Reményi Gergő

Connector

TARTALOMJEGYZÉK

Bevezetés

Az elmúlt évek során már többször is felötlött bennem a választott projektem valamilyen használható formába való öntése. Mivel eddig nem sikerült ezt megvalósítanom, tökéletes alkalom a diplomamunka elkészítése, hogy belefogjak a projektem elkészítésébe, ami több fázisból és elkülönülő részből fog állni. Mielőtt belefognék a részletes leírásába a témámnak, szeretnék időt fordítani a munkám kezdeti lépéseire és ötleteire\*.

Az első alkalommal már az alapképzésem közepén szerettem volna foglalkozni ezzel a témával, amikor szintén egy szakdolgozati témaválasztás előtt álltam, viszont nem volt lehetőségem megvalósítani a dolgozatot, így már akkor tudtam, hogy mesterképzésem során mindenképpen megpróbálkozom azzal a feladattal, hogy ezt a projektet valósággá formáljam. Ahhoz, hogy érthető legyen a téma mindenki számára, a téma kiválasztásának gondolatmenetét fogom felvázolni, valamint annak továbbfejlesztését és kibővítését a jelenlegi állapotáig.

A végleges projektem egy való életbeli problémára ad megoldást, amely nagyban elősegíti a hétköznapi életet akár munka, hobbi vagy szabadidő tekintetében.

Mint említettem, az alapképzés során döntöttem el, miszerint szeretnék egy olyan alkalmazást fejleszteni, amivel egyszerűen nyilvántartható bármilyen megtekintett médiatartalommal (TV sorozat, film) kapcsolatos információ. Ezalatt az információ alatt értem azokat az adatokat, melyek rendre a médiatartalom adatai (cím, alcím, műfaj, szereplők, kiadó stb..), valamint a megtekintéshez kapcsolódó adatok. Utóbbihoz tartoznak azok a feladatok, hogy tároljuk azt, hogy milyen százalékban tekintettük meg a médiát, mikor tekintettük meg. Ehhez automatikusan hozzákapcsolódik az, hogy milyen tartalomról beszélünk, TV show vagy filmről beszélünk. Ezeket külön esetekként kell kezelni a későbbiekben. Ezekhez az általános tárolásokhoz kapcsolni szerettem volna funkciókat, ezzel már valamilyen szinten közelítünk a jelenlegi szakdolgozat lényegi részéhez. Ezek a funkciók, amelyek kapcsolódnak a projekthez teljesen automatizált funkciók, amelyekbe a felhasználónak egyáltalán nem kell beavatkoznia, lehetősége nincs is rá, mivel a háttérben az alkalmazás megold mindent, ami szükséges a projekt működéséhez. Ez az automatizáció a jelenlegi projekt egyik fő alappillére. Ezen funkciók közé tartoznak például, adott tartalom automatikus importálása több REST API-n keresztül történő integráció által, vagy az idegen nyelvű médiához tartozó anyanyelvű felirat beszerzése. Ezekről a funkciókról később bővebben írok.

Az alapötlet önmagában egy nyilvántartó rendszer lett volna, melyet egy szemi-automatizált rendszer végzett volna a felhasználó beavatkozásával egyetemben. Ennek a rendszernek a megvalósítása első körben egyetemi témaválasztási okok miatt nem jöhetett létre, viszont most sikeres elfogadás után sikerült implementálnom.

Ennek a témának a teljesen automatizált változatát választottam alapjául a jelenlegi projektemnek. A szakdolgozatomhoz azonban az idők során felgyülemlett ötleteimet valamilyen módon szerettem volna integrálni. Így egy olyan rendszerré sikerült bővítenem a nyilvántartó rendszerem, amely valóban egy hétköznapi élet segítését szolgáló alkalmazás. Ebben olyan adatfeldolgozás és olyan statisztikák, valamint automatizált rendszerek vannak jelen, amelyekkel egyszerűen, hatékonyan tudunk kimutatásokat végezni. Használata olyan egyszerű, valójában nem kell tennie semmit a felhasználónak, mivel ez egy hétköznapi tevékenység követő rendszer. Ezzel egyszerűen a használathoz csak egy asztali alkalmazást kell telepíteni, majd végezni a hétköznapi feladatainkat. Mivel napjainkban szinte minden tevékenység igényel valamilyen elektronikus eszközt (okostelefon, számítógép, stb..), ezért ezeknek a használati aránya nagyon nagy. Ebből következik, hogy iszonyú mennyiségű adat nyerhető ki ezen eszközök használata alapján. Ez vezetett ahhoz, hogy a témabővítés során ez szolgált egy olyan alapul, amellyel szerettem volna olyan funkciókat beépíteni, melyekhez ez adja a megfelelő adatokat.

Így tehát a használt alkalmazásaink követésével foglalkoztam, amelyekhez különböző funkciókat rendelek. Továbbá az alkalmazás szervezése és fejlesztési modell miatt, minden alkalmazás követése egy külön modulként valósul meg. Így minden felhasználó egyszerűen választhatja ki melyik alkalmazást szeretné követni az alkalmazáson belül.

Mivel mindennapjaink egyik legismertebb és leghasználtabb platformja a közösségi média, így lehetőséget látok abban, hogy egy egyszerűen kezelhető, embereket összefogó, velük könnyen interakcióba léphető felületbe az eddig említett alkalmazások követésére szolgáló modulokat beépítsem. ???????????

Egy letisztult kép úgy fest az alkalmazásról, miszerint az alap felület egy közösségi média, mely regisztrációhoz kötött minden felhasználó számára. Ezzel olyan lehetőségek merülnek fel, melyeket az alkalmazások követésével összekötve hasznosíthatunk. Ezalatt az emberi interakció egy olyan módja valósul meg, amellyel a legpontosabb közös érdeklődések alapján ismerhetünk meg, vagy kérhetünk segítséget olyan felhasználóktól, akik esetleg tapasztaltak az adott területen, vagy jártasak az adott alkalmazás használatában.

Viszont a középpontba először maga a modulszerkezet kerül, ez adja az alkalmazás magját, eszerint tudunk kimutatásokat és adatfeldolgozást végezni.

Miután egy felhasználó regisztrált, az implementált modulok közül saját maga választhatja ki melyik modult szeretné követni.

A téma céljai közé tartozik az emberi kapcsolatteremtés sokrétegűsége és lehetőség arra, hogy képet kapjunk arról, hogy milyen intenzitással használunk elektronikus eszközöket, és azon belül is pontos képet a saját érdeklődési körünkről, valamint adott modulokon belül kisegítő/automatizált rendszerrel segítsük a felhasználókat.

Mivel ez egy modulonként is nagyon sokrétű téma, így az implementálást magával az alapképzéses ötletem (TV show és filmek) funkciójával kezdtem meg. Ehhez is több szempontot figyelembe kell venni. Ez nem csak önmagában az adatot gyűjti össze a megtekintett tartalomról, hanem olyan plusz kiegészítőkkel jár, amely elősegíti a használatot is. Figyelembe kell venni, hogy a médiatartalom adatait valamilyen szolgáltatótól meg kell szerezni. Ezt viszont nem elég egy helyről begyűjteni, mert nem garantált a létezése csakis egy helyen. Továbbá ugyanez igaz felirat beszerzés esetén is. További funkciók a modulon belül lehetnek aktívak.

Egy aktív modul képes más aktivált modullal is kapcsolatba lépni, így ezek között kommunikáció, és további hasznos funkciókat tudunk beépíteni. Ezen belső funkciókról később írok.

IDE MÉG LEHET ÍRNI A TECHNOLÓGIÁKRÓL

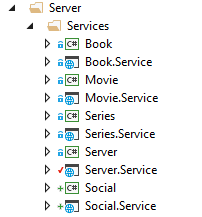
TÁRGYALÁSI RÉSZ

A leírásomat az alkalmazás fejlődési története, kutatása, implementációja időrendi sorrendjében fogom ismertetni, úgy, ahogyan az első lépésektől indult. Ehhez elsősorban arra van szükség, hogy minden az alkalmazáshoz tartozó technológiát megismerjünk olyan szinten, hogy azt a tervezett alkalmazásban használni tudjuk. A téma alapjául ASP.NET és C# programozási nyelv szolgál, amelyet korábbi projektjeim esetén már használtam, ezért választottam ezt a technológiát. Így nem ennek a megtanulása volt a szempont, hanem a projekt strukturális kiépítése. A strukturális felépítésnél a microservicek és adatbázisok elkülönítése, valamint ezeknek a felépítése volt a cél.

Az adatbázisok felállítása előtt tesztelésre került a Web API-k használata. Ebben a tesztben különböző HTML kérések helyességét ellenőriztem le, hogy milyen módon tudjuk ezeket kezelni. Például, hogyan lehetséges adott endpointon különböző route kéréseket megvalósítani.

Ezeken keresztül hogyan tudok adatokat továbbítani, visszaadni, és ezekkel további műveleteket végezni.

Ezek a műveletek egy .NET Core szerverre futnak be, ami egy microserviceként értelmezhető (.NET Core 2.1 + .NET Framework 2.0 együttese), s itt történnek meg azok a hívások, amelyek kielégítik a beérkezett kérést. Ez a kérés a szerver után egy .NET Framework projekten keresztül kerül további használatra, ahol a lényegi műveletek hajtódnak végre. A szerver maga a hívások fogadására és a végleges információ – amit a Framework projekt szolgáltat - visszaadására fog szolgálni. Adat manipuláció itt nem történik, csak a Frameworkben.



1. ábra Microservicek struktúrája

NOSQL rész

NoSQL és MongoDB bemutatása

Miért választottam?

Azért választottam relációs és nem-relációs adatbázist, mert úgy szerettem volna kialakítani a tárolást, hogy azokat az adatokat, amelyeknek nem tudok kötött modellt megadni, azokat mindenképp egy nem-relációs adatbázisban tároljam le.

Ahhoz, hogy ezek létrejöhessenek, egy üres API alkalmazás alapján próbáltam meg adatbázis kapcsolatot kiépíteni. Az adatbázis kapcsolat létrehozásához is kétfajta kapcsolatra van szükség, mivel két különböző adatbázisról van szó.

Mivel a mesterképzés során megismerkedtem a MongoDB adatbázissal, és azonnal meg is szerettem a használatát, így nem volt kérdés, hogy ezt a technológiát szeretném alkalmazni. Ezt az adatbázist egy olyan tárolónak szentelem, amibe olyan adatok kerülnek, amiből nagyon sok eset fog előfordulni egy kötetlen modell alapján. Ez tárolható lehet egy relációs megoldásban is, viszont a NoSQL ilyen körülmények között történő használata vezetett ehhez a megoldáshoz.

Az adatbázis kezeléséhez a Robo 3T nevű alkalmazást használom az adatreprezentációhoz. Itt a MongoDB futtatása mellett (melyet Command Prompt-ban szükséges futtatni) létrehoztam magát az adatbázist és a szükséges kollekciókat.

A Visual Studion belül a Mongo használatához NuGet Manageren belül szükséges telepíteni a MongoDB.Driver kiegészítőt, amellyel a szükséges annotációkat megadhatjuk a modelleken belül.

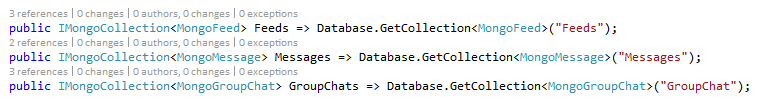
Miután ez sikeresen megtörténik, az adatok tárolásához való struktúra felépítése a következő lépés. A tároláshoz egy repository-t hoztam létre, amely önmagában csak az adatbázissal kommunikál és nem végez műveletet az adatokkal. Így az adatokat tárolja vagy kikéri az adatbázisból, és ezek alapján visszatér valamilyen objektummal, vagy értékkel.

A repositoryban lévő metódusok a kiterjesztett interface alapján kerülnek implementációra, amely megadja mely metódusokat kötelező tartalmazni az adatbázis kommunikáció.

A repository mivel csak az adatbázissal van kapcsolatban, ezért felsőbb szinten van egy olyan service osztály amely a kapott értékekkel műveleteket végezhet. Így egy olyan burkoló struktúrában van, ahol mindenkinek megvan a maga szerepe, és teljesül az az objektum-orientált szemléletmód, miszerint újra felhasználható bárhonnan a kapcsolat.

Ezek mellett szükséges az appsettings.json konfigurációs fájlon belül megadnunk a kapcsolathoz kötelező adatokat, melyek rendre a ConnectionString és az adatbázis neve.

Ahhoz, hogy sikeresen interakcióba léphessünk a MongoDB-vel az alkalmazásban, el kell érjük magát a létrehozott adatbázist. Erről egy külön osztály gondoskodik, név szerint a BaseMongoDbDataManager, amely a konfigurációs fájlból felolvassa a kapcsolathoz szükséges adatokat amelyeket az imént megadtunk, és a mongo driver segítségével egy új klienst hozunk létre belőle. Ezt a repository-n belül egy konstruktor az osztály létrehozása esetén önmagától megoldja, és az adatbázis használatra kész.

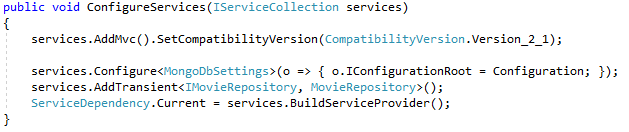
Ahhoz, hogy a kollekciókat elérjük, a repository-n belül beállítom a szükséges változókat, ahhoz, hogy használatra készen legyenek a későbbiekben. Erre szintén a driveren keresztül, az adatbázisból a GetCollection() metódussal tudok a kollekciókon műveleteket végezni. 

2. ábra Kollekciók beállítása a Social repositoryban

TRANSIENT?

Szükséges ugye beírni az addTransientet a Startup-ba. Az beállítja mint service a futás során. Ez egy dependency injection

Ez két típust vár paraméterül, az első paramétere annak a servicenek a típusa amit hozzászeretnénk adni a containerhez (minden olyan dolog itt található ami a futáshoz kell, ami egy szerveren rajta kell legyen, pl: kód, library stb), a második pedig az implementáció amit hozzá használni szeretnénk. Így az első az egy interface a második egy osztály lesz ami ezt az interfacet terjeszti ki.



3. ábra Service dependency injection a startup.cs fájlban

Ez a fajta implementáció már a javított változat, a jelenleg is használt forma. Az első megoldás egy ettől kezdetlegesebb verzióként indult, ahol a ServiceDependency.Current-be nem került be a service. Így egyszerűbb a használata, mert az adatbáziskapcsolattal kommunikáló repositoryt körülölelő osztályban egyszerűen a jelenlegi service hívásával tudom használni az interface metódusain keresztül az adatbázisműveleteket.

Ezzel elkészült a NoSQL kapcsolat, ezentúl használható az összes kollekció a repository által. A műveletek megtalálhatóak a MongoDB driver hivatalos dokumentációjában, viszont problémák adódtak a sima Find-al, talán?

SQL RÉSZ

MSSQL bemutatása

A nem-relációs adatbázis megoldását MongoDB használatával valósítottam meg, a relációs adatbáziskapcsolatot pedig MSSQL szerver segítségével.

Ahhoz, hogy az MSSQL használható legyen, az MSSQL service telepítésére van szükség, amely a számítógép indulásakor indulni fog, és használható amíg nincs leállítva. Az létrehozásához és annak műveleteihez a Microsoft által ajánlott Microsoft SQL Server Management Studio 18-at használtam. Az localhoston futó szerverhez adminként csatlakozva létrehozhatók az adatbázisok. Ezt követően elkészítettem az SQL szkriptet amely létrehozza majd az adatbázist, a megfelelő táblákat a hozzátartozó attribútumokkal, elsődleges és külső kulcsokkal együtt. Az adatbázisok felépítése a függelékben található.

A használatáról és az implementálásáról

Mivel a projekt nagy mennyiségű adatait a NoSQL adatbázisban tároltam, ezért a relációs adatbázis olyan adatok tárolására fog szolgálni, amely ritkán változik, mint például a regisztrált felhasználók, a hozzátartozó modulok, aktivált modulok adatai, azok az asztali alkalmazások, amelyek követhetők, valamint a követett alkalmazások felhasználókhoz rendelve.

Ahhoz, hogy kapcsolatba tudjunk lépni vele a forráskódból, szintén szükségünk lesz a másik adatbázishoz hasonlóan a ConnectionString-re. Ez a szolgáltatások közt futó service neve, a belépéshez tartozó felhasználónév és jelszó, valamint az adatbázis neve szükségeltetik.

Ehhez egy külön SQL kapcsolattal foglalkozó osztályban létrehoztam(BaseSqlDataManager) a kapcsolatot az adatbázissal, ehhez megadva a szükséges adatokat a ConnectionStringhez, amelyek szintén a NoSQL adatbázis mellett a konfigurációs fájlból olvasható be. A kapcsolat az osztály

Ahhoz hogy az adatbázison műveleteket végezzek, a MongoDB-hez hasonló felépítést választottam, miszerint a csak adatbázissal foglalkozó osztályt egy interface írja le, melyet a containerben a current serviceben tárolom, s ezt a servicet példányon keresztül használom. Ezen a példányon keresztül hívhatók azok a műveletek amelyeket el szeretnénk végezni.

A konkrét adatbázisműveletekre egy erre megfelelő osztályt hoztam létre, melyben minden SQL művelet megtalálható. Ez egy többszörösen leszármazott osztály, mely az alap SQL kapcsolatig visszavezethető. Felhasználhatók azok a metódusok, melyek a BaseSqlDataManager osztályban vannak implementálva. Ezek közé tartozik a GetConnection, GetCommand, GetParameter, ExecuteNonQuery. Ezek azok a metódusok amelyek szükségesek lesznek az adatok kezeléséhez.

Ezeket sorban ismertetem a következőkben.

GETPARAMETER

Ezt a metódust használva olyan lekérdezések írhatók, mely explicit mondja meg az SQL szervernek, hogy milyen paramétereket adtunk át, amely a lekérdezés újrafelhasználhatóságát segíti, továbbá biztosítja, hogy a paraméterben átadott érték biztonságosan van kezelve, így az SQL Injection támadás ellen biztonságot nyújt.

<https://www.mssqltips.com/sqlservertip/2981/using-parameters-for-sql-server-queries-and-stored-procedures/>

GETCOMMAND

A paraméterben megkapott lekérdezést alakítja SQL kompatibilis paranccsá.

EXECUTENONQUERY

Az előző két metódust használva létrehozza a kapcsolatot az adatbázissal, megformálja a kompatibilis lekérdezést a GetCommand metódussal, továbbá, ha paraméterek is vannak, azokkal együtt végrehajtja a lekérdezést az adatbázison.

A használatához az adott modulon/microservicen belül egy erre megfelelő osztályt származtatok az erre megfelelő BaseSqlDataManager-ből, s ezen keresztül végzem a kívánt műveleteket.

Egy lekérdezéshez elég csak a lekérdezés stringet megadni, majd az execute-al futtatni.

Új rekord beszúrásához szintén szükséges megadni az SQL scriptet, valamint azokat az értékeket, amelyek attribútumokként fognak szolgálni, majd ezt a scriptet az ExecuteNonQuery metódussal hajtjuk végre.

A MongoDB-be való beszúráskor a modelleknek szükséges tartalmaznia egy ObjectId-t, amelyet alapesetben a Mongo önmaga generál a UNIX idő, random szám és egy counter által. Ezt nekünk nem kell konkrétan megadni és generáltatni, viszont egy ID adattagot, és ezt annotációval felszerelni a következő módon, melyet a telepített mongo driver szolgáltat.

[BsonId]

[BsonIgnoreIfNull]

[BsonRepresentation(BsonType.ObjectId)]

Én a legtöbb esetben ezeket az annotációkat fogom használni, amelyeket bővebben leírok egyesével milyen hatást fejtenek ki.

A BsonId specifikálja, hogy ez az attribútum lesz az Id a tároláskor.

A BsonIgnoreIfNull az nem csak erre az egyetlen tagra vonatkozik. Ez gondoskodik arról, hogy ha valamelyik tag null értéket képvisvel, akkor azt nem veszi figyelembe szerializációkor.

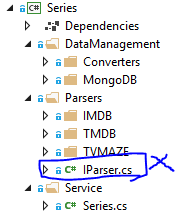
Az utolsó, a BsonRepresentation(BsonType.ObjectId) megköti a külső reprezentációját az attribútumnak, ez alakítja a Mongonak megfelelő ObjectId-ra az Id adattagot.

SOROZAT MODUL LEÍRÁS

Miután elkészültem az adatbázisok felépítésével, a projekt lényegi részének megvalósítása következik. Mivel a témám alapötlete a sorozattal és filmekkel foglalkozik, így ezt vélem a projekt magjának, ennek megfelelően ezzel kezdtem a tervezést.

Ennek első körben szintén a felépítésének megtervezése a feladat. Először szükséges volt eldöntenem, miszerint szeretném a sorozatokat kezelni, milyen modell alapján fogom tárolni, hogyan fogom ezekhez az adatokat szolgáltatni, majd később hogyan és melyik adatbázisban tárolom le. Ezek alapján a szempontok alapján először a sorozat tárolásáról döntöttem, amit MongoDB-ben fogok tárolni, mivel feltehetőleg elég sok új film és sorozat érkezik elég rövid belátható időn belül, ezek mellett folyamatosan változnak az esetlegesen megjelenő új évadok és új részek, vagy bármilyen frissítés esetén.

Mivel minden egyes request az úgynevezett Controllerek-en fut keresztül elsőként, így minden új funkció mindenekelőtti lépése a controllerben való megfelelő HTML request elkészítése. Ez az útmutató, innen fog megindulni minden adatkéréshez vezető út. A sorozatokkal kapcsolatos legfontosabb teendő, hogy valamilyen módon az adatbázisomban szeretném tárolni. Viszont ehhez nem előre kézzel megírt INSERT scriptek vannak előkészítve, hanem egy külső forrásból, valamilyen API-n keresztül kérem ki az adott sorozatot. Ezt import-ként használom az alkalmazásban.



4. ábra Series struktúrája

IMPORT MENETE

Ahhoz, hogy letárolásra kerüljön egy sorozat, a funkció controllerben lévő route-jának megfelelő szignatúráját követve, egy JSON objektummal meghívva –melyben egy string szerepel, mint cím- történik az importálás menete.

A sorozatok importálása feltételezve, hogy az egyik API nem tartalmazza az adott sorozatot, ezért másikon keresztül is szükséges lehet megpróbálni az importálást. Viszont ez nem jelenti azt, hogy ha az előbbin keresztül megtaláljuk akkor az utóbbin nem keresünk. Ha mind a két API (TvMaze, TmDB) rendelkezik a keresett médiával, akkor mind a két helyről betöltjük a megfelelő osztályba az attribútumokat a megfelelő adatszerkezetekbe.

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

1. public async Task ImportSeries(string title)

2. {

3. await IsSeriesImported(title);

4.

5. var tvMazeInternalSeries = await new TvMazeParser().ImportSeriesFromTvMaze(title);

6.

7. if (tvMazeInternalSeries != null)

8. {

10. var tmdbInternalSeries = await new TmdbParser().ImportTmdbSeries(title);

11. tvMazeInternalSeries.Merge(tmdbInternalSeries);

12.

13. tvMazeInternalSeries.Seasons = tvMazeInternalSeries.Seasons.OrderBy(x =>

14. x.SeasonNumber).ToList();

15.

16. await \_repo.AddInternalSeries(tvMazeInternalSeries);

17. }

18. else

19. {

20. var tmdbSeries = await new TmdbParser().ImportTmdbSeries(title);

21. await \_repo.AddInternalSeries(tmdbSeries);

22. }

23. }

A metódus paraméterében megkapja az adott sorozat címét, melyet felhasznál a későbbiekben. A 3. sorban egy ellenőrzés történik, hogy létezik-e az adott sorozat az adatbázisban. Ez egy bool értéket ad vissza, s eszerint haladunk tovább az importálásban.

Abban az esetben, ha megtaláltuk a keresett médiát, ugyanezt a folyamatot megismételjük a TmDB API hívásával is, majd objektumot építünk fel belőle. Ha nem találtuk meg az első helyen, akkor csak ez a procedúra fog bevégződni.

Ezt követően az egyik legfontosabb része az importálásnak az a két helyről érkezett sorozatok egyesítése, mely a 11. sorban fellelhető. Ezt követően a SeriesRepository –amely csak a MongoDB-vel kommunikál– letárolja az adatbázisban az adott sorozatot.

INTERNAL DOLGOK KIFEJTÉSE BŐVEBBEN

A Merge metódus bővebben:

Ahhoz, hogy tisztában legyünk a modellekkel, szükséges kifejtenem, hogy két nagy modellcsoport létezik a projektben. A letárolásra kerülő objektumoknak a saját microservice-ében saját, adatbázisuknak megfelelő, azzal kompatibilis modelle van. Ez azt jelenti, hogy MongoDB esetében az ObjectId-nak kötelező megjelenni a modellben, ezért szükség van egy külön adatbázis modellre, amely csak a letároláshoz szükséges. Ezért minden MongoDB-ben tárolt objektumnak külön modellje van csak a tároláshoz, és külön modellje van a kódon belüli és adott esetben servicek közötti használatra. Mivel ez egy belső modell, ezért ezek a modellek Internal előtaggal jelennek meg. Ezek az internal modellek ugyanazt a modellt reprezentálják, viszont nincsenek annotációhoz kötve, ezért bárki elérhető őket bármikor. Ezeket az Internal modelleket egy külön .NET Framework 2.0 projekt tartalmazza. Ennek a neve a projektben Internal.Contracts, s ezek használata történik minden belső műveletre, és csak a letárolás, valamit adatbázisból való felolvasáshoz használom az SQL és NoSQL modelleket.

Mivel a fenti kódban látható, hogy az internalSeries változón hívjuk a Merge metódust, ezért nyilvánvaló, hogy ez egy példánymetódus, amely elsősorban a TVMAZE-ről kapott adatokat részesíti előnyben, hiszen csak akkor kerül meghívásra, amikor itt (5. sor) megtalálja a sorozatot.

Mivel a két eredeti API oldalán eltérően szerepelhetnek egyes adatok ugyanarra vonatkozóan, így szükséges lesz azokat az attribútumokat is egyesíteni amelyekkel később valamilyen műveletet szeretnék végezni. A műfaj például egy ilyen eset tanúja, ami ugyanannál a sorozatnál teljesen különböző műfaj megnevezéseket kapott, s erre a későbbiekben épülnek funkciók az ajánlás rendszernél. Ezért arra is sor kerül, hogy a két műfaj listából, amely a két különböző API-n keresztül érkezett, a különböző műfaj neveket egy egységes listába teszem.  
Ehhez kapcsolódóan a műfaj tagok neve is teljesen különbözőként szerepel különböző oldalak szolgáltatásában, így én a Genre elnevezésűt preferálom és a későbbiekben ezt használom. Mivel valahol Categories néven fut, így a Genres és Categoriest az előbb említett módon egyesíteni fogom.

\*\*Első implementáláskor a Genre és Category külön szerepelt így duplikáltként ment, amíg nem kezdtem el használni az ajánlás rendszert.

Ettől nagyobb probléma a sorozatokhoz tartozó évadok egyesítése. A két sorozat évadai között egyesével szükséges végigmenni és az azonos évadokat olyan módon egyesíteni, hogy minden egyes tulajdonság az évadon belül lehetőleg feltöltve legyen. Ez úgy történik, hogy a két évad minden egyes attribútuma közül a nem null értékűt választom, amennyiben van ilyen.

INTERNAL REQUESTEK

INTERNAL EXCEPTION

INTERNAL ENUM

WEBCLIENTMANAGER?

\*\*\*\*\*\*\*\*\*\*\*\*

Ahhoz, hogy valamilyen adatokkal dolgozni tudjak, először is szükség van adatokra. Ezeket a TVMAZE és TMDB oldalak hivatalos API-ján keresztül szereztem meg. Ezekről olyan módon lehetséges adatok kinyerése, hogy egy megfelelő request segítségével a megfelelő helyre egy JSON típusú visszatérési értékkel egy adott objektumot kaphatunk.

Az API-k dokumentációja az a két weboldalon található meg, és az ezekhez szükséges endpoint hívások is példákkal együtt.

Ahhoz, hogy egy sorozatot modellé tudjunk alakítani a későbbiekben, az adatok kinyerésére kell fókuszálni. Ezeknek az adatoknak a lekérése nem mindig lehetséges egyetlen endpoint hívással.

Ez az eset áll fenn mind a két weboldalról történő letöltés esetén. Mivel egyik oldal sem ad olyan adat kikérési lehetőséget, hogy egyben ad ki információt egy megadott sorozatról, ezért a részekre bontott API hívásokkal kell kézileg összeilleszteni egy sorozat elemeit. Elsőként a TVMAZE alapján elemzem, viszont a TMDB teljesen hasonló felépítést használ, így a lényeges implementáció az első példa alapján triviális.

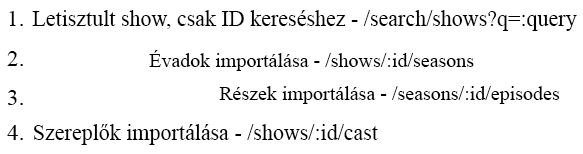
Mivel én név szerint szeretnék majd sorozatok importálni az adatbázisomba, ezért így szeretnék az API-n keresztül sorozatot kikérni. Megoldható lehet ID-val is, viszont névvel való keresésnek mindenképpen lennie kell, mert először tudnom kell mi az amit megszeretnék keresni.

Ezzel viszont csak egy részeredményét kapom meg a médiának, mert a TVMAZE által támogatott név szerinti keresés eredménye csak az alapértelmezett adatokat tartalmazza. Így nincs benne az évadoknak az információja, az évadokhoz tartozó részek és azoknak az adatai. Ezzel arra kényszerültem, hogy ezeket külön hívásokkal hozzácsatoljam a modellemhez. Ezt követően az évadok esetében is ugyanebbe a problémába ütköztem, mert az epizódok nem voltak beágyazva, ezért minden évadon belül egy újabb szinttel lejjebb kell menni, és importálni minden évadhoz a megfelelő részeket, ami egy újabb API hívás. A sorozat TvMaze-n tárolt ID-jával a részek együtt lekérhetőek voltak.

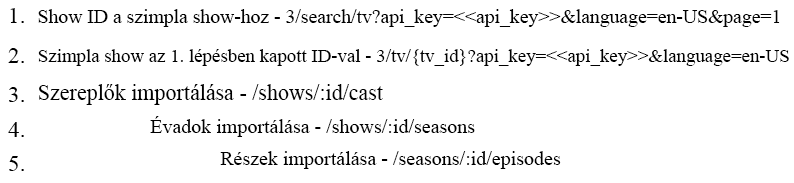
Ezt követően a meglévő adatokat egyesíteni kellett egy modellbe, amellyel megtörténhet a további feldolgozás.

~~APIKRÓL MÉG LEÍRÁST VISSZÁBB, VISSZAADÁSUK~~, MELYIK SOROZAT HOGYAN ÁLL ÖSSZE MODELLÉ?

Egy model a következő kérések után áll össze modellé



5. ábra TvMaze model felépítése



6. ábra TMDb model felépítése

Mielőtt egy sorozatot importálni szeretnénk, kötelezően ellenőrizni kell, hogy létezik-e a kért sorozat már a MongoDB-ben? Amennyiben nem importáltuk még, akkor az API-kon keresztül meg kell nézni, hogy valamelyik oldalon fellelhető-e? Abban az esetben, ha nem találtuk sehol, akkor tudatjuk a felhasználót, hogy a kért sorozat nem található.

Annotációk használata megjelenik több helyen is, mint például az adatbázis modelleknél, a controllereknél, az internal modelleknél stb. Ezekre a módosítókra azért van szükség, mert egyes esetekben megkönnyíti a munkát, máskor pedig kötelezően használni kell. Mind a kettőre példát fogok szolgáltatni. Ahhoz, hogy a sorozatokat az API-kon keresztül importálni tudjuk, valahogy a weboldalon keresztül kapott JSON objektumokat szükséges feldolgozni és értelmezni, ezeket pedig a megfelelő osztálytagokba betölteni. Erre egy generikus metódus ad megoldást, amely később tökéletesen felhasználható lesz, amikor ugyanezt a műveletet el szeretném végezni akár filmek esetén, ami szintén több helyről történik.

Ez a generikus metódus egy webkliens példányon keresztül a kért URL visszatérési értékét egy stringben tárolom. A deszerializációhoz a Newtonsoft csomagot szükséges telepíteni a NuGet Package Manager-ben. Ezen keresztül a JsonConvert.DeserializeObject<T>(data) a megadott típusra alakítja a data paramétert.

Visszatérve az annotációkhoz, az adattagok nevének kötelezően egyezni kell azokkal az attribútum nevekkel amelyek az API-n keresztül kapott JSON-ben szerepelnek. Ahhoz, hogy ne kelljen bonyolult elnevezésekkel dolgozni a T osztályokban (TmdbShow,TvMazeShow stb..) a tagok fölé megadva a JsonProperty annotációkat kiküszöbölhető az elnevezésekkel való probléma. Így továbbra is használható a C# által használt szintaxis.

A modellek építésénél fontos figyelni az attribútumok többszörös beágyazódására a JSON-ben, mert ezt a hierarchiát nekünk is követni kell az osztályok létrehozásának pillanatában. Az értelmezés eszerint fog folytatódni.

Az adatok megszerzéséhez az Internal.Core projektben egy ezzel kapcsolatos NetworkManager osztályt hoztam létre, ami csak ezzel foglalkozik, hogy az adatot lekérje az API-n keresztül és deszerializálja.

A sorozatok importálásához a jelenleg működő két helyen kívül több olyan API is létezik amely ingyen szolgáltat adatokat, ezek közül az elkövetkezőkben az OMDB-ről érkező sorozatok is használhatóak lesznek. Ennek az alapjai már le vannak fektetve a projektben, amit az egyesítéssel és a letárolással bővíteni szükséges. Ez a funkció jelenleg nincs használatba véve, viszont a későbbiekben bármikor hozzácsatolható.

A sorozatok műfajával kapcsolatban több problémába is ütköztem. Szerettem volna minden műfajt Enum-ként deklarálni és azzal dolgozni, de olyan enum szerializációs hibába ütköztem, amelyet a Mongo nem tud feloldani. Ezért ezt elhagyva stringként tárolom a műfajokat, viszont később találtam rá lehetséges megoldásokat, mely szerint annotációval megoldható, a lényege viszont ugyan ez, mert a tárolás ebben az esetben is stringként lesz kezelve. A különbség, hogy a felolvasás után az adott enum-ba tudja alakítani, ezt viszont annotáció nélkül is el tudtam volna végezni, így nem láttam szükségét a használatának.

MÉG NEM ÍRTAM AZ ISUPTODATE METHODRÓL AMI A TVMAZERŐL JÖN

+ TOVÁBBI FUNKCIÓK :

rate episode, season, series

comment on episode, comment on series

set favorite series, set fav episode

MOVIE MODUL

A sorozat elkészültét követően a filmekkel is foglalkoztam. Beépítettem azokat a lehetőségeket, amikkel a sorozat is rendelkezik. Eszerint az importálás, filmnek az adatbázisban való ellenőrzése, egy elkezdett filmnek az adott pozíciójának frissítése, eldöntése annak, hogy egy filmet elkezdtünk-e már, továbbá, látottnak való jelölése, valamint, hogy láttuk-e. Emellett lehetőség lesz a későbbiekben filmek ajánlására is és a felhasználói oldalon lekérhető lesz miket láttunk az elmúlt ’n’ napban. Ez már egy felhasználói funkcióként fog teljesülni, nem az automatizált részhez tartozik, hanem a felhasználói felület használatához.

Az importálás hasonlóan a sorozatok importálás módszeréhez, itt is több oldalról összeszedett filmek kerülnek az adatbázisba. Jelenleg két oldalról gyűjtött filmek lesznek egyesítve (Tmdb, Omdb). Az elsődleges oldal a TMDB, ami alapján a film feldolgozás történik. Először innen gyűjtünk adatokat, majd ehhez csatolom hozzá az OMDB-n szerzett információkat. Az egyesítés abban az esetben történik meg ha mind a két helyről sikeresen begyűjtöttünk adatokat. Amennyiben ez nem sikerült, akkor csak a TMDB-ről érkező adatokat tároljuk le (ha megtaláltuk a filmet). Az OMDB API-ján keresztüli keresés csak a pontos névmeghatározás esetén ad jó eredményt, így azt is vizsgálnunk kell hogy a két API-n történt keresés megegyezik, mivel itt fejlesztést megelőző tesztelés esetén is előfordul hibás eredmény, az API-ban történő keresés hiányossága miatt. Ez egy kiegészítés, ami megmutatja azt, hogy itt is lehetséges az a módszer, amit a sorozatok esetén használtam. Az egyesítés folyamata megegyezik, így ezt nem részletezem. Az oldalakról kapott eredményt Internal modellekben tárolom, ezért ezeket tároláskor a Mongonak megfelelő modellre alakítom. Emellett szükséges megnézni azt, hogy az adott film esetleg nincs-e importálva már az adatbázisba, így ennek megfelelően kell eljárni. Erről egy hibakezelés folyamán response-ban a megfelelő hibakóddal és üzenettel térek vissza.

Mivel ez a modul később kapcsolatban fog állni egy asztali alkalmazással, amely a kliens gépén fog futni, ezért azok a funkciók amelyek erre lesznek kihegyezve, ide kerülnek implementálásra. Ezek közül például az elkezdett filmeket adott felhasználónak automatikusan bejelöljük és tároljuk a látottságát az adott filmnek. Erre szolgál az UpdateStartedMovie metódus, ami figyelembe veszi, hogy esetleg láttuk-e már a filmet, ha még nem, akkor milyen mennyiségben láttuk. Egy filmet látottként értelmezek amennyiben 95%-ban végignéztük a tartalmat. Efölött az adatbázisban való tároláskor a befejezett filmek közé kerül. Ellenkező esetben a felhasználó elkezdett filmeihez fogjuk beszúrni a jelenlegi adatokkal, ami a média lejátszó bezárásakor lévő adatokat jelenti (hány másodpercet látott, időpont amikor a jelölés kérése érkezett, stb..). Ebben a metódusban fogjuk hasznát venni az importálásnak, mivel egy importálás akkor történik meg valójában, amikor egy felhasználó egy olyan új tartalmat néz, ami még nem létezik az adatbázisban tárolva. Amennyiben befejezünk egy filmet, akkor az a felhasználó profiljából a megkezdett filmek közül törlődésre kerül, és átkerül a befejezettekhez.

Továbbá a webes felületen elérhető funkciók közé tartozik az értékelés lehetősége. Ezt a felhasználó a saját filmjeire és sorozataira tudja alkalmazni úgy, hogy tetszése szerint egy skálán értékelni tudja az adott tartalmat.

KÓDBAN TEGYEM AZ ELEJÉRE

+ ESETLEG A LEÍRÁSBAN KÜLÖN LEHETNE SZEDNI HOGY AUTOMATIZÁLTSÁGRA ÉPÜLŐ METÓDUSOK : MarkAsSeen, MarkAsStarted, stb stb

ÉS FELHASZNÁLÓI FUNKCIÓKRA ÉPÜLŐ METÓDUSOK : Rate, GetLastDays

BOOK MODUL

SERVER MODUL

SOCIAL MODUL

RECOMMEND RENDSZERRŐL

Az alkalmazás automatizáltsága alapján a modulok kommunikálhatnak egymással amennyiben a megfelelő modulok ehhez aktiválva vannak. Ezek a modulok ilyen esetben nem folyamatos kapcsolatban vannak. A különböző microservicek között web requestekkel történik a kommunikáció. Mivel minden service külön szerverként van jelen az alkalmazásban. Amennyiben a felhasználó úgy dönt, hogy együtt használja ezeket a modulokat, plusz funkciók is megjelennek.

Vannak csak adott modulon belüli ajánló rendszerek, és vannak olyanok, amelyek más modulon keresztül érkeznek a felhasználóhoz. Először vegyük figyelembe a modulon belüli, amihez nincs szükség más aktivált service működésére. Ezeket a modulokat a későbbiekben ismertetett asztali alkalmazás (kliens alkalmazás, DesktopClient) által vehetjük használatba, vagy tilthatjuk le őket.

Az egyéni ajánlások olyan funkciók, amelyek a saját modulon belül végzik a feladatukat. Ezek úgy működnek, hogy ha a felhasználó aktivál egy modult, ahhoz nincs szüksége másra az alkalmazásnak, az ajánlás rendszer automatikusan használható. Ez később kikapcsolható a kliens programban az igényesség érdekében.

FILM RECOMMEND

Hasonló működést tanúsít a film serviceben a Recommend metódus. Ez a saját filmes érdeklődési körünkből fog olyan filmeket ajánlani, amelyeket a legtöbbet nézünk a műfajok közül. Tehát egy műfaj alapú ajánlás történik. Ez nem csak a befejezett filmekből tevődik össze, hanem a már megkezdett filmek is csatlakoznak az ajánlás elkészültéhez. Ezt követően duplikáció nélkül összeszámolom melyik műfaj hányszor szerepel, majd ezeket csökkenő sorrendbe rendezve a három legtöbbet nézett műfajok közül történik az ajánlás hasonló kategóriában. Az ’n’ legtöbbet nézett ajánlás paraméterezhető, viszont alapesetben az n a 3 értéket fogja felvenni.

SERIES RECOMMEND

A filmeknél történő ajánlás megtalálható a sorozatoknál is. Olyan sorozatokat ajánlok, amelyet a felhasználó még nem kezdett el, vagy nem vett fel a listájára. Itt is azokból a műfajokból ajánlok, amit a felhasználó elkezdett, vagy már befejezettek alapján. A kettőt együtt vizsgálom, és szintén csökkenő sorrendben megnézem, melyek azok a sorozatok amelyek a legnézettebbek. Itt is az első háromból próbálunk ajánlani. Azonban nem lehetséges az ajánlás, ha az adatbázisban nincs ennek megfelelő mennyiségű sorozat ebben a műfajban. Ebben az esetben a háromból annyit ajánlunk amennyit tudunk.

Ezeket az ajánlásokat több helyen is alkalmazhatjuk. Akár a webes felületen, akár az asztali alkalmazás is tud ajánlani egy megtekintett film vagy sorozat után a hasonló kategóriában. Ezek akár megjeleníthetőek lehetnek a Formon belül, egy listában, melyből a felhasználó válogathat, így a webes felületet sem kell használnia.

Sorozatokra vonatkozóan kétféle ajánlás is implementálva van. Ez a kettő ágazat a legtöbbet nézett műfaj és előre megadott műfajok listája alapján való ajánlás. Ezek egyarán használhatók a webes felületen történő ajánláshoz. Az előbbi az egy személyre szabott ajánlatkérés, ami a felhasználó által preferált műfajok közül válogat, míg az utóbbi például az oldalon a sorozatok közt böngészve, egy adott sorozathoz hasonló sorozatokat tud ajánlani. Ehhez az éppen böngészett sorozat műfajainak listája szükséges, s ezekből dolgozik a RecommendSeriesByGenre metódus.

Ehhez hasonló módon működik az az ajánlás rendszer, amely az MPCManager-be van beépítve. Ez szintén konfigurálható lehet a DesktopClient-ben, amennyiben nincs rá szüksége a felhasználónak. Ez egy külső modullal történő kommunikáció, mivel az MPC modulból egy másik modult veszünk igénybe. Ez úgy működik, hogy amikor a felhasználó végignézett egy sorozatot, és az ajánlás be van kapcsolva (alapesetben be van, továbbá ez a későbbiekben tovább fejlesztésre vár az imént említett konfigurálhatósággal kapcsolatban) valamint az a modul, amelybe a kliens a kérést küldeni szeretné, akkor az ajánlás elérhető lesz. Ez a sorozatok és filmek esetében egy könyvadaptációnak az ajánlását teszi lehetővé. A médiatartalom befejezte után, a bezárást követően, abban az esetben ha a megfelelő modulok aktívak a rendszer könyvet tud ajánlani a megtekintett sorozatból. A könyvek külön modulban helyezkednek el, és külön kollekcióban vannak letárolva. A könyvek ajánlása az adatbázisban tárolt könyvekből történik, így előbb, ha létezik a sorozatból könyvadaptáció, valakinek olvasnia kellett a könyvet, hogy bekerüljön az adatbázisba. Jövőbeli tervek között szerepel, hogy a könyvek ajánlása ne csak cím alapján történjen, hanem a felhasználók a webes felületen böngészhessenek a tárolt könyvek között, s ezeknek referenciaként megadhassanak sorozatokat, amelyekhez kapcsolódik. Így sokrétűbbé válhat az a módszer, ami alapján a kiválasztás történik.

KÖNYVEK AJÁNLÁSÁT NEM IS ÉRTEM MI A SZAR VAN. ÚJRATESZTELÉS SZÜKSÉGES.

KÓDBAN IS BELEKELL TENNI A VIZSGÁLATOT A SERIES RECOMMENDBEN HOGY HA NEM TUDUNK 3AT AJÁNLANI AKKOR NE TUDJUNK EZ A LINQ-NÁL A LEGVÉGÉN VAN

CONVERTEREKRŐL

Mivel elég sok modellel rendelkezik a projekt idáig, s ezeket valahogy kezelni kell olyan téren, hogy kompatibilisek legyenek egymással, ezért egy módszert ki kellett fejlesztenem, amivel egyszerűen és gyorsan tudok modelleket konvertálni. A korábbiakban már taglaltam az úgymond belső és külső modelleket, ezek rendre az Internal és Mongo modellek. Az SQL adatbázis tárolásához nincsen szükség modellekre, mivel ott paraméterizálva fogunk eljárni SQL lekérdezésekkel, így felesleges a használata.

Azok szerveződéséről

Ezeket a konvertereket minden egyes különálló modulba szükséges beépíteni a saját modelljeire való tekintettel. Ezek használata csak itt lesz kötelező, mivel az adatbázis tárolás / felkérés esetén Internal modellre váltok a könnyebb használat miatt. Két fázisa volt a konverterek állapotának. A kezdetleges változat az egységesen tartalmazott egyik irányból a másikba és fordítva minden átalakítást. Később ez formálódott át egy szeparált mindkét irányt külön elérhető konvertáló osztályokba.

Azt fontos tudni róla, hogy InternalSeriesből készít MongoSeries vagy épp fordítva. S ezeken belül nyilván az összes adattagot is konvertálni szükséges, az adatszerkezeteket, az azon belül elhelyezkedő osztályokat stb. Minden egyes elemet egy a projekten belül általánosan elfogadott, bárhol használható modellé alakítom.

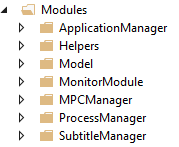
DESKTOPCLIENT

A szakdolgozat sorozatkövető moduljának egyik alappillére már megvan ebben a pillanatban. Mivel már tudtam sorozatokat importálni így elkezdtem azt az asztali alkalmazást implementálni, amivel az asztali gépen futó alkalmazásokat fogom monitorozni.

Ennek a neve a DesktopClient alkalmazás lesz, amibe több különböző modul és kezelő lesz beépítve. A modulok ezen a szinten is bővíthetők. A felhasználónak a login után a DesktopClienttel nincs dolga, interakcióba nem léphet vele, mivel ez csak egy úgynevezett serviceként van jelen, mint egy háttérfolyamat amely az adatokat feldolgozza és lekommunikálja a szerverrel. Ez a későbbiekben változni fog, mivel felhasználói interakcióhoz szükség lesz, ha adott modulokat használni szeretne. Ezek elsősorban konfigurációs fájlok beállítását jelentik.

Miután a login megtörtént, a szerver és a kliens egymással információkat oszthat meg, ezzel fenntartva a valós időben történő frissítéseket a webszerver felé. A legfontosabb része ennek az asztali kliensnek és a szervernek az, ahogyan a modulok be és felépülnek. Így a login rendszer ezúttal a háttérbe kerül és a modulokra fókuszálok. A DesktopClient több fontos feladatot is fog végezni, ezeken fogok keresztül menni mélyebb leírásban a következő fejezetekben.

MPC MANAGER



7. ábra DesktopClient mappa struktúrája

A DesktopClient a Media Player Classic Manager-t (MPCManager) valósította meg először. Ez a kezdeti modul, ami bekerült, és olyan funkciókat lát el ami már a microserviceket használja.

A kliens egy Windows Form projektként készült el, mert a jövőben tervek között szerepel, hogy valamilyen UI-t kapjon, amivel a felhasználó interaktálhat.

Az MPC Manager terve alapvetően a megtekintett média alapján importál adatbázisba, jelöl látottnak adott részt, figyelmeztet kihagyott rész / új rész esetén, ajánl könyvadaptációt, szerez be megfelelő időzítésű magyar feliratot idegen nyelvű média esetén. Ezen funkciók megvalósításához legfontosabb lépés, hogy képesek legyünk megmondani milyen tartalom az amit a felhasználó éppen néz.

Mivel ez kikerülhetetlen lépés, és nincsenek erre technológiák, így ezt manuális módon kell megoldani, a médiatartalom címének valamilyen feldolgozásával. Itt viszont felmerül a kérdés, hogy ezeket milyen módon próbáljam meg kinyerni. Az első ötlet nyilván a futó folyamatok között való keresés valamilyen módon, majd az ablak címének kinyerése. Ez az ötlet viszont hamar kudarcba fulladt, mivel a médiacím nincs kötelezve, hogy csak a tényleges címet tartalmazhatja, akár teljesen más névvel is rendelkezhet. Ennél több esélyünk lehet, ha megtudnánk nézni milyen mappaszerkezetben található az adott fájl ami éppen fut, mert elképzelhető, hogy csak a fájlnév az ami teljesen különbözik a tényleges tartalomtól. Ezzel még továbbra sem száz százalékos a siker, de jobb esélyekkel indulunk ahhoz, hogy eldöntsük a média címet a webszerver import metódusának kötelező paraméterét. Az információ elég kevés, viszont feltételezem minden médiánál, hogy a mappaszerkezetében vagy a fájl nevében megtalálható a megtekintett tartalom címe. Ezen a gondolaton továbbhaladva folytattam az implementálást, viszont a folyamatok lekérdezése továbbá nem opció, mivel nem ad elég információt a folyamat útvonaláról. Így egy kis kutatás után kiderült, hogy a Media Player Classic és a VLC Player is rendelkezik beépített web interface-el, ami lehetővé teszi, hogy minden elérhető információt, amivel a lejátszó rendelkezik, kitelepítsük egy szerverre. Ez tartalmaz minden olyan információt, amit az előzőkben taglaltam.

MÉDIA BEFEJEZÉSEKOR VALÓ MENTÉS ÉS BEJELÖLÉSRŐL ÍRNI

A működésről

SOROZAT

A folyamatos háttérben történő figyelés egy végtelen ciklussal történik külön szálon, ami a memória igényt lecsökkentve egy késleltetéssel dolgozik, így minimálisra csökkentve a memóriahasználatot.

Az MPCManager segítségül veszi a futtatott folyamatok listáját, amivel eldönti, hogy az adott média lejátszó elérhető lesz-e. Miután ez megtörténik **beállítom azokat a változókat amelyekkel dolgozni szeretnék (útvonal, fájlnév-amennyiben nem üres-, cím-a fájlnévből-, epizód, évad, releaser, minőség). Abban az esetben ha van cím, azt az osztályon belül adattagként elmentem, hogy később újrafelhasználható legyen. Erre még kitérek bővebben, előbb haladjunk a működés szempontjából.**

Miután sikerültek a szükséges változók beállításai, el kell döntenünk, hogy filmről van sorozatról beszélünk, mivel ez a későbbiekben kezelni fogja a filmeket is. Ezt egyszerűen úgy tettem meg, hogy ellenőriztem, hogy valahol tartalmazza-e a fájl azt hogy hanyadik évad és epizódról van szó (S01E01). Ennek továbbfejleszthető változata az, ha csak a címet tudjuk, akkor az API-n keresztül ellenőrizhető a tény. Ebben az esetben ettől eltekintek, a cím vagy a mappák egyike kötelezően tartalmaznia kell azt, hogy milyen évad és epizód.

Majd ezt követően, ha a lejátszó futó állapotban van, akkor aszerint cselekszünk, hogy mit döntöttünk el az előző lépésben. A mostani lépés egyetlen alkalommal fog futni, ez pedig a lejátszó indításakor történik meg. Ennek megfelelően fogunk lépni, -most a sorozat oldaláról tekintem először-. Amennyiben ezt a sorozatot a felhasználók közül még senki sem kezdte el nézni, így automatikusan a kliens import requestet kér a server felé és adatbázisban kerül letárolásra az új sorozat. Abban az esetben ha valamilyen hiba történt, a megfelelő kivételkezelések szolgáltatnak információt, vagy éppenséggel a felhasználó kap asztali értesítést, amennyiben az rá hatással van.

Továbbra is abban az állapotban vagyunk, hogy most indult el a médialejátszó, és megtaláltuk a sorozatot az adatbázisunkban, vagy be lett sikeresen importálva. Ezt követően, ha nem anyanyelvű médiát néz a felhasználó (amit szintén a névből derítek ki, hogy ha tartalmazza a ’hun’ karaktersorozatot valahol a cím vagy fájlnév, akkor idegen nyelvű) megkezdődik egy újabb nagy integrációs lépés, ami nem más mint a magyar felirat beszerzése az adott sorozathoz. Ennek sok lépése van, ezt szintén bővebben kifejtem a következő alfejezetben.

Ha nincs még felirat az adott epizódhoz, és megtaláltuk a megfelelőt, ezt úgy vesszük használatba, hogy ugyanazt a nevet kapja meg a feliratfájl mint maga a médiafájl. Viszont ezzel a jelenleg futó lejátszó nem tudja alkalmazni a feliratot, így azt választottam, hogy újraindítom lejátszót és a felirat automatikusan megjelenik. Mindez másodpercek alatt történik, s már megfelelő felirattal nézhető az adott média.

FILM

Mivel a filmek elég hasonlóak így ezt nagyobb léptekben fogom bejárni az elejétől hogyan működik. Ehhez vissza kell menni ahhoz a lépéshez, hogy eldöntöttük, hogy sorozatról van-e szó. Ez ennek az ágnak a másik része, ebben az esetben a filmekkel kapcsolatos folyamatok fognak lezajlani. Itt különbségek merülnek fel már csak a címből kiindulva is, és annak szerkezetéből (ezt a következő alfejezetben szintén bővebben fejtegetem) is. Miután kiderült, hogy filmről beszélünk, a ManageMovie metódus gondoskodik a film állapotának követéséről, valamint a feliratok beszerzéséről, továbbá az importálásról. Annyiban változik az első lépés, hogy az importálás elég kerül egy olyan vizsgálat, hogy esetleg láttuk-e már az adott filmet. Ezt a felhasználói profil oldalán megtekinthető, adatbázisból kideríthető információ. Amennyiben a felhasználó nem látta még, így kezdődik meg az importhoz kapcsolódó ág, amely teljesen megegyezik a sorozat mintájához. Miután előfordulhat olyan, hogy korábban már belekezdtünk egy filmbe, vagy már láttuk, ezért a felhasználó ezekről tájékoztatást vagy úgynevezett figyelmeztetést kap, amelyet követően tovább folytathatja a médiát.

A felirat letöltése szintén nagyban megegyezik a sorozatokban implementált kereséssel és letöltéssel. Itt a szűrés a cím, minőség, releaser alapján történik. A jelenlegi állapotában, mivel a feliratok között több is kerülhet vissza ezzel a keresési szűréssel, a legelsőt alkalmazom majd a médiához. Ezt viszont miután a Form alkalmazás alakot ölt, visszaadható lehet az összes szűrésnek eleget tevő felirat, így a felhasználható szabadon dönthet a meglévők közül melyik szeretné igénybe venni.

Felirat manager

A feliratok kezelése a sorozatok és filmek esetében külön-külön segítő osztályban vannak tárolva. A feliratokat saját kézzel kellett kikeresnem, itt nem használtam hivatalos API-t, mert nem rendelkezik az adott oldal ilyennel. Ezeket a [www.feliratok.info](http://www.feliratok.info) weboldalon lehet megtalálni, ahol szűrhető kereséssel szűkíthető a lista. Ezen kívül megadható, hogy milyen nyelven szeretnénk kikérni, melyik évad melyik epizódját.

Használok egy külső osztályt, melyet a MovieSubtitleManager és SeriesSubtitleManager mappákban lévő osztályok használhatnak. Ez a közös osztály a SubtitleFetcher. Ebben implementálva van az adott oldalról történő feliratok listájának összeszedése.

Mivel több oldalas lehet a feliratok listája, ezért a feliratok megszerzése egy while ciklusban történnek, amelyek addig hajtódnak végre amíg már nincs több oldal.

Az összeszedéshez XPATH-ot használva egyszerűen minden olyan HTML tagot hozzáadom egy listához ami HtmlNode-okat tartalmaz. Minden ilyen node egy darab feliratot jelent, a bennelévő címmel, leírással, letöltési linkkel, dátummal, feltöltővel.

Miután ez a lista elkészült, ezek között keresve a FindAdherentSubtitleAndDownload metódus fogja megkeresni a megfelelő feliratot, majd letölti, átnevezi és mappáz ha szükséges. Ezekre azért van szükség, mert a sorozatok több formában is elérhetőek, ezért több módon kell készülni a felirat kezelésére. A legegyszerűbb eset, amikor a sorozat egyetlen epizód önmagában, s ehhez létezik egy olyan felirat, ami pontosan ugyanehhez a részhez, ugyanolyan releaser és ugyanolyan minőséghez készíttetett.



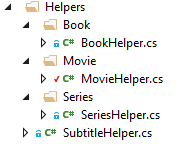
8. ábra Felirat felépítése

Az ábra szerinti **The Walking Dead – 9x12** lesz a cím. Az ez alatt elterülő szöveg pedig a leírás. Mind a kettő fontos szerepet fog játszani, mivel itt találhatóak a releaserek és a minőséghez tartozó adatok. Ez azért fontos, mert egy felirat adott epizódhoz eltérhet egy másiktól, ami ugyanehhez az epizódhoz tartozik. Szükséges ugyanazt a releasert és minőséget keresnünk majd, amit a médialejátszó is játszik a felhasználónál.

REGEX ALFEJEZET

Ezekkel az adatokkal már rendelkezünk, mivel beállítottuk őket a médialejátszó indításakor a web interfaceről kikért adatokkal. Most nézzük meg, hogyan lehet ezeket az adatokat kinyerni egy stringből, hiszen mindig változni fog, nem csak a címe és az értékek, de akár a felépítés is változhat minimálisan. Általában egy megadott elnevezési formát követnek a sorozatok, viszont nem mindig teljesen pontos. Így szükség van egy olyan szövegfeldolgozásra, amely lehetővé teszi, hogy ezeket könnyedén kinyerhessük belőle. Erre a legmegfelelőbb technológia a regex használata. Ehhez fel kellett idéznem, és gyakorolnom milyen lehetőségek vannak, mi a szintaktika és milyen logika alapján működik.

A regex használatához egy külön helper struktúrát építettem fel, ami minden modulhoz külön-külön segítséget tud nyújtani. Ezen kívül vannak közös metódusok, amiket a SubtitleHelper szolgáltat.



9. ábra Regex mappastruktúra

A SubtitleHelper két fontos metódust tartalmaz, ami a GetQuality és GetReleaser. Ez a két metódus fogja bármilyen szövegkörnyezetből felismerni a minőség és releaser nevét. Ez működni fog a filmeken is, mivel ott is megvannak ezek az adatok. A SeriesHelper ezt egészíti ki metódusaival

SERIESHELPER

Itt történnek meg a sorozat specifikus regex műveletek: GetTitle, DoesItContainHun, GetEpisodeFromFeliratokInfo1x2, GetSeasonFromFeliratokInfo1x2, GetSeasonFromFeliratokInfoEvad, GetSeasonFromFeliratokInfoThird, GetSeasonNumber, GetEpisodeNumber, DoestItContainSeasonAndEpisode, DoesItContainSeasonAndEpisodeS01E01.

A SubtitleHelper metódusai eleinte itt kaptak szerepet, melyek később a projekt bővülésével, ahogy új modulok kerültek be, úgy szerveződtek ki közös osztályba.

A SeriesHelper-ben helyet foglaló regex-el foglalkozó metódusok egy adott szövegben csoportokon belül kiragadják a megadott minta alapján a kívánt részletet. Ennek nagyon specifikus leírás szükséges. Ahhoz, hogy egy sorozat címét, évad és epizód számát, releaser-ét mindenféle kézi string feldolgozás helyett kinyerjük, ennek a megoldása könnyed és gyorsan használható. A regex minta amire jelenleg használható a kliens, az a következő cím példákra teljesül:

Veep.S06E01.720p.HDTV.x264-KILLERS.mkv

Modern.Family.S10E06.1080p.AMZN.WEB-DL.DDP5.1.H.264.HUN.ENG-PTHD.mkv

Death.by.Magic.S01E08.1080p.WEBRip.X264-INFLATE.mkv

Ezekre a példákra a használt minta felismerési csoport szerint

Átfogó csoportok

1. 2. 3. 4. 5. 6.

^(.+?)[. \_-]+?s(\d+)[. \_-]\*e(\d+)\.(1080|720[pi])[. \_-]\*.+?([^-]+)?\.(mkv|avi|mp4|srt)?$  
 cím évad és epizód minőség köztes releaser kiterjesztés  
 szöveg

Konkrét csoportok

^(.+?)[. \_-]+?s(\d+)[. \_-]\*e(\d+)\.(1080|720[pi])[. \_-]\*.+?([^-]+)?\.(mkv|avi|mp4|srt)?$

Ennek a fenti példák közül egyre gyakorolt hatása a következő:

Modern.Family.S10E06.1080p.AMZN.WEB-DL.DDP5.1.H.264.HUN.ENG-PTHD.mkv

A regex minták elejét és végét két karakter jelzi, ezek rendre a ’^’ és ’$’ jelek. Ezek közé kerül az a minta, mely a feldolgozandó string-et nézi át.

1. csoport

Ezt a mintát részekre lehet bontani, ami felparszolja ezeket a címeket. A főbb részeket ki fogom emelni külön. Mivel a feltételezés, hogy az összes cím ezt a felépítést tartalmazza mint a fenti három, így a cím önmaga az elején lesz található. Az erre vonatkozó csoport a nagy S betűt követő számokig tart. Ez írja le a címet, amely a mintában citromsárga kiemeléssel van jelölve. Ez olyan karaktersorozatot jelent, amik között megjelenhetnek a pont, aláhúzás és kötőjel elemek, s ebből több is előfordulhat, így megadva a lehetőséget a többszöri előforduláshoz, abban az esetben ha nem egy szavas a cím.

1. csoport

Az évad és epizód értékének megjelenése elengedhetetlen ahhoz, hogy a sorozatokkal kapcsolatos műveleteket végre tudjuk hajtani. Valamint azért is, mert ha ez nem jelenik meg akkor, mint már említettem feltételezzük, hogy filmről van szó. A cím felismerése után következnek ezen értékek egy nagy vagy kis ’S’ betű és szintén kis vagy nagy ’E’ betű után. Lehetséges, hogy a két érték csak 1 karakterből áll, így ezt biztosítani kell, s ezek decimális számokként szerepelnek. Ennek a csoportját világoskék színnel emeltem ki a mintában. Az ebben a főcsoportban lévő konkrét csoportok a konkrét évad és epizód szám, amely narancssárga, és sötét sárga színnel kiemelt.

1. csoport

A minőség csoportban felsorolható több érték is logikai vagy-al elválasztva. Ezt sötét lila színnel jelöltem.

1. csoport

Ez a csoport a szöveg végéig tartó karaktereket takarja, ide a kötőjelig tartó szövegrészlet kerül, itt általában kódolási technológiák neve kerül és audiósáv információk. Ebből nekem a ’HUN’ részlet lesz fontos később, viszont ezt külön regex-el kezelem. Ezt a továbbiak során még szükséges ide beépíteni.

1. csoport

Ez egy olyan csoportot fog keresni a kapott szövegben, amely egy kötőjel után szöveget talál, amíg egy ponttal nem végződik. Ezt egy opcionális csoportként kezelem, így ha nem szerepel benne, a minta felismerés továbbra is működik, viszont ez a csoport nem lesz elérhető.

1. csoport

A fájlok legvégén szerepelnek a kiterjesztések, ezért ezt logikai vagyok között felsorolva keresem. Az opcionális ’?’ karakter adja meg itt is annak lehetőségét, ha mappa névről van szó, akkor ott kiterjesztésről nem beszélhetünk, így a mintafelismerés azon is működőképes.

Emellett a címre vonatkozó minta mellett készítettem csak az évad-epizód párosra is felismerést. Ezt a sorozat eldöntéshez használom, mivel a csoportokra való hivatkozás változhat, amennyiben nem létezik minden csoport a fent említett minta által, így nullreference kivételt kapnánk.

A 8. ábra alapján egy újabb regex minta készült az epizód felismerésre. Itt a korábban címmel foglalkozó mintától eltérően másként történik a leírása. Ezért erre külön az évad és epizód közt megjelenő karakterrel történő felismerést készítettem el, amelyet a feliratok.info oldalon fellelhető feliratokra fogom alkalmazni a felismeréshez.

MOVIEHELPER

A filmek esetében a regex a következő módon alakul. Mivel itt általában a fájlok neve nagyon kevés esetben tartalmazza a szükséges információkat, ezért ezt a mappanevekből fogom megszerezni. A módszer viszont ugyanaz mint a sorozatok esetében. A mint alapján a kapott cím a feldolgozást követően csoportokba szerveződik. Három csoportot különböztethetünk meg. A címet, az évet és a releasert. A minőség megszerzése külön történik, a SubtitleHelper osztályban.

1. 2. 3. 4.

([ .\w']+?)\W(\d{4})\W(\d{1}D)?.\*(HUN)?-([^-]+) na ez még totálszar így

cím év 3D releaser

IDE MÉG LEHET PÉLDÁT TENNI HA SZÜKSÉGES

A MovieHelepr továbbá elvégzi majd az adott médiának a pozíciójának lementését az adatbázisba, ezért ezt szintén egy átfogó osztályba kellene rakni, mivel a sorozat is ezt használja.

FELIRAT LETÖLTÉS

3 ESET LEHETSÉGES

PONTOS EGYEZÉS

Visszatérve a feliratok letöltéséhez sorozatok tekintetében, a legegyszerűbb módszert fejtegettem, miszerint meg kell egyezzen a felirat címében lévő évad és epizód szám, valamint a címnek is egyeznie kell. Itt felmerült az a probléma, hogy nem elég a címben való contains metódust használni, mivel lehet hogy létezik olyan sorozat amely 1 szó hosszúságú, viszont nekünk 3 szó hosszú a címünk, és csak valamilyen formában tartalmazza azt az 1 szót. Így ez nem ad egyértelmű választ, szükséges olyan szempontból vizsgálni, hogy a címben lévő szöveg a kötőjelig milyen szavakat tartalmaz. Ezt követően kell összehasonlítani a saját címünket felirat címével.

Miután megbizonyosodtunk arról, hogy egyezést találtunk mind cím és epizód szerint, akkor kezdődhet meg a felirat időzítésének és minőségének megfelelő kiválasztása. Ezekhez a saját médiánkból kiindulva már megvannak a megfelelő értékek, ezekhez keressük a megfelelőt, amennyiben létezik ilyen. A feliratok leírása fogja tartalmazni a szükséges információkat, így ezekben kell vizsgálódni. A legegyszerűbb módszert választva, itt már a contains metódus tökéletesen meg tudja állapítani, hogy adott esetben szerepelnek-e az általunk kért igények a feliratban.

ÉVADPAKK

Miután egy évad teljesen véget ér, a feliratokat évadpakkban becsomagolva lehet elérni, így ezt külön kell vizsgálni, mivel az oldalon történő elnevezések már nem egyeznek a pontos egyezésben lévő elnevezésekkel. Ekkor sa felirat címe már az évadot fogja tartalmazni, valamint a leírásából is kikerülnek azok az adatok, amik egyértelműen az epizódra vonatkoztak. Ekkor szükséges figyelembe venni azt is, hogy az évadok leírása is eltérőek a címben és a leírásban. Ez azt jelenti, hogy a cím ekkor magyarul ’8. évad’, a leírás pedig angolul ’Season 8’ adja meg ezt az adatot. A cím teljesen átalakult mert az évad kifejezés zárójelben van és a kötőjel is eltűnik, s ugyan ez a helyzet a leírásával is. Ezeket figyelembe véve, újabb regex mintákkal kialakítva fogom(melyek a SeriesHelper-ben elérhetők) a vizsgálatokat végezni.

A vizsgálat első körben úgy történik, hogy megpróbálom kinyerni a felirat címéből és leírásából az évadot a két különböző regex technikával. Amennyiben sikerül a címből kinyerni, azzal továbbhaladva történik a letöltés, a név, a minőség és releaser ellenőrzésével.

Másik esetben, ha a címben nem találtuk meg az évadot, akkor a leírásból próbáljuk folytatni a letöltést, ami ugyanahhoz az eredményhez vezet mint az első ág.

NINCS CÍM

Ebben az esetben csak egy kötőjel szerepel a címben, így egy keresési lehetőség azonnal kiesik. Ekkor arra tudok hagyatkozni, hogy a leírásban fogom eldönteni, hogy pontos egyezésről van szó, vagy évadpakkról beszélünk.

CHECKMATCHING ÉS LETÖLTÉS

A megfelelő felirat kiválasztása után két további fontos lépés következik. A CheckMatching metódus végzi el azt a feladatot, ami megmondja, hogy a letöltést el lehet-e kezdeni vagy sem. Ebben történik az oldalon talált névvel való egyezés ellenőrzése, továbbá a releaser, a minőség, epizód és évad megegyezése.

A felirat letöltése mind a felirat kiválasztás és egyezés megtörténte után indulhat meg. Ez a felirathoz rendelt URL hívását követően letölti, majd a megfelelően elhelyezi a feliratot a médiához. Elvégzi a megfelelő módosításokat a feliraton, beleértve az átnevezést, kicsomagolást, mappázást.

Mivel az évadpakkok megnehezítik a helyzetet, ezért őket a feliratok letöltésénél is külön kell kezelni. Ennek magyarázata, hogy a pontos egyezés esetében a feliratok egyetlen fájlt tartalmaznak, amelyek .srt kiterjesztéssel rendelkeznek. Ekkor nincs más dolgunk, mint a felirat média mellé való helyezése, és annak átnevezése a médiával egyezve. Az évadoknál az az eltérés, hogy a feliratok is együttesen érhetők el, ami azt jelenti, hogy becsomagolva érkeznek. A letöltés után .rar kiterjesztésű csomagot kapunk, melynek kicsomagolására, mappázására, majd a mappából csak az adott részhez illő feliratot kiszedve, a médiához való illesztése és átnevezése igényelt.

Mivel azt már a felirat letöltése előtt tudjuk, hogy egyetlen fájlt vagy csomagot fogunk kapni, ezért a Download metódus már aszerint kezdi meg a letöltést, miszerint ez a bool változó igaz vagy hamis értéket képvisel.

VISSZATÉRÉS AZ ELEJÉHEZ

A feliratok letöltése után, a média elindulását követően az alkalmazás figyelembe veszi, hogy a megfelelő felirat már korábban letöltésre került-e. Ez érvényes az évad feliratokra is, ekkor a már alkalmazás által letöltött évad feliratok közül, kiválasztja azt amelyik szükséges a még nem elkezdett részhez, ekkor megtörténik az alkalmazása a lejátszó újraindítását követően, amely automatikusan történik. Az csomagban érkező feliratok közül ilyenkor törlődik az adott felirat a duplikáció elkerülése végett, így segítve a felhasználó tájékozódását, amennyiben szeretne közbelépni.

RECOMMENDBOOK

RECOMMEND SERIES

APPLICATION MANAGER

SOCIAL RÉSZHEZ KIEGÉSZÍTÉS A TOVÁBBIAKRÓL (KÖZÖS ÉRDEKLŐDÉS ALAPJÁN VALÓ AJÁNLÁS, AJÁNLÁS MEGADOTT PROGRAMHASZNÁLATOK ALAPJÁN, SEGÍTSÉGKÉRÉS A BEJELÖLT PROGRAMHOZ, EGYÉNI STATISZTIKÁK KIMUTATÁSA TÉMÁK ALAPJÁN –HOBBI, MUNKA-, PROGRAMOK KATEGORIZÁLÁSA TÉMÁKBA –MUNKA, HOBBI, SZÓRAKOZÁS-, ÁLTALÁNOS STATISZTIKA, AZAZ MINDEN FELHASZNÁLÓ ÁLTAL TÖRTÉNT STATISZTIKA KIMUTATÁSA + EZ TÉMÁK ALAPJÁN IS)