44] Queue Worker
https://laravel.com/docs/8.x/queues (Utolsó megtekintés: 2021. 05. 05.)

[45] Tesztelés
http://aries.ektf.hu/~gkusper/SzoftverTeszteles.pdf (Utolsó megtekintés: 2021. 04. 15.)