Spotify kliens adatmániásoknak

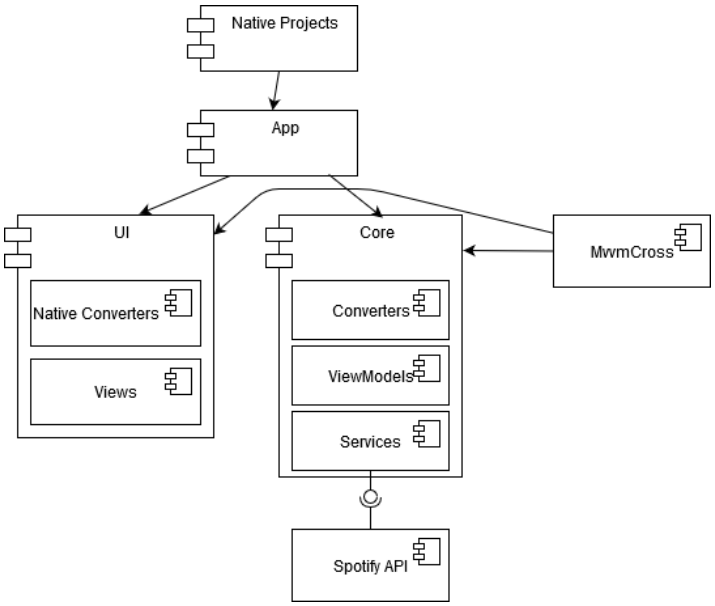
Önálló laboratórium – Készítette: Séllyei Bence (VF90WF)

Összefoglaló

Az alkalmazás a sokak által ismert és használt Spotify applikáció mobilverziójának rekreációja. Az alkalmazás egy új felhasználói réteget megcélozva az ún. adatmániásoknak készült. Az adatmániások egy olyan réteget képviselnek, akik szeretnének többet megtudni a zenén belül egy adott előadóról, albumról további metaadatok alapján, és ezek segítségével alakítanák ki zenei profiljukat, gyűjteményüket. Sok felhasználó van úgy, hogy szeretne szélesebb körben keresni (például megjelenési év, műfaj) a streaming szolgáltató által nyújtott zenékben, azonban ez az ő alkalmazásaikban nem elérhető. Erre a hivatalos Spotify alkalmazások nem nyújtanak lehetőséget, a felhasználói felületeiken nem jelentek meg ilyen adatok, az alkalmazásaik inkább az egyszerűségre, felhasználóbarát felületre és használatra törekednek. Azonban a Spotify publikus API-ján keresztül a keresett, kívánt adatok elérhetőek, így az adatmániások számára vizionált funkciók elérhetőek mindenki számára, már csak egy felhasználói felület hiányzott, ami ezeket megjeleníti. A feladat célja az volt, hogy az említett átlag felhasználó számára rejtett adatokat segítségül hívva egy új alkalmazás álljon elő, amely képes kielégíteni az adatmániás réteg igényeit.

Architektúra

Az alkalmazást Xamarin.Forms frontenddel valósítottam meg, backendként pedig a Spotify hivatalos, publikus API-ját használtam fel.

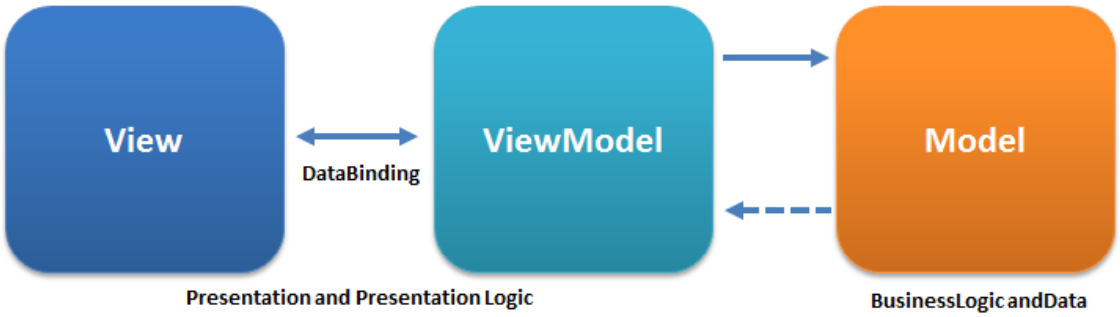


Xamarin.Forms

A Xamarin.Forms egy olyan UI technológia, amellyel könnyen fejleszthetünk Android, iOS és Windows alkalmazásokat közös kódbázissal. A felhasználói felületeket XAML nyelven, a mögöttük lévő logikát pedig C#-ban írhatjuk meg, futtatáskor pedig a XAML-ben példányosított közös UI elemek minden platformra az adott platformnak megfelelő UI elemként renderelődnek ki. Természetesen arra is van lehetőség, hogy saját egyedi UI elemeket készíthessünk a platformokra külön is, azonban erre ebben a projektben most nem került sor.

MvvmCross

Az alkalmazás elkészítése során az MVVM (Model-View-ViewModel) mintát követtem, ennek megvalósításához pedig az MvvmCross külső keretrendszert használtam. Az MVVM egy olyan architekturális minta, amely elválasztja egymástól a felhasználói felületet és az üzleti logikát. A nevéből jövően 3 szereplője van a mintának, ezek az adatmodell, nézetmodell és a nézet. A nézetmodell (ViewModel) alakítja át és készíti elő az adatmodellben található adatokat a nézet számára, hogy az megjeleníthesse őket.



1. MVVM szerkezete (forrás: Wikipédia)

Az MvvmCross az alapvető MVVM funkciók mellett (navigáció, adatkötés, függőséginjektálás, stb) tartalmaz más fejlesztésben hasznos eszközt is. Ilyenek például a View Presenter-ek, Value converter-ek, aszinkron műveleteket segítő saját osztályok, vagy saját nézetelemeik, osztályaik. Az MvvmCross nézetelemei nem saját megoldások, csupán a Xamarin.Forms-os elemek lettek kiegészítve pár plusz művelettel, tulajdonsággal.

Navigáció MvvmCross-al

Ebben a részben ismertetem, hogy az alkalmazás milyen navigációs megoldásokat használ az MvvmCross keretrendszerből. Először is a navigáció egy IMvxNavigationService interfészű objektumon keresztül történik, ennek kell meghívni a Navigate<> függvényét. Itt a Navigate függvény template paramétereket kap, amikből az első paraméter, hogy melyik ViewModel-re szeretnénk továbbnavigálni. Ha adatot is akarunk küldeni a következő ViewModelnek, akkor második template paraméterként meg kell adni a küldendő objektum típusát, a függvény paramétereként pedig magát a küldendő objektumot.

```
private async Task OpenArtistAsync(FullArtistDto artist)
{
    if (NavigationService != null)
    {
        await NavigationService.Navigate<ArtistViewModel, FullArtist>(artist.Artist);
    }
}
```

Fogadó oldalon a ViewModel ősébe be kell csomagolni a fogadni kívánt objektum típusát és felül kell definiálni az MvxViewModel Prepare(T parameter) függvényét. Ez a függvény fogja majd fogadni és eltárolni a ViewModelen belül a küldött paramétert. Ha a ViewModel számára szükségünk van még további műveletek elvégzésére a kapott objektumon, akkor még egy függvényt, az Initialize-t kell felüldefiniálni a ViewModelben.

```
public override void Prepare(FullArtist parameter)
{
    Artist = parameter;

}

1 reference
public override async Task Initialize()
{
    await base.Initialize();

    var topTracks = await SpotifyApiService.GetApi().GetArtistsTopTracksAsync(Artist.Id, "HU");
    for (int i = 0; i < 5; i++)
        TopTracks.Add(topTracks.Tracks[i]);
}
```

Tapasztalatok, megoldások

MvvmCross-al folytatott munka során észrevettem pár hibát, ami engem akadályozott a program megírása közben. Az első ilyen hiba az volt, hogy a View-k általában nem találták meg a hozzájuk tartozó ViewModelt. Ilyenkor a ContentPage BindigContext-jének be kellett

állítani magát a ViewModelt. Ez a lépés pedig azért okoz további problémákat, mert csak olyan osztályt lehet BindingContextnek megadni, aminek van paraméter nélküli konstruktora és ezzel a konstruktorral fogja lepéldányosítani az osztályt, amikor arra a nézetre navigálunk. Az ilyen paraméter nélküli konstruálásnál pedig nem lehet dependency injectiont használni, anélkül pedig navigálni sem lehet a nézetek között. Ezért a DI elkerülése érdekében az MvvmCross beépített IoC Provider-ét használva kértem el az összes ViewModelnek az IMvxNavigationService-t. A második akadályozó hiba pedig az volt, hogy az MvvmCross a tab-os megjelenítéshez nem hoz létre minden tabhoz saját navigation stacket, hanem egy ilyen stack van az összes tabhoz. Ez azt eredményezi, hogy ha a 2. vagy 3. tabról új nézetre szeretnénk navigálni, akkor ezek a nézetek (abban az esetben, ha szeretnénk, hogy a navigation tabbar mindig a képernyőn legyen) az 1. tab felett nyílnak meg, nem pedig afelett ahol megnyitottuk őket. Ha a navigation tabbar-t elhagyjuk, akkor viszont a megfelelő tab felett nyílnak meg a nézetek, ekkor azonban nem tudunk tabot váltani, csak akkor ha a navigation stacken egészen visszamegyünk a tabokra. Ez egy ismert hiba az MvvmCross-nál, remélem, a következő verzióban már kijavítják.

Spotify API

A Spotify API-ja egy hivatalos, publikus API, melyet maga a Spotify hivatalos applikációk is használnak kommunikációra. Ebben az alkalmazásban egy .NET-ben megírt API Wrappert[1] használtam. Ez a wrapper megvalósítja majdnem az összes elérhető API végpontot szinkron és aszinkron módon is, tartalmaz saját modellosztályokat a hívások válaszainak feldolgozásához, valamint hibakezelést is. Egy másik package-el telepíthetőek még ugyanebből a wrapperből a Spotify által támogatott authentikációs módok implementációi is. A wrapper az API-val együtt folyamatosan fejlődik, csak pár frissítéssel van lemaradva, de így is teljesen jól használható. A wrapperben található függvények nevei nem mindig egyeznek meg teljesen az API végpontjainak neveivel, de az ezen való kiigazodáshoz segít a wrapperhez írt dokumentáció. Sajnos a dokumentáció nem naprakész, elég hiányos, ezért érdemes fejlesztés közben a forráskódot is nézni, mert lehet már bekerültek olyan funkciók, amik a dokumentációban nem jelentek még meg.

[1] https://github.com/JohnnyCrazy/SpotifyAPI-NET

Főbb komponensek

Az alkalmazás egy tabokból álló alkalmazás, ami azt jelenti, hogy jelen esetben három különböző nézet (tab) mindig meg van nyitva az alkalmazásban és ezek között a képernyő alján lévő navigációs sávval lehet navigálni. A fő három tab a következő nézetekből áll:

- Browse: a felhasználó zenehallgatási tevékenysége alapján összeálló nézet
- Search: Spotify könyvtárában való keresés és a keresési találatok megjelenítésére szolgáló nézet
- Library: felhasználó saját könyvtárát megjelenítő nézet

A Library nézeten belül is van még további 3 tab, ezek az alkalmazásban megjelenő fő entitások (Playlist, Album, Artist) egy felhasználó által lementett listáját jelenítik meg.

Az MVVM minta alapján minden nézethez tartozik egy ViewModel, amik a projektben az MvvmCross-os MvxViewModel osztályból származnak le, amely megvalósítja az INotifyPropertyChanged interfészt. A tabos megjelenítéshez a XAML fájlokban el kell helyezni a megfelelő nézetelemeket (MvxTabbedPage) és az ilyen fájlok code-behindjában (xaml.cs fájlok) az osztályoknak meg kell adni egy prezentációs attribútumot (pl. MvxTabbedPagePresentation). A xaml.cs osztályokban is érdemes a sima Xamarin.Forms-os osztály helyett az MvvmCross-os osztályból leszármazni.

BrowseTabPage és BrowseTabViewModel

Ez a nézet nagyon hasonlít a hivatalos Spotify alkalmazásban megtalálható Browse tabhoz mind tartalmilag, mind kinézetileg. A nézet horizontálisan görgethető listákat tartalmaz, melyek megjelenítik a bejelentkezett felhasználó legutóbb hallgatott albumait, kedvenc előadóit és albumait, új albummegjelenítéseket és a Spotify által ajánlott lejátszási listákat. A horizontális listák megvalósításához egy külső komponenst, a Sharpnado.Forms.HorizontalListView[2]-t használtam fel, amit Githubon találtam. Ezt a sima ListView-hoz hasonlóan lehet használni, annyi különbséggel, hogy meg lehet neki adni egy property-jén keresztül (ColumnCount), hogy a képernyőn egyszerre hány elem látszódjon, ennyi elemet is fog elgörgetni, ha elhúzza rajta az ujját a felhasználó. A lista kinézetének egy

[2] https://github.com/roubachof/Sharpnado.Presentation.Forms

Frame elemet adtam meg és arra tettem rá egy TapGestureRecognizert, így mindegy hogy az elem képét vagy szövegét érinti meg a felhasználó, meg fogja nyitni az elemet egy újabb nézeten. A nézet ViewModelje tartalmazza a horizontális listákhoz tartozó listákat, valamint kommandokat az adatbetöltéshez és navigációhoz.

SearchTabPage és SearchTabViewModel

Ez a nézet a Spotify zenekönyvtárában való keresést valósítja meg. A nézet tetején van egy searchBar amiben név szerint lehet keresni az adatbázisban. Fejlesztői oldalról a nézet úgy néz ki, hogy van a searchBar alatt négy listanézet és kereséskor a találatok alapján ezek a listák megjelenítődnek aszerint, hogy a keresés hozott-e eredményt az adott listába. A négy lista a különböző entitások szerint vannak szétbontva, az elsőben csak a zeneszám találatok jelennek meg, a másodikban az előadók, harmadikban az albumok, negyedikben pedig a lejátszási listák. Amikor a keresés elindul, akkor az összes lista eltűnik, majd a találatok alapján jeleníti meg őket, valamint kiszámolja, hogy a nézeten az adott lista mennyi helyet foglaljon. Erre azért van szükség, mert ha nem adnánk meg, akkor előfordulna, hogy minden listához külön scrollbar jönne létre, ami a görgetést nagyban megnehezítené, vagy túl kevés elem esetén nem igazítaná a magasságot az elemek számához magától, így üres helyek maradnának a lista aljában.

LibraryTabPage és LibraryTabViewModel

Ez a nézet nem tartalmaz érdemi logikát, csak egy gyökérként szolgál a már korábban említett Libraryben megtalálható további három tabnak. Itt vannak felsorolva, hogy milyen nézetek jelenjenek meg az egyes tabokon, és további tabokat is itt lehet hozzáadni.

AlbumsTabPage és AlbumsTabViewModel

A felhasználó által lementett albumok egy listáját jeleníti meg ez a nézet. Ahhoz hogy kevés navigációval is sok adatot lásson a felhasználó a saját könyvtárában található elemekről accordionokat vezettem be erre a nézetre és ezekből állítottam össze egy listát. Az accordionok megvalósításához egy külső osztálykönyvtárat, a Xamarin.CustomControls.AccordionView[3]-t használtam fel. Ezzel az osztálykönyvtárral egy

[3] https://github.com/DottorPagliaccius/Xamarin-Custom-Controls

darab accordiont vagy accordion listát is összeállíthatunk. Az accordion legördülő részébe pedig bármilyen egyedi nézetet megadhatunk. Az egyetlen hiba amit ezzel kapcsolatban találtam az az, hogy az accordion fejlécére nem lehet URL-ből betöltött képet feltenni, csak olyat ami fájlból van betöltve. A ViewModel adatbetöltésen és navigáción kívül nem tartalmaz más logikát. Adatbetöltésnél az API végpont egy saved album objektumot ad vissza, amely tartalmazza a teljes album objektumot, ebben van benne az adatok nagy része, amit megjelenít a nézet. Ezt talán csak ennél a hívásnál vettem észre, de van olyan adattag, amit az API nem tölt ki, pedig nekem szükségem lett volna rá. Ez az adat szerencsére benne volt az albumhoz tartozó előadó objektumában, ezért onnan el tudtam kérni.

ArtistsTabPage, ArtistsTabViewModel, PlaylistsTabPage, PlaylistsTabViewModel

Nagyon hasonlóak az AlbumsTabPage-hez, accordionokat használ ez is az adatok megjelenítésére, csak mást jelenít meg és más API végpontokkal kommunikál.

AlbumPage, ArtistPage, PlaylistPage

Albumok, előadók és lejátszási listák bővebb megjelenítésére szolgáló nézetek. Többnyire az accordionos nézeteken is látható adatokat jelenítik meg kiegészülve az albumok vagy lejátszási listák dalaival. Itt adatbetöltésre általában nincs szükség, az adatokat egy másik nézettől kapja navigációs megoldások segítségével. Erről bővebben a Navigáció MvvmCross-al fejezetben lehet olvasni.

SpotifyApiService

Ez az osztály egy singleton mintát megvalósító osztály, amely tartalmazza a külső komponensből jövő SpotifyWebApi osztályt. Erre az osztályra két dolog miatt volt szükség. Az első, hogy már korábban említett okok miatt elkerüljem a dependency injectiont. A másik ok, hogy időhiány miatt nem lett authentikáció megvalósítva az alkalmazásban, ezért az applikáció csak akkor használható, ha kérünk egy authentikációs token a Spotify for Developers[4] oldalról, megadjuk, hogy melyik scopeokat szeretnénk használni a tokennel és beillesztjük ebbe az osztályba. Az alkalmazás funkciói számára szükséges scope-ok is ebben az osztályban vannak felsorolva.

[4] Például: https://developer.spotify.com/console/get-current-user-saved-albums/?limit=1

ValueConverter-ek

A ViewModelbeli adatok adatkötéssel vannak hozzárendelve a nézetek elemeihez, ez azért jó, mert ha a megváltozik egy adat, akkor automatikusan frissül is a hozzá tartozó nézet. Viszont így az adatszerkezetben lévő adat nem feltétlenül olyan formában jelenik meg, mint ahogy azt mi elképzeltük, de a ViewModel-ben lévő objektumba nem kellene beleírni, hiszen annak nem kellene a nézettől függenie. A nézetek számára az adatokat ValueConverter-ekkel lehet átalakítani, erre az MvvmCross nyújt is egyszerű megoldásokat. Én legtöbbször arra használtam az ilyen konvertereket, hogy szövegeket fűzzek az értékekhez.

Új value convertert a következőképpen lehet létrehozni: kell egy új osztály, ami leszármazik az MvxValueConverter<TFrom, TTo> osztályból. Ide a TFrom helyére azt kell beírni, hogy milyen típusú adatból konvertálunk, a TTo helyére pedig azt hogy milyen típusba konvertálunk. Leszármazás után a Convert, ConvertBack függvényekben elvégezzük a konverziót. A UI-os projektben létre kell hozni az összes ilyen konverter osztályhoz egy natív megfelelőt, ezt úgy kell, hogy az előbb létrehozott konvertert wrappeljük egy MvxNativeValueConverter osztályba, és ebből származunk le a UI projektben. Az itteni natív konverter osztály lehet üres osztály. Végül a natív konvertert az App.xaml-ben fel kell venni a ResourceDictionary-be és hozzá kell rendelni egy kulcsot, amivel majd hivatkozunk a konverterre a különböző nézetekben.