Movies web application implementation using laravel framework

Kapas Zoltan [a]

[a] *Máv Szolgáltató Központ Zrt., Könyves Kálmán körút 54-60, Budapest, 1087, Hungary, zoltan.kapas25@gmail.com*

Absztrakt

A cikkben bemutatásra kerül egy internetes alkalmazás, mely lehetővé teszi a felhasználók számára, hogy egy dedikált oldalon értékeljék és kedveljék kedvenc filmjeiket. Emellett lehetőségük van megosztani gondolataikat a filmekről egy hozzászólás szekcióban, ahol más felhasználókkal is interakcióba léphetnek. Az alkalmazás alapját a kiváló Laravel PHP keretrendszer képezi, míg a felhasználói felület kialakításához a Bootstrap CSS keretrendszert használták fel. Ennek köszönhetően a weboldal kitűnően működik és jól megjelenik különféle képernyőméreteken és eszközökön. A filmek részletei egy REST API-alapú rendszer segítségével kerülnek megjelenítésre az oldalon. A cikk első szakaszában részletesen bemutatásra kerül az alkalmazás megvalósítása, kiemelve a Laravel keretrendszer és a Bootstrap CSS keretrendszer használatának lényeges szempontjait. Kitérés történik a felhasználók kedvenc filmjeinek értékelési és kedvelési funkcióira, valamint a hozzászólások lehetőségére is. A következő részben a tesztelés fázisai kerülnek bemutatásra, kiemelve az alkalmazás különböző aspektusainak ellenőrzését és validációját. A tesztek fontosságára és az alkalmazás stabilitásának biztosítására is kitér a tanulmány. A cikk utolsó szakaszában pedig a továbbfejlesztési lehetőségekről esik szó. Kiemelésre kerülnek azok a területek, ahol az alkalmazás még tovább finomítható vagy bővíthető. Ezek lehetnek például új funkciók bevezetése, a felhasználói élmény javítása vagy a rendszer teljesítményének optimalizálása. Összességében a cikk egy izgalmas internetes alkalmazást mutat be, amely a filmrajongók számára nyújt lehetőséget a filmek értékelésére, kedvelésére és az ezekkel kapcsolatos interakcióra. A technológiai háttér részletes bemutatása és a tesztelés fontosságának hangsúlyozása mellett a továbbfejlesztési irányokról szóló gondolatok is hozzájárulnak az olvasók mélyebb megértéséhez.

Kulcsszavak: programozás; Laravel; REST API

Filmek webalkalmazás megvalósítása laravel keretrendszerrel

Zoltan Kapas [a]

[a] *Máv Szolgáltató Központ Zrt., Könyves Kálmán körút 54-60, Budapest, 1087, Hungary, zoltan.kapas25@gmail.com*

Abstract

This article presents a web application that allows users to rate and like their favorite movies on a dedicated page. They also have the opportunity to share their thoughts about the movies in a comment section where they can interact with other users. The application is based on the excellent Laravel PHP framework, while the Bootstrap CSS framework was used to create the user interface. Thanks to this, the website works perfectly and displays well on various screen sizes and devices. Movie details are displayed on the site using a REST API-based system. In the first section of the article, the implementation of the application is presented in detail, highlighting the essential aspects of using the Laravel framework and the Bootstrap CSS framework. The evaluation and liking functions of the users' favorite movies, as well as the possibility of comments, are addressed. In the next section, the phases of testing are presented, highlighting the verification and validation of different aspects of the application.

The study also covers the importance of tests and ensuring the stability of the application. In the last section of the article, further development opportunities are discussed. Areas where the application can be further refined or expanded are highlighted. These can be, for example, introducing new functions, improving the user experience or optimizing system performance. Overall, the article presents an exciting web application that provides movie fans with the ability to rate, like, and interact with movies. In addition to the detailed presentation of the technological background and the emphasis on the importance of testing, thoughts on further development directions also contribute to a deeper understanding of the readers.

Keywords: programming; Laravel; REST API

1. Bevezető

A téma kiválasztása azzal indokolható, hogy egy webes filmes alkalmazás fejlesztése történjen, amely kiemelkedő aktualitást kap a filmkészítés és a moziba járás növekvő népszerűsége miatt. A folyamatosan bővülő filmkínálat követése kihívást jelent, és gyakran előfordul, hogy már látott filmek felismerése nehézséget okoz. Ráadásul az egyes filmekre vonatkozó vélemények és értékelések könnyen feledésbe merülnek. Ezért elengedhetetlen egy olyan online platform létrehozása, ahol az emberek megoszthatják gondolataikat, értékeléseiket a filmekkel kapcsolatban, legyen szó pontozásról vagy szöveges visszajelzésekről. Az alkalmazás célközönségét főként azok alkotják, akik rendszeresen néznek filmeket, akár napi vagy heti szinten, és gyakran látogatják a mozikat. A cikk célja az említett webes alkalmazás fejlesztési folyamatának érthető bemutatása.

Az internet már a legtöbb ember mindennapi életének része, és szinte mindenki rendelkezik internet-hozzáféréssel vagy olyan eszközzel, amely lehetővé teszi az internet használatát, mint például okostelefonok, tabletek, laptopok vagy számítógépek. Az online térben bankügyeket intézünk, híreket olvasunk, videókat nézünk és kapcsolatot tartunk az ismerőseinkkel. Ezért jött az ötlet, hogy az internetet olyan platformként használjuk fel, ahol maga a program elérhetővé válik, hiszen a weboldalak könnyen hozzáférhetők szinte mindenféle internet-hozzáféréssel rendelkező eszközről, ami rendelkezik egy böngészővel.

A fejlesztési folyamat során a választás a Laravel M. Bean (2015) keretrendszerre esett, ami egy ingyenes, nyílt forráskódú PHP keretrendszer R. Nixon (2014), amely könnyen kezelhető és számos előre elkészített, hasznos funkciót kínál. Emellett a Bootstrap 4 keretrendszer is beépítésre került, ami egy ingyenes front-end eszköz, és az HTML, CSS és JavaScript nyelveket használja. J. Fielding (2014) Az adatbázis-kezeléshez pedig a MySQL J. Fielding (2014) adatbázis-kezelő rendszer került választásra, mivel ez az interneten egyik legelterjedtebb adatbázis-kezelő rendszer, és grafikus felülettel rendelkezik, amely könnyen kezelhető.

A cikk a *Filmek webalkalmazás tervezése laravel keretrendszerrel* című tanulmány folytatása, amely elsősorban az implementálási fázisra koncentrál. A felhasznált technológiák és eszközök, valamint az implementáció alapját képező tervek a fentebb említett cikkben megtekinthetők.

2. Implementálás

A program készítése során az MVC architektúrát kellet figyelembe vennem, ezért a programom 3 fő nagyrészre oszlik, a Controllerek, nézetek és a modellek.

2.1. The Movie Database Web API

A The Movie Database (TMDB) segítségével elérhetővé válnak a filmek címei, leírásai, műfajai és még sok más hasznos adat, amit könnyedén kinyerhetünk az API segítségével. Az API hivatalos dokumentációja megtalálható a https://www.themoviedb.org/documentation/api címen.

Ez az API a REST szoftverarchitectúrára épül, ami annyit tesz, hogy az API által szolgáltatott adatokhoz egy HTTP ige és elérési út segítségével juthatunk hozzá. A visszakapott válasz üzenetet egy JSON objektum formájában kapjuk meg.

2.1.1. API objektumok

Amennyiben egy API hívást szeretnénk végezni, akkor szükségünk van a http://api.themoviedb.org/3/ linkre, ami minden API hívásnál megegyezik. Ezután megkell határozzuk a keresés típusát, amit a következő 3 kulcsszóval tehetünk meg:

- search: Ez a leggyakoribban használt keresés, ez egy szöveg alapú keresés. A paraméterek között kapott szöveges információ alapján próbálja megtalálni a legközelebbi hasonló elemet.
- discover: Ennek segítségével visszakapunk egy nagyobb halmazt az általunk meghatározott paraméterek alapján, mint pl: értékelés, népszerűség stb…
- find: A find képes azonosítók alapján keresni, ezáltal biztosítva a legpontosabb egyezést.

A hívások rendelkezhetnek opcionális, illetve szükséges paraméterekkel. Az opcionális paramétereknek lehetnek előre beállított értékeik, ami annyit tesz, hogy ha nem adunk meg értéket a paraméterhez akkor egy előre beállított értéket fog használni. Ilyen opcionális paraméter például a language paraméter melynek segítségével beállíthatjuk, hogy az adott film

információit milyen nyelven kapjuk vissza, ezt a &language=de rész URL-hez való hozzáfűzésével tehetjük meg. Azonban a szükséges paraméterek használata kötelező egy sikeres kéréshez, ilyen paraméter például az API kulcs és a keresett film azonosítója.

Azonban, ha egy paramétert nem helyesen adunk meg, vagy olyan értéket szeretnénk lekérdezni, ami nem létezik, akkor egy hibaüzenetet kapunk vissza egy objektum formájában. Ez tartalmazza a kérés HTTP kódját és magát a hiba üzenetet, ami következő képen nézhet ki: {"status_code": 404, "status_message": "error message"}. A hibák kezelésénél figyelembe kell venni a visszakapható hibaüzeneteket is és azokat kezelni. Szerencsére a dokumentációban minden API objektumhoz megtalálhatóak a hibák típusai és felépítésük, ezáltal megkönnyítve a kezelésüket.

2.1.2. Autentikáció

Az API használatához azonban szükséges rendelkeznünk egy úgynevezett API kulccsal. Ezt a kulcsot a TMDB oldalán való regisztrációt követően igényelhetjük. Amennyiben egy érvényes kérést szeretnénk végrehajtani, a legtöbb kéréshez hozzá kell fűznünk az api_key=api_kulcs paramétert és értéket. Ahol az api_kulcs a weboldalon beszerezhető API kulcs.

2.2. Migrációk

Migrációkat a php artisan make:migration migracio_neve paranccsal hozhatunk létre. A migráció egy fájl alapú megközelítése az adatbázis menedzselésének. Segítségével elvégezhetjük az adatbázis műveleteket, akár vissza is vonhatjuk őket. Minden migrációt egy szöveges fájlban tárolunk, ezáltal egyszerűen hozzáférhetünk a fejlesztő programunkkal. Minden migráció rendelkezik egy up() és egy down() metódussal, az előbbi azt dönti el, hogy mi történik amikor végrehajtjuk a migrációt, az utóbbi pedig, hogy mi történik amikor visszavonjuk.

A users tábla migrációjában látható a felépítése egy egyszerűbb migrációnak. Az id mezőt az increments() metódussal hozzuk létre, így a beszúrt rekord azonosítója mindig egyel nagyobb lesz az előtte lévőnél. A unique() metódus pedig megakadályozza hogy két ugyan olyan értékű rekord jöjjön létre.

2.3. Modellek

Egy jól megtervezett webalkalmazásnak modell centrikusnak kell lennie. Szerencsére a Laravel számos módon segíti a fejlesztők munkáját rengeteg hasznos eszközzel, amelyek segítik a modellek készítését és menedzselését.

2.3.1. Tmdb

Ez az osztály végzi a TMBD API-hoz való kapcsolat létrehozását és minden olyan műveletet, amellyel ezeket az adatokat dolgozza fel. Alapvetően egy változót tartalmaz, amelyből az $APIkey, ez a változó tárolja a TMDB API kapcsolódásához szükséges API kulcsot. Az APIURL konstans a http://api.themoviedb.org/3/ linket tartalmazza, amely minden API kéréshez szükséges.

A _call metódus végzi a lekérdezés linkjének összeállítását és lekérdezés elküldését az API-nak. Ez a metódus rendelkezik egy $url változóval, amely a lekérdezés linkjét tárolja. Ami úgy készül el, hogy az APIURL konstanshoz hozzáfűzzük az $action változót, amely a lekérdezés típusát kapja meg a függvény meghívásakor. Aztán szükséges az API kulcsot hozzáfűzni az $APIkey változóval, majd a függvény meghívásakor megadható opcionális paramétert is hozzáfűzi, amelynek az értéke alapból egy üres string.

Majd ezt a cURL segítségével lefuttatja és a visszakapott JSON objektumot dekódolja a json_decode() függvénnyel, majd visszatér a kapott értékekkel, amelyet más metódusok segítségével feldolgozhatunk.

Ilyen metódus például a findMovieByID, melynek feladata a paraméterként kapott film azonosítója alapján megkeresni az adott filmet. Amennyiben megtalálta azt, átadja az adatokat egy Movie osztály példánynak. Azonban a műfajokat a lekérdezés egy tömb formájában adja vissza, amelyben a számok a műfajok azonosítóját jelölik, és ezt a getGenresText() metódus átalakítja olyan formátumra, amelyet tárolni lehet. Továbbá ez a lekérdezés a film borítóképét csak a fájl neveként adja vissza, amelyet a getImageFromPath() metódus végez el. Aztán a findMovieById() metódus visszaad egy Movie típusú objektumot.

A következő adatfeldolgozó metódus a searchMovie, amely képes a filmeket keresni a címük alapján, ha léteznek olyan filmek, amelyeknek a címe egyezik a keresett kulcsszóval akkor azt mindet visszaadja. Ez nagyon hasonlóan működik, mint a findMovieById() metódus, az egyetlen nagyobb különbség, hogy itt figyelembe kell venni azt is, hogy ha valaki olyan módon keres rá egy filmre, hogy a feltételnek több film is megfelel. Ez könnyedén kezelhető egy

foreach ciklussal, mivel ez a visszakapott tömb minden elemén végig megy és a tömb elemeit átalakítja Movie formátumúra, ezzel megkönnyítve az adatok kezelését. Végül az összes Movie objektum példányt visszaadja egy tömbben összegyűjtve. Fontosabb metódus még a getDiscoverMovies, amely a jelenleg felkapott filmeket tartalmazza, afféle film ajánló. Ez hasonlóan működik mint a searchMovie() metódus azzal a különbséggel, hogy ez a „discover/movie” elérési utat használja és egyetlen paramétere a $page, ami megtudja adni, hogy hányadik oldalt adja vissza.

2.3.2. Az Eloquent osztályok

Az osztályokat a php artisan make:model Modelnév -m paranccsal lehet létrehozni. A -m kapcsoló segítségével a migrációkat is létrehozhatjuk, amelyek az adatbázis tábláit segítenek létrehozni.

A User modell felel a felhasználó adatainak tárolásáért és azok szolgáltatásáért. A hasRating() metódusa vissza ad egy igaz vagy hamis értéket attól függően, hogy van-e értékelése az adott filmhez a felhasználónak. A hasFavourite() metódus pedig az vizsgálja, hogy kedveli-e a filmet. A getGenreCount() összegyűjti a filmeket, ahol a felhasználónak van értékelése és vissza adja őket egy Eloquent collectionben. A getAllGenreCount() metódus a getGenreCount() metódussal összegyűjtött műfajokat teszi egy asszociatív tömbbe.

A Rating modell az értékelések tárolásáért és megjelenítéséért felel. A getRating() metódus vissza ad egy Rating típusú modellt, amely a felhasználó és a film azonosító alapján választja ki az adatbázisból az értékelést. A getMovieAvgRating() metódus paraméterként kér egy film azonosítót és vissza adja az összes értékelés átlagát az adott filmre. Az utolsó metódusa a countUserRatings(), amely vissza adja a felhasználó azonosítója használatával, hogy hány értékelése van az adott felhasználónak összesen. A Favourite modellben megtalálható metódusok megegyeznek a Rating modellben felhasznált metódusokkal.

A Comment modell a hozzászólások modellje. A countUserComments() a felhasználó azonosítójának paraméterével megszámolja és vissza adja hány hozzászólása van az adott felhasználónak. A getDateCarbon() függvény átalakítja a hozzászólás bejegyzésének idejét emberi szemmel könnyebben olvashatóra, és a helyes időzónát is beállítja, az alábbi sor segítségével:

```
Carbon::createFromFormat('Y-m-d H:i:s', $this->date, 'Europe/Budapest')->diffForHumans();
```

Következő modell a Movie, amely filmek modellje. Ez a modell kapcsolatban áll a Comment, Rating és Favourite modellekkel. Mindegyik modellhez a hasMany() metódussal kapcsolódik. A getHTMLImage() metódus vissza adja a film poszterének HTML kódba ágyazott linkjét, amelynek a szélességét és magasságát két paraméter segítségével adhatjuk meg. Itt található még a getMoviesByGenre() metódus is, amely vissza adja az összes filmet az adott műfaj alapján amit egy paraméterrel kap meg. A getGenres() metódus vissza adja az adott film összes műfaját egy tömbben. Azonban ha mi ezt szöveges formátumban szeretnénk visszakapni vesszőkkel elválasztva akkor a getGenresText() metódust szükséges használnunk. Ennél a metódusnál szükséges volt figyelni arra, hogy néha az API segítségével visszakapott filmeknél nincs megadva műfaj és ezt egy if feltételes utasítással szükséges lekezelni.

2.4. Controllerek

Ebben a fejezetben szeretném ismertetni az alkalmazás fejlesztése során használt controllereket, amelyek a program legfőbb logikai részét tartalmazzák.

2.4.1. UserController

Ebben a Controllerben található minden olyan logika, ami felhasználóhoz kötődik. Ez a Controller kapcsolódik a user viewhoz, amely a felhasználó profil oldalát jeleníti meg. Ez csak egy metódussal rendelkezik a show-al, amit a következő route hív meg:

```
Route::get('/user/{id}', 'UserController@show');
```

Ez a route GET metódussal kinyeri az azonosítót a linkből és ezt továbbítja a UserControllerben lévő show metódusnak. Ezután az id segítségével kiválasztja a user, comments, ratings és favourites adatbázisokból az összes olyan rekordot, ahol a user_id oszlop megegyezik a linkből nyert azonosítóval.

Ez a Controller végzi még el a diagram elkészítését is. A $chart = new DoughnutChart(); sorral inicializálható a diagram. Ez a sor a Chart.js keretrendszer segítségével létrehoz egy fánk diagramot. Majd ezt szükséges feltölteni adatokkal amit a User modellben található getAllGenreCount() függvény végez el. Ez a függvény visszaadja a felhasználó által értékelt kategóriák számát, minden egyes kategóriára lebontva. Majd ezt az értéket a diagram labels metódusának kell átadni figyelve arra, hogy ide csak a kategóriák nevei kerüljenek, amit a PHPban található array_keys() függvénnyel oldottam meg. Továbbá fel kell tölteni értékekkel is a diagramot amit az array_values() függvénnyel oldottam meg. A diagram színeit az options metódussal adtam meg, hogy azok jobban kivehetők legyenek.

Majd az összes adatot szükséges átadni a nézetnek, hogy az képes legyen azokat megjeleníteni, amelyet a compact-tal tehetünk meg ha azt átadjuk a with metódusnak.

2.4.2. MovieRatingController

Ez a Controller végzi az összes értékeléssel kapcsolatos műveletet, mint például az értékelés hozzáadása az adatbázishoz és módosítása. A store metódus ami tárolja az értékelést a következő route hív meg: Route::resource('/rating', 'MovieRatingController'). Az értékelések tárolásánál figyelembe kell venni, hogy ha valaki már rendelkezik egy értékeléssel akkor nem egy új adatbázis rekordot kell létrehoznunk, hanem felül írni egy már meglévő rekordot az új értékekkel. Ezt a feltételt egy if utasítással végeztem el, aminek a feltétele a következő képen alakul:

```
Rating::where('movie_id',$request->movie_id)
->where('user_id', $user->id)->exists()
```

Ez az utasítás végez egy lekérdezést, ahol a $request változó az oldalról visszakapott hidden input mezőben lévő movie_id azonosítót és az értékelés értékét tartalmazza, ami az adott oldalon lévő film azonosítója. A lekérdezés kiválasztja movie_id alapján azokat az értékeléseket, amelyeknél a felhasználó a jelenleg bejelentkezett felhasználó azonosítója megegyezik a ratings táblán tárolt user_id-vel, ennek létezését az exists() metódus vizsgálja meg, ha létezik true értékkel tér vissza, amennyiben ez a rekord létezik azaz már van értékelése az adott felhasználónak akkor a Rating modell getRating() metódusával a felhasználó és a film azonosítójával visszakapjuk a jelenlegi értékelést. Majd ezt felül írjuk a $rating->fill($data); sorral, ahol a $data változó tartalmazza az új értékeléshez tartozó adatokat, és végül mindezt a save() metódussal mentjük. Azonban, ha még nem létezik ez az értékelés szükséges azt létrehozni és az előbbi módszerrel feltölteni a $rating változót az új adatokkal.

2.4.3. MovieFavouriteController

Ennek a Controllernek a feladata a kedvelések tárolása, illetve eltávolítása az adatbázisból. Mint a MovieRatingController-nél itt is a store metódusban található a program logikája, amelyet a következő sor hív meg:

```
Route::resource('/favourite', 'MovieFavouriteController');
```

Az első lépésben szükséges létrehozni egy tömböt, amiben a Controller számára elküldött adatok szerepelnek.

A jelenleg bejelentkezett felhasználóhoz tartozó adatokat az Auth osztályból lehet kinyerni az alábbi módon: $user = Auth::user();, ez a felhasználóhoz kapcsolódó összes adatot tartalmazza. A $data változóban a user_id a felhasználó azonosítója, a movie_id a film azonosítója, amit egy hidden input mező ad át az oldalról. A date pedig a pontos dátum, amelyet a Carbon osztály generál a Budapesti időzóna szerint. Ezután szükséges megvizsgálni, hogy létezik-e már a kedvenc a jelenleg bejelentkezett felhasználónak az adott filmnél, ezt a User modell hasFavourite() függvényével végzem el, amely egy igaz vagy hamis értékkel tér vissza. Amennyiben létezik az adott kedvenc, a Favourite modell getFavourite() metódusával ki kell választanunk a jelenlegi kedvencet majd a delete() metódussal törölni azt.

2.4.4. MovieController

Itt kerül feldolgozásra az összes filmekkel kapcsolatos adat. Ez a Controller a GET metódussal kap egy azonosítót, amely a film azonosítója a következő route segítségével:

```
Route::get('/movie/{id}', 'MovieController@index');
```

Ez a route átadja az azonosítót az index függvénynek, amely az adott azonosító alapján kiválasztja a filmet az adatbázis movies táblájából. Azonban, ha a film nem létezik a táblában akkor a TMDB API segítségével hozzá kell adni azt. Az alábbi if utasítás azt vizsgálja, hogy a $movie változóban található-e érték, amennyiben nem található, akkor lefut az utasítás. Ahhoz, hogy az API segítségével megkeressük a filmet szükség van egy új TMDB példányra, majd a findMovieById metódus segítségével kiválasztható a film, amit értelemszerűen menteni kell az adatbázisba a save() paranccsal.

Mivel az oldalon a filmekhez tartozó kommenteket is meg kell jeleníteni, ezért azokat ki kell listázni a comments táblából a film azonosítója alapján lehetőleg időrend szerint csökkenőben. A következő lépés az értékelés megjelenítése a bejelentkezett felhasználónak. Itt fontos megvizsgálni, hogy az adott felhasználó be van-e jelentkezve és rendelkezik-e az adott filmhez tartozó értékeléssel. Amennyiben rendelkezik akkor a $rating változó értéke az adatbázis ratings táblájából kiválasztott értékelés, ahol a film és a felhasználó azonosítója megegyezik az adatbázis rekorddal. Azonban, ha nem rendelkezik értékeléssel, de be van jelentkezve akkor a $rating változó star tulajdonságát nullára kell állítani, hogy a nézet ne fusson hibára amikor megszeretné jeleníteni azt.

Ezt az esetet a kedvencek megjelenítésénél is figyelembe kell venni, azonban itt nem kell nullára állítani az értéket, mert itt csak azt szükséges vizsgálni, hogy az adott bejelentkezett felhasználónak kedvence a film vagy sem.

A másik függvénye a Controllernek a search, amely a filmek keresésért felel. Fontos megjegyezni, hogy itt nem csak a filmek címére, hanem azok műfajára is kereshetünk, külön-külön vagy akár műfajra és film címre is egyaránt. Ezt a keresést a következő kód részlet biztosítja:

```
if($request->genre != null && $request->title != null)
```

A $request változó tartalmazza a kereső mezőből kinyert adatokat, amely nem más, mint a cím és a műfaj. Amennyiben ez a két érték nem egyenlő null-al, azaz a felhasználó rákeresett valamire csak akkor futnak a le a feltételben lévő utasítások. Amennyiben a feltétel teljesül a következő kódsor kiválasztja a filmeket, amelyeknek a műfajuk és a címük megegyezik a keresés szövegével, és a paginate függvény segítségével oldalakra rendezi azt.

Azonban arra is szükséges egy feltételt írni, hogy ha a felhasználó csak egy adott műfajra vagy film címre keres külön-külön. Ezeket az elseif utasítással vizsgáltam meg.

Továbbá ennek a Controllernek a feladata az is, ha a felhasználó által keresett film címe nem létezik az adatbázisban, akkor azt a TMDB API segítségével keresse meg, és amennyiben talál ilyen című filmet vagy filmeket, azokat jelenítse meg a keresés eredményei között. Itt fontos megjegyezni, hogy nem menti az összes filmet, amit az API segítségével keresett, hanem csak azt adja hozzá amelyikre a felhasználó rá kattint.

2.4.5. MovieCommentsController

Ez a Controller végzi a kommentek tárolásával és törlésével járó műveleteket.

```
Route::resource('/comment', 'MovieCommentsController');
```

Ez a route felel a kommentekhez tartozó adatok szolgáltatásáért és a kommentek tárolásáért.

```
Route::get('comment/{id}/delete','MovieCommentsController@destroy');
```

Ez a route a hozzászólás azonosítóját adja át a linkből a destroy metódusnak a GET segítségével. Itt két metódus található meg, az első a store, amely a hozzászólások tárolását végzi el. Ehhez csak a bejelentkezett felhasználó és a film azonosítójára és az oldalról visszakapott hozzászólás szövegére van szüksége. Ezen adatok segítségével a Comment modellen meghívott create statikus metódussal elmenthető a hozzászólás. Majd a redirect()->back() visszatérési értékkel visszatér az előző oldalra.

A másik függvény a destroy, amely a hozzászólások törlésért felel. Itt arra kell ügyelni, hogy ha valaki szeretné törölni a hozzászólást, és nem ő írta azt akkor azt a kérést el kell utasítani,

mivel mindenki csak a saját hozzászólását törölheti. Ilyenkor az oldal egy 403-mas kódot dob vissza, ami a „Hozzáférés megtagadva” HTTP státuszkód. Azonban, ha megegyezik a jelenleg bejelentkezett felhasználó azonosítója a hozzászólás írójának az azonosítójával akkor a delete metódus törli azt.

2.5. Nézetek

A nézetek felelősek az adatok megjelenítéséért és a felhasználó által bevitt adatok továbbításáért. Minden nézet egy blade.php kiterjesztésű fájl. A Blade egy egyszerű, de nagyon hasznos sablon motor, melynek segítségével kezelhetünk adatokat a nézeteinken belül. Habár az MVC elvnek a lényege, hogy a logika ne nagyon szerepeljen a nézeteinkben, néha szükségünk van egy kevés logikára az adatok megjelenítéséhez. A Blade fájlok segítségével kiírathatunk változókat, végig mehetünk tömbökön, használhatunk feltételes utasításokat és rengeteg más hasznos dolgot. A Blade szintaktikája nagyrészben eltér a PHP szintaktikájától. Amennyiben egy változót szeretnénk kiíratni az dupla kapcsoszárójelek között tehetjük meg, ami azért is hasznos mert ez alapból védelmet biztosít az XSS ellen.

A nézeteket több csoportba célszerű osztani, az egyik a layouts ahol csak az oldalak felépítését vagy kinézetét tároljuk. Ezek számára az adatokat és tartalmat a views mappában lévő oldalak szolgáltatják. A layouts-ban lévő oldalak a @yield(’tartalom’) sorral hívják meg a views-ban található @section(’tartalom’) részeket. Itt található még meg az inc mappa is, amely a konstans kód beágyazásában segít, mint például a navigációs bár, lábléc vagy a headerben definiált linkek és scriptek. A bejelentkezést és a regisztrációt a Laravel alapból legenerálja nekünk, ami pedig az auth mappában található meg.

Fontos megjegyezni, hogy minden nézetnek van egységes pontja, amelyek az inc mappában találhatóak. Mivel mindegyik weboldalra szükséges beszúrni a Bootstrap keretrendszer CDN linkjét, a diagramhoz szükséges Chart.js-t, a navigációs bárhoz szükséges JQueryt és a FontAwesome betűtípust, ami ikonok megjelenítésére szolgál. Továbbá minden oldalra beszúrásra kerül a navigációs bár, ami az oldalon való navigációra szolgál. Végül a footer is megtalálható itt, aminek a feladata a lábléc megjelenítése.

2.5.1. Filmek megjelenítése

A filmek megjelenítéséért a movie.blade.php felelős. Ez a nézet jeleníti meg a hozzászólásokat, értékeléseket és a kedveléseket az adott filmekhez. Ez az oldal több fő részből épül fel,

amelyből az első a content, ez a rész jeleníti meg a film képét, címét és leírását. Ez a view a MovieControllertől kapja a megjelenítendő információkat.

A második fő szekció a rating amely az értékelést jeleníti meg. Ez a szekció tartalmaz némi logikát, ami egy for ciklust tartalmaz. Ez a ciklus nullától ötig megy végig és egy if feltételes utasítással megvizsgálja a jelenlegi ciklus vezérlő változó értékét és ha az megegyezik az értékelés értékével akkor azt a rádió gombot bejelöli. Erre azért van szükség, mert ha valakinek már van értékelése azt úgy kell megjeleníteni, hogy az a csillag legyen bejelölve amilyen értékelést hagyott a felhasználó. Ezeket a rádió gombokat dinamikusan generálom a ciklus változóval, hogy egyedi módon lehessen rájuk hivatkozni. A gomboknak külön label mezőjük van, amely a CSS megformálásában segít. Fontos még megjegyezni, hogy ez az if utasítás egy @auth blokkba van. Ez a blokk csak akkor fut le, ha a felhasználó be van jelentkezve másképpen hibára futna a program. Amennyiben a felhasználó rákattintott az öt csillag egyikére akkor meghívja a this.form.submit() metódust, amivel a form automatikusan elküldésre kerül a POST metódussal a MovieRatingController store funkciójának.

Továbbá amennyiben a felhasználó nincs bejelentkezve akkor is meg kell jeleníteni az értékelést ezt a @guest blokkal tehetjük meg, és itt is ugyan úgy minden rádió gombot bejelölés nélkül kell kiíratni. Szükség van még egy rejtett input mezőre, amely a film azonosítóját tárolja.

A következő főbb része favourite, ez a szekció a POST metódussal adja át az adatokat a MovieFavouriteController store funkciójának. Mivel itt is fontos megvizsgálni azt, hogy az adott felhasználó be van-e jelentkezve, ezért szükséges egy if feltételes utasítás, amely az Auth::user()->hasFavourite($movie->id) megvizsgálja, hogy létezik-e kedvelése az adatbázisban a bejelentkezett felhasználónak a jelenlegi oldalon lévő filmhez. Amennyiben létezik ilyen bejegyzés akkor a checkbox bejelölve kerül kiíratásra, ha nem akkor értelemszerűen nem lesz bejelölve.

Az utolsó szekció pedig a comments, ami a hozzászólások megjelenítésére szolgál. Itt figyelembe kell venni, ha nem létezik komment akkor a kód ne fusson le, különben hibát kapnánk a nem létező változó meghívása miatt. Ezt az ellenőrzést egy if utasítás végzi, amely a count() függvénnyel megszámolja a $comments tömb tartalmát és ha az nagyobb mint nulla csak akkor fut le. Egy foreach ciklus végig megy a MovieControllertől kapott $comments tömbön és minden egyes hozzászólást kiírat. A hozzászólás idejét a getDateCarbon() függvény segítségével iratom ki, ez a függvény átalakítja a dátumot könnyeben olvasható formátumra.

Minden egyes hozzászólás rendelkezik egy lenyíló menüvel, ahol megtalálható a hozzászólás törlése link, de ez csak akkor, ha az adott felhasználó írta a hozzászólást. Ezt a vizsgálatot a következő utasítás végzi:

```
@if (Auth::user()->id == $comment->user->id)
```

A feltételes utasítás megvizsgálja, hogy az adott bejelentkezett felhasználónak az azonosítója megegyezik-e a hozzászólás írójának azonosítójával. Itt lehet látni, hogy az Eloquent milyen módon kapcsolja össze a modelleket az adatbázissal és egymással. Ezt az egész feltételes utasítást szükséges egy @auth blokkba rakni, hogy a program ne fusson hibára.

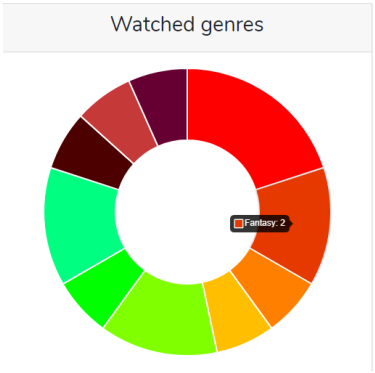
2.5.2. Keresés

A filmek keresésének megvalósítását a search.blade.php végzi el. Ez a nézet akkor kerül megjelenésre, ha a /search link kap egy GET kérést vagy valaki megnyitja a keresés oldalt. Az adatokat a MovieController search metódusától kapja vissza, amely egy collection, amelyben Movie típusú elemek találhatóak meg. Ennek a nézetnek csak egy szekciója van a content. Ebben a szekcióban kerül a tömb feldolgozásra egy @foreach ciklussal. Fontos megjegyezni, hogy a MovieController a collectiont a paginate() függvényével adja át a nézetnek. Ez a függvény képes az adatbázisból kinyert adatokat oldalakba rendezni és azokat egyszerre átadni a nézetnek, hogy az oldalak megjeleníthetők legyenek szükség van az alábbi sor beszúrására a nézetben:

```
{{ $movies->appends(request()->input())->links() }}
```

2.5.3. Felhasználói profil

A felhasználói profil megjelenítésre a user.blade.php szolgál. Itt kerülnek megjelenítésre a felhasználó statisztikái. A UserController show metódusától kapja az információit. Mind a hozzászólásoknak, kedveléseknek és az értékelésnek az összesített számát jeleníti meg. Itt található még egy fánk diagram is, amely megjeleníti, hogy a felhasználó hány filmet nézett meg milyen műfajban. Különböző szakaszokra bontja szét a diagramot, mindegyik szakasz valamilyen színt használ a megkülönböztetéshez és ha a felhasználó ráviszi valamelyikre a kurzorját akkor láthatja az adott műfaj nevét is (1. ábra).



1. ábra: A műfajok diagramja

2.5.4. Navigációs bár

Fontos megemlíteni még a navigációs bárt, amely az inc mappában található navbar.blade.php. A navigációs menüpontok lista elemek. Ezen található meg a bejelentkező menüpont, amelyet csak akkor kell megjeleníteni, ha a felhasználó nincs bejelentkezve erre a @guest sor szolgál. A linket olyan módon szükséges megadni, hogy az mindig a login oldalra mutasson, még akkor is, ha a weboldal másik címre költözne, erre szolgál az url() metódus.

Amennyiben a felhasználó be van jelentkezve szükséges megjeleníteni egy kijelentkező gombot és a felhasználó saját profiljára mutató linket. Az alábbi kódsoron látható, hogy a felhasználó profiljának linkjét a felhasználó nevére kattintva érheti el.

Itt található még meg a kereső, amelynek segítségével műfajokra és filmekre kereshetünk rá. Ez a kereső egy GET metódussal küldi el az adatokat a MovieController search függvényének, egy címet és egy műfajt továbbít. Mint az alábbi kódsoron látható amennyiben a $request változó tartalmaz title vagy genre értéket, tehát valaki rákeresett egy filmre vagy műfajra akkor azt a kereső mező úgymond elmenti, hogy tudja a felhasználó pontosan mire is keresett rá miután lenyomta a keresés gombot.

2.5.5. Middleware

A middleware segítségével filterezhetők a HTTP kérelmek, ezáltal a middlewarek megvalósíthatnak bejelentkezést, naplózást vagy akár egy CORS middleware segítségével elhelyezhetnek megfelelő fejléceket a kimenő válasz csomagokra.

A middleware létrehozásához szükség van a php artisan make:middleware <név> parancsra. Ez a parancs létrehozza az osztályt az app/Http/Middleware könyvtárban.

Két fajtáját különböztethetjük meg:

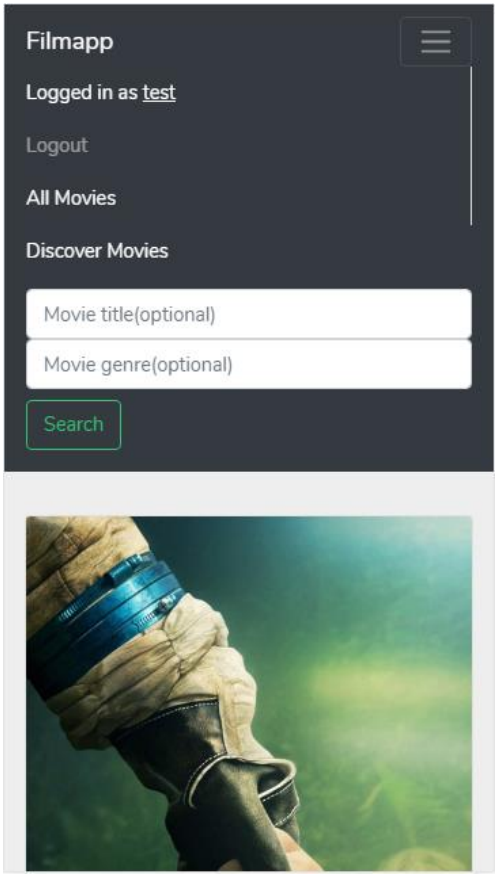
1. BeforeMiddleware: A kérés kezelése előtt fut le.
2. AfterMiddleware: A kérés kezelése után fut le.

Ahhoz, hogy egy middlewaret használni lehessen az útvonalakon, azt regisztrálni kell a következő sort ’Nev’ => \App\Http\Middleware\Nev::class a $routeMiddleware változóban, ami az app/Http/Kernel.php elérési útvonalon található meg. Azonban, ha több útvonalhoz szeretnénk kapcsolni a middlewaret akkor lehetőségünk van csoportok létrehozására is.

Én az oldal bizonyos funkcióinak használatát korlátoztam abban az esetben, ha az adott felhasználó nincs bejelentkezve. Ez az auth middleware segítségével tudtam véghez vinni. Az auth middleware akkor jön létre amikor a Laravellel hozzá adjuk a hitelesítés funkciókat az alkalmazáshoz. Mivel több oldal használatát is korlátoznom kellett ezért célszerű volt egy middleware csoport létrehozására. Mint az alábbi kódsoron is látszik, a hozzászólások írását és törlését, az értékelést és a kedvelést tartalmazza a csoport. Amennyiben valamilyen felhasználó úgy akarja elérni ezeket a funkciókat, hogy nincs bejelentkezve, akkor automatikusan a bejelentkező oldalra irányítja a middleware.

2.6. Kijelző kompatibilitás

A program megírása során, nagy hangsúlyt fektettem arra, hogy bármilyen méretű képernyőn és eszközön ugyanaz az átláthatóság és használhatóság jellemezze. Ez a Bootstrapnak köszönhetően volt megvalósítható. Az oldal minden egyes eleme úgy van felépítve, hogy a méretarányokat megtartsa a kijelző méretétől függően. A kijelző kompatibilitás fejlesztése közben segítségemre állt a Google Chrome beépített HTML vizsgálója, ami segített szimulálni a különböző kijelző méreteket (2. ábra).



2. ábra: A navigációs bár egy Galaxy S5 készüléken (360x640)

3. Tesztelés

A tesztelés az egyik legfontosabb feladata a fejlesztés fázisának. Megkellett vizsgáljak minden funkciót és program részt, hogy azok megfelelően működnek-e és teljesítik-e a specifikációban leírt funkciókat és elvárásokat.

3.1. Adatbázis tesztelése

Az adatbázis tesztelése egy nagyon fontos feladat, hiszen minden fontos adatot az adatbázis tárol. Adatbázis nélkül az egész program használhatatlanná válik, mert a weboldal teljes mértékben függ az adatbázistól. Nem tudna filmeket, kommenteket, értékeléseket megjeleníteni, továbbá az egész felhasználó hitelesítés se működne.

3.1.1. Tinker

A Laravelben beépítetten megtalálható a PsySH konzolra írt nagyon hasznos interaktív shell, amely képes a felhasználó által bevitt adatokat értékelni és visszaadni azok eredményét, ezt másnéven REPL-nek is nevezzük, ami a Read-Eval-Print-Loop rövidítése. A Tinker

segítségével adatbázis műveleteket hajthatunk végre, mint például a lekérdezés, frissítés és törlés. Legfőbb hasznos tulajdonsága, hogy képesek vagyunk osztályokat használni a bevitelnél, ezáltal a Laravel segítségével egyszerűen hozzáadhatunk, módosíthatunk vagy lekérhetünk adatbázis rekordokat. Ezáltal lehetőségünk adódik az Eloquent által használható methódusok használatára is, mint például: User:all() parancs segítségével kinyerhetjük az összes felhasználó adatait a táblánkból. Fontos megjegyezni, hogy ezzel a paranccsal egy Eloquent collection objektumot kapunk vissza amin további műveleteket végezhetünk. Képesek vagyunk változók használatára is, ezeket a PHP-ban már megszokott módon $ jellel definiálhatunk.

3.1.2. Factory

Az adatbázis felöltése kézzel nagyon hosszadalmas és megterhelő folyamat, mivel minden rekordot egyesével kell felvinnünk manuálisan. Emiatt a probléma miatt hozták létre a gyárakat azaz Factorykat. A Factory segítségével képesek vagyunk

Mielőtt feltöltenénk egy Factoryt létre kell azt hoznunk, ezt a php artisan make:factory FactoryNeve paranccsal tehetjük meg, amely a database/factories mappába hozza létre az adott fájlt. Ezt megtehetjük úgy is, hogy a Factory alapból tartalmazza a modell definiálását, ha használjuk a --model ModellNeve kapcsolót a parancs után. Összesen öt darab Factoryt használtam fel, minden táblához egyet, kivéve a genres táblához mivel ott mindig ugyan azok az adatok. Az alábbi kód segítségével hoztam létre a users táblához tartozó Factoryt.

Ezeket a factorykat a következő kódsorral hívhatjuk meg: factory(\App\Movie::class, 20)->create(); ahol meg kell adnunk az osztályt és hogy hány rekordot szeretnénk generálni az adatbázis táblánkba. Mint az előbbi kódrészleten látható felhasználtam a Laravelben alapból megtalálható Faker osztályt. Ez egy olyan osztály melynek segítségével véletlen szerű adatokat generálhatunk megadott feltételek alapján.

3.1.3. Seeder

Azonban a Factorykat nem elég csak elkészíteni, valahol meg is kell hívni őket, erre szolgálnak a seederek. Amennyiben létre szeretnénk hozni egy seedert azt a php artisan make:seeder SeederNeve paranccsal tehetjük meg. Ez a parancs létrehoz egy seedert a database/seeds mappába. A seedereket a php artisan db:seed parancs segítségével futtathatjuk le, vagy amikor egy adatbázis migrációt végzünk a következő módon: php artisan migrate:refresh –seed.

3.2. Unit tesztelés

A unit tesztek segítenek a fejlesztőknek a hibák azonosításában és javításában, és egyfajta dokumentációként is szolgálnak. A unit teszteknek ideálisan le kell fedniük a program minden lehetséges irányát.

Az adatbázis tesztelése mellett még nagyon fontos az oldalon lévő funkciók tesztelése is, habár itt nem teszteltem le minden lehetséges hibaforrást csak néhány nagyobb részt. A tesztek egy részét a PHPUnit-al végeztem el, ami egy unit tesztelő keretrendszer PHPra. Ez a keretrendszer alapból megtalálható a Laravelben. A unit teszteléshez php artisan make:test TestNév parancsot kell futtatnunk a terminálba. Ez létrehoz egy fájlt a tests/Feature mappába. Rengeteg részét le lehet tesztelni a weboldalnak, mint például adatbevitel és visszatérési értékek, linkek működése és például adott szövegrészletek szerepelnek-e az oldalon.

Az alábbi kódsor teszteli, hogy a movies link megnyitásával a HTTP állapotkód megegyezik a 200-al, ami annyit tesz, hogy sikeres volt a kérés.

```
$response = $this->get('/movies');
$response->assertStatus(200);
```

Leteszteltem azt is, hogy a felhasználó a login link megnyitásával a helyes nézetet kapja vissza.

```
$response = $this->get('/login');
$response->assertSuccessful();
$response->assertViewIs('auth.login');
```

A következő kódsor azt vizsgálja, hogy amennyiben egy felhasználó be van jelentkezve és megnyitja a login oldalt, az sikeresen visszairányítja-e a főoldalra. Itt megfigyelhető az, hogy a user factory osztállyal létrehozott felhasználó használható a bejelentkezés szimulálásához.

```
$user = factory(User::class)->make();
$response=$this->actingAs($user)->get('/login');
$response->assertRedirect('/home');
```

A következő metódus a test_user_can_register azt vizsgálja, hogy működik-e a regisztrációs funkció. Itt látható, hogy létrehoztam egy teszt felhasználót, amit egy tömbben tároltam, és azt egy POST metódussal elküldtem a register oldalnak. Aztán megvizsgáltam, hogy a regisztráció

után visszairányítja-e a felhasználót a főoldalra. Végül az adatbázist kellett megvizsgálni, hogy a users tábla biztosan tartalmazza az általunk regisztrált felhasználót.

```
$user = [
            'name' => 'Teszt Elek',
            'email' => 'testemail@test.com',
            'password' => Hash::make('passwordtest'),
        ];

        $response = $this->post('/register', $user);
        $response->assertRedirect('/');
        $this->assertDatabaseHas('users', [
            'name' => $user['name'],
            'email' => $user['email'],
]);
```

Hogyha bonyolultabb teszteket szeretnénk készíteni, akkor használhatjuk a Laravel Duskot ami a tesztelést úgy oldja meg, hogy szimulál egy böngészőt és azon keresztül végzi a tesztelést. Ezáltal a kliens oldali funkciókat is letesztelhetjük, mint például a HTML require. A Duskot mint minden más komponenst a composerrel lehet, az alábbi módon composer require laravel/dusk. A futtatása pedig a php artisan dusk paranccsal lehetségses.

Az utolsó teszt a test_user_cannot_login_with_incorrect_password, amely azt vizsgálja, hogy a felhasználó képes-e hibás jelszóval bejelentkezni. Létrehoz a a factory metódus egy felhasználót, aki a „valid-password” jelszót kapja, majd megnyitja a login oldalt és begépeli a létrehozott felhasználó emailjét és a hibás jelszót, aztán ezt elküldi és megvizsgálja, hogy a böngésző még mindig a login oldalon áll és keletkezett-e hibaüzenet.

```
 $this->browse(function (Browser $browser) {
     $user = factory(User::class)->create([
       'password' => Hash::make('valid-password'),
     ]);
           $browser->visit('/login')
            ->type('email', $user->email )
```

```
            ->type('password', 'invalid-password' )
            ->press('Login')
            ->assertPathIs('/login')
            ->assertSee('These credentials do not match our records');
        });
```

4. Tovább fejlesztési lehetőségek

4.1. Bejelentkezés és regisztráció

A regisztráció és bejelentkezés jelenleg elég minimális funkciókkal rendelkezik. Mivel amennyiben valaki elfelejti a felhasználójához tartozó jelszavát, akkor arra nincs mód, hogy azt módosítsa vagy lekérje. Továbbá, ha egy oldal rendelkezik bejelentkezés funkcióval, azt a weboldalt szükséges SSL hitelesítéssel ellátni mivel enélkül titkosítás nélkül küldi tovább a böngésző a felhasználó jelszavát a szervernek. Fontos lehet még a kétfaktoros hitelesítés bevezetése is, de ez csak akkor javasolt, ha fontos adatokat tárolunk, mint például lakcím, bakszámlaszám, telefonszám stb.

A regisztrációt szükséges lehet bővíteni életkor és nem megadásával, mivel ennek segítségével további statisztikák végezhetők, melyek segítségével akár ajánlásokat is személyre szabhatunk.

4.2. Hozzászólások

A hozzászólások jelen állapotban csak filmekhez kapcsolódhatnak és az egyetlen művelet, amit egy létező hozzászóláson elvégezhetünk az a törlés. Ezt lehetne bővíteni, hogy akár hozzászólásokra válasz hozzászólásokat írhatnánk vagy akár a felhasználók kedvelhetnének adott hozzászólásokat. Jelen formában, ha valaki hozzászól egy filmhez és azt szeretné módosítani, azt csak úgy teheti meg, ha azt törli és újra írja a javított szöveggel. A hozzászólásokat alapból úgy terveztem, hogy azok későbbi bővítésre készen álljanak, ezért már megtalálhatóak a linkek, amelyek segítségével ezek a műveletek elvégezhetők lehetnének.

4.3. Sorozatok hozzáadása

A filmek mellett lehetne a sorozatokkal kapcsolatos funkciók bevezetése is, ez szintén megvalósítható TMDB API segítségével, mert tartalmaz sorozatokkal kapcsolatos API lekérdezéseket. Azonban ez bonyolultabbá tenné az alkalmazást, mert figyelembe kell venni, hogy egy sorozatnak több évadja van és azok több részből állnak.

4.4. Felhasználói profil bővítése

Jenleg a felhasználói profilokat nem lehet személyre szabni semmilyen módon. A későbbiekben szükséges lehet olyan funkciókat implementálni mint:

- Profil kép feltöltése és módosítása
- Jelszó megváltoztatása
- Név megváltoztatása
- Tartózkodási hely megadása
- Listák készítése (pl.: Megnézendő filmek)

5. Konklúzió

A projektem keretében egy filmes webalkalmazást fejlesztettem, melyet a Laravel keretrendszer segítségével valósítottam meg. A megvalósításhoz számos különféle függőséget használtam. A projekt során betekintést nyertem a REST alapú rendszerek felépítésébe és kezelésébe, valamint elsajátítottam az MVC technológia előnyeit, miközben a Laravel keretrendszerrel dolgoztam. A felhasználói felületet a Bootstrap keretrendszer segítségével alakítottam ki.

A projektet fokozatosan, terveim alapján valósítottam meg. Az első lépés az információk gyűjtése volt, hogy meghatározzam, milyen keretrendszereket kell bevonnom a fejlesztésbe, és milyen célokat kívánok elérni a programmal. Ezt követte a második lépés, melyben megterveztem magát a programot és az adatbázist. A következő fázisban valósítottam meg az implementációt, melyet részletesen tervezett lépésekre bontottam. Először elkészítettem a modelleket, majd a controllereket, és végül a felület kinézetét finomítottam. Ezt követte a tesztelés fázisa, amely az utolsó lépés volt a folyamatban.

Irodalomjegyzék

Bean, M. (2015). *Laravel 5 essentials*. Packt Publishing Ltd.

Nixon, R. (2014). *Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5*. " O'Reilly Media, Inc."

Fielding, J. (2014). *Beginning responsive web design with HTML5 and CSS3*. Apress.

Rövid szakmai életrajz

2020 óta alkalmazás fejlesztőként dolgozom a Máv Szolgáltató Központ Zrt.-nél. A szakterületem az automatizált tesztelés és szoftverfejlesztés amelyeket Java nyelven végzek.