



EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

INFORMÁCIÓS RENDSZEREK

TANSZÉK

Youtube Playlist State Manager App

(Youtube Playlists+)

Témavezető:

Dr. Vörös Péter
egyetemi adjunktus

Szerző:

Horváth Ádám
programtervező informatikus BSc

Budapest, 2024

Tartalomjegyzék

1. Bevezetés	3
2. Felhasználói dokumentáció	5
2.1. Telepítés	5
2.1.1. Android	6
2.1.2. Windows	7
2.2. Főoldal	8
2.3. Keresőoldal	9
2.3.1. Speciális keresések	9
2.3.2. Link megosztása	10
2.4. Lejátszási lista oldal	11
2.4.1. Videók	11
2.4.2. Változások	12
2.4.3. Előzmények	13
2.5. Személyre szabhatóság	14
2.6. Kontextus menük	16
3. Fejlesztői dokumentáció	18
3.1. Fordítás, futtatás	20
3.2. Architektúra	21
3.3. Model	22
3.3.1. Anchor (Rögzített pozíció)	22
3.3.2. Media	23
3.3.3. Video	24
3.3.4. VideoChange (VideóVáltozás)	26
3.3.5. VideoHistory (VideóElőzmény)	27
3.3.6. Playlist (Lejátszási Lista)	29
3.4. Persistence (Perzisztencia) könyvtár	34

3.4.1. Persistence (Perzisztencia) osztály	35
3.4.2. Codec	36
3.4.3. Preferences (Preferenciák)	36
3.4.4. ThemeCreator	37
3.4.5. PlaylistStorage	38
3.4.6. AnchorStorage	39
3.5. Provider (ViewModel / NézetModell)	40
3.5.1. AnchorStorageProvider	41
3.5.2. PlaylistStorageProvider	41
3.5.3. PreferencesProvider	42
3.5.4. FetchingProvider	43
3.6. View (Nézet)	45
3.6.1. HomePage (Főoldal)	46
3.6.2. SearchPage (Keresőoldal)	46
3.6.3. PlaylistPage (Lejátszási lista oldal)	47
3.6.4. AboutPage (Információs oldal)	48
3.6.5. Other widgets (Egyéb komponensek)	48
3.7. Services (Szolgáltatások)	50
3.7.1. BackgroundService	51
3.7.2. NotificationService	52
3.7.3. PopupService	52
3.7.4. SharingService	53
3.7.5. YoutubeService	53
3.7.6. NavigatorService	53
3.8. Tesztelés	55
3.8.1. Szolgáltatások	55
3.8.2. Nézet	60
3.9. Összegzés	63
3.9.1. További lehetőségek	63
Irodalomjegyzék	64

1. fejezet

Bevezetés

A szakdolgozat egy Multi-Platform (Android és Windows) alkalmazás fejlesztéséről szól, mely Youtube lejátszási listák állapotát tudja kezelní.

Hogy mit is jelent ez? Tegyük fel, hogy:

1. van a felhasználónak egy saját lejátszási listája
2. ez a lista viszonylag sok videót tartalmaz (több mint 50)
3. az egyik videót letörölték Youtube-ról

Ekkor a felhasználó nem fog értesülni arról, hogy a lejátszási listája már a törölt videót nem tartalmazza. Csak akkor fogja észrevenni, amikor pontosan a törölt videót szeretné visszakeresni a lejátszási listájában, de nem találja. Eltölthet vele akár fél órát is, mire valóban megbizonyodik róla, hogy a videó törlődött, és esetleg keres egy másik videót amire lecserélheti.

Ha pedig több videót is töröltek, az már akár órákat is elvehet a felhasználó idejéből, hogyha elég nagy a lista és mindenképpen meg akar bizonyosodni róla, hogy minden videója helyettesítve egy lett másikra.

Ennek a problémának az automatizálását és könnyítését szolgálja az alkalmazás. A lejátszási listák állapotát az alkalmazás lementi lokálisan, mely főként a videók listáját jelenti. Lementés után ha a jövőben úgy dönt a felhasználó, hogy kíváncsi, hogy egy listájának állapota megváltozott-e, akkor ezt megteheti az alkalmazás segítségével. Ehhez az alkalmazás összehasonlítja a lejátszási lista lementett állapotát, és a Youtuben szereplő jelenlegi állapotát. Hogyha bármilyen változás történt volna a két állapot között, akkor azokat megjeleníti és a megfelelő módon kezelhetővé

teszi. Új videókat hozzáadhat a lementett állapothoz, törölteket pedig kivehet (és cserélhet, ekkor 1 törölt és 1 új videó lenne az eltérés).

Előfordulhat, hogy a felhasználó nem talál egyből a törölt videónak megfelelő videót, amivel lecserélhetné. Ehhez a legegyszerűbben egy jegyzettömböt használhat, amibe beleírja a videókat, amiket nem talált meg, és máskor újra rákeres YouTube-on. Ez a funkció az alkalmazásba is be van építve, minden egyes lejátszási listához tartozik egy "Planned" (Tervezett) lista, melybe beleírhatóak később a lejátszási listához hozzáadandó videók.

Egy másik felmerülő probléma lehet lejátszási listák esetén a videók pozíciója. Hogyha a felhasználó egy bizonyos videót például a lejátszási lista közepén szeretne tartani, akkor minden alkalommal, amikor a lejátszási lista változik, akkor ellenőriznie kell a videó pozíóját. Erre szintén ad megoldást az alkalmazás, egy lejátszási lista bármely videójára rakhattunk ún. "Anchor"-t (Rögzítést). Ez a Rögzítés pedig ellenőrzi, hogy az adott videó a megfelelő helyen szerepel-e.

Emellett hogyha Android platformra töltjük le az alkalmazást, akkor van lehetőség a program háttérben való futtatására. Ekkor az alkalmazás bizonyos időközönként automatikusan lefogja ellenőrizni a háttérben a lementett lejátszási listáink állapotát, majd ha bármi változást észlel, akkor arról értesítést küld a felhasználónak. Így még emlékeznie sem kell a felhasználónak arra, hogy néha megnézze az alkalmazást, ugyanis ameddig nem jön értesítés, addig megbizonyosodhat arról, hogy a listáiban nem történt változás.

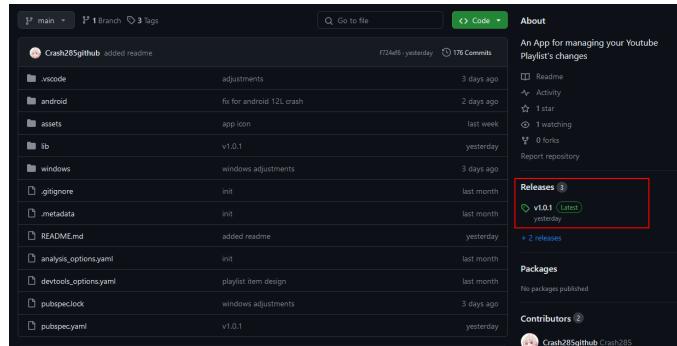
2. fejezet

Felhasználói dokumentáció

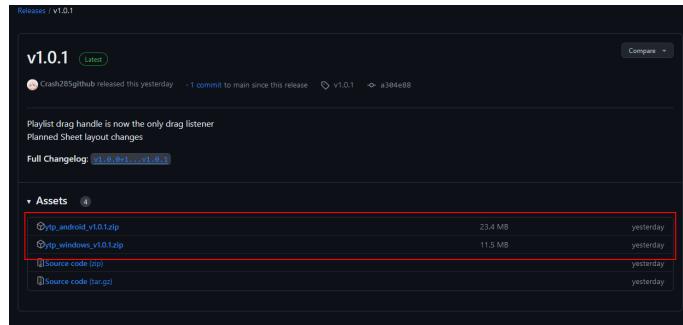
Ebben a fejezetben lesz bemutatva az alkalmazás használata, telepítése, funkciói ábrákkal kiegészítve.

2.1. Telepítés

Az alkalmazás [githubról](#) letölthető. A "Releases" fülre (ld. 2.1 ábra) kattintva juthatunk el a legfrissebb verzióhoz, ahonnan letölthetjük a platformunknak megfelelő (ld. 2.2 ábra) alkalmazást.



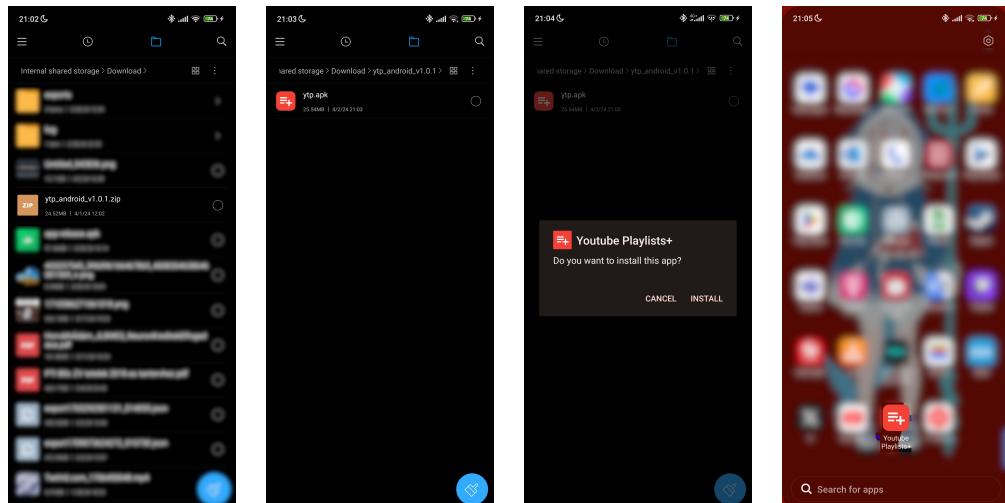
2.1. ábra. A github "Releases" fül



2.2. ábra. A letöltendő fájlok platformtól függően.

2.1.1. Android

Githubról való letötés után a zip fájl meg fog jelenni a "Letöltések" mappánkban (vagy ha más helyre lett letöltve akkor ott fog megjelenni.) Ezután ki kell csomagolnunk a zip fájlt. Ha ekkor rányomunk az apk fájlra, lehetséges hogy az Android nem fogja egyből engedni az ismeretlen forrásból származó alkalmazások telepítését. Ezt engedélyezni kell a beállításokban, mely minden telefongyártó esetében eltérő módon történik. Ha ezt a lépést viszont megtettük, akkor az alkalmazás már telepíthető lesz. Telepítés után pedig az alkalmazás meg fog jelenni az összes többi alkalmazásaink között.



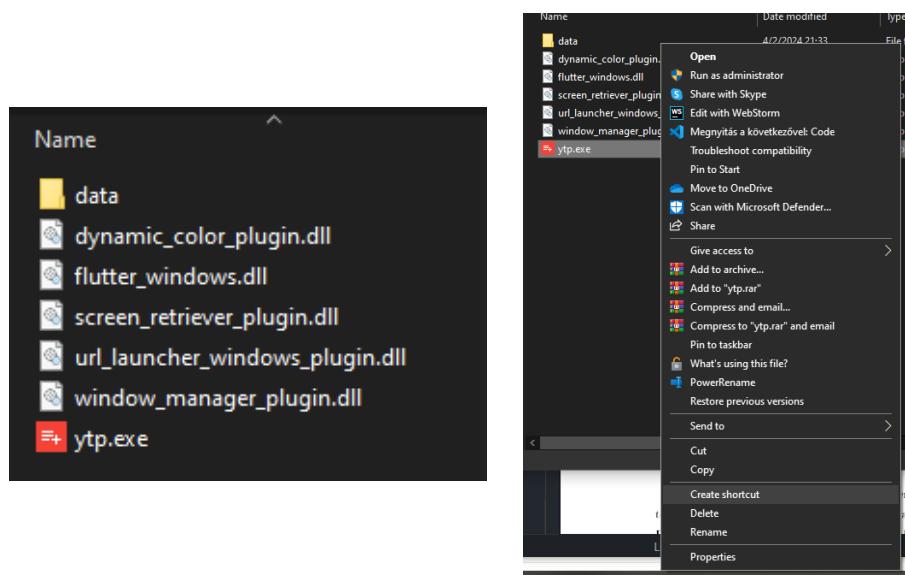
(a) A letöltött zip fájl (b) A zip fájl tartalma egyetlen .apk fájl

(c) A telepítő menü (d) Az alkalmazás

2.3. ábra. A telepítés lépései

2.1.2. Windows

Windows esetében a letöltött zip fájl több fájlt is tartalmaz (ld. 2.4 ábra). Az alkalmazás a 'ytp.exe' fájl indításával futtatható. A többi fájlra is szüksége van viszont, melyeknek ugyanabban a mappában kell maradnia, mint a futtatható fájl. Hasznos ezért, hogy ha a kicsomagolt mappát elrakjuk egy biztonságos helyre, és a "ytp.exe" fájlból egy parancsikon csinálunk, amit utána bárhova rakhatunk. Fontos viszont, hogy az eredeti .exe fájl a helyén maradjon, ugyanis annak a többi fájlra is szüksége van a működéshez.



(a) A zip fájl tartalma

(b) Parancsikon létrehozása

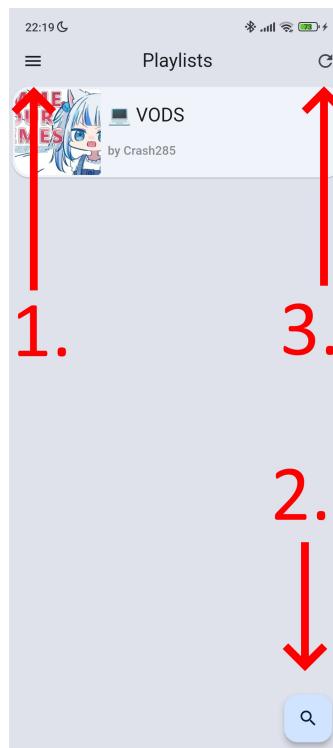
2.4. ábra. A telepítés lépései

2.2. Főoldal

Az alkalmazás a **Főoldalon** nyílik meg, ahol elsősorban a lementett lejátszási listáinkat láthatjuk. Első indításkor viszont még nem lesznek lementett lejátszási listáink. A jobb alsó keresőikonnal rendelkező gombra (ld. 2.5 ábra) kattintva a **Keresőoldalra** (2.3) juthatunk. Itt tudunk lejátszási listákat keresni Youtube-ról, majd azokat hozzáadni az alkalmazáshoz.

Emellett a bal felső menü gombra (ld. 2.5 ábra) nyomva a **Személyre szabhatósági** (2.5) menüt nyithatjük meg.

Ha már vannak lejátszási listáink, akkor a jobb felső sarokban megjelenik egy frissítés gomb (ld. 2.5 ábra), ezzel az összes lementett lejátszási listánkat frissíthetjük, ellenőrizhetjük.

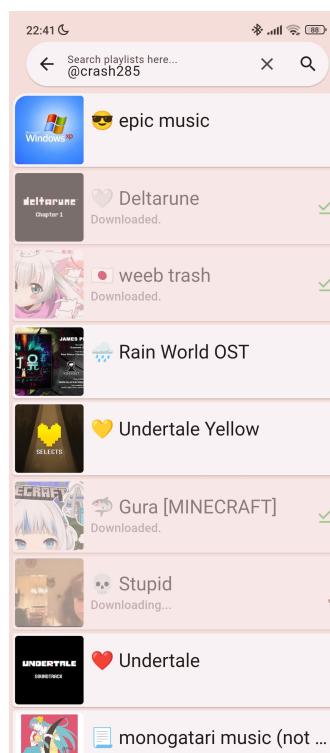


2.5. ábra. A főoldal, egyetlen lementett lejátszási listával; 1. a menü gomb, 2. a kereső gomb, 3. a frissítés gomb

2.3. Keresőoldal

Hogyha új lejátszái listát szeretnénk hozzáadni az alkalmazáshoz, akkor ezt megtehetjük a beépített keresőfunkció segítségével. A **Főoldalon** (2.2) a jobb-alsó sarakban található keresőikonnal rendelkező gombra (ld. 2.5 ábra) nyomva juthatunk el a **Keresőoldalra**, ahol végrehajthatjuk a keresést és a lementést. Az oldal tetjén található a keresőmező, ahova beírhatjuk a keresni kívánt listákhoz tartozó kulcsszavakat.

A keresést a kereső ikonnal, vagy Enter lenyomásával indíthatjuk el. A keresés után meg fognak jelenni különböző (még nem lementett) lejátszási listák, melyekre nyomva hozzáadhatjuk őket az alkalmazáshoz.



2.6. ábra. A keresőoldal, 3 letöltött (Downloaded) és 1 jelenleg letöltés alatt lévő (Downloading...) lejátszási listával.

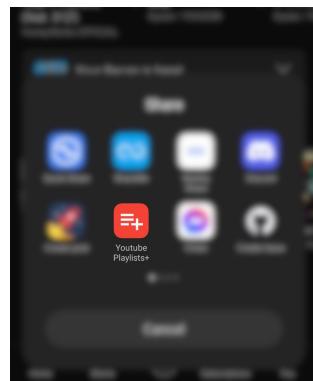
2.3.1. Speciális keresések

Alap esetben elsősorban a lejátszási listák címéhez próbálja asszociálni a Youtube a beírt kulcsszavakat. Lehet azonban, hogy egy adott csatorna lejátszási listáit keresük (mint például a sajátunkét). Hogyha Youtube csatornákkal szeretnénk asszociálni a kulcsszavakat, akkor segíthet, ha a '@' karaktert elé írjuk.

Emellett, hogyha nem listázott lejátszási listát akarunk keresni, akkor ezt az alap kereső nem fogja felajánlani. Ebben az esetben viszont a keresőbe bemásolhatjuk a lejátszási lista url-linkjét is, ugyanis így már nem keresni fog az alkalmazás, hanem az url alapján közvetlenül fogja elérni a lejátszási listát.

2.3.2. Link megosztása

Android platformon erre van egy kényelmesebb megoldás, hogyha a megosztás funkciót használjuk a lejátszási lista linkjére, ugyanis az operációs rendszer fel fogja ajánlani az alkalmazást. Miután pedig megosztjuk az alkalmazással a lejátszási listát, az automatikusan lefogja tölteni a megosztott lejátszási listát.



2.7. ábra. Lejátszási lista lementése megosztással

2.4. Lejátszási lista oldal

Ezen az oldalon egy adott lejátszási listát tudunk kezeln. Ez egy 3 **fülből** álló oldal; Változások (Changes) (2.4.2), Videók (Videos) (2.4.1) és Előzmények (History) (2.4.3). Ugyanitt lehetőségünk van a jobb felső gombok segítségével a listát törölni, vagy a listát frissíteni.

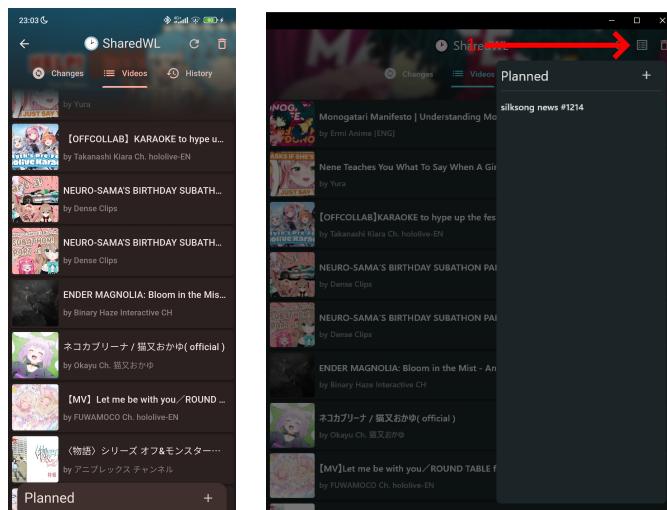
A következő részekben az egyes fülekre térünk ki.

2.4.1. Videók

Ha változás nélkül lépünk a lejátszási lista oldalra, akkor ez a fül fog fogadni. Itt a lejátszási lista összes videója megtalálható abból az állapotból, amikor utoljára frissítve lett. A videók azonos sorrendben vannak, mint Youtube-on.

Androidon ugyanezen a fülön található a 'Tervezett' (Planned) alulról felhúzható panel, ahova elsősorban olyan videókról tárolhatunk adatokat, melyeket 'tervezünk' hozzáadni a lejátszási listához. Erre elsősorban akkor van szükség, hogyha egy törölt videónak nem találunk egyből helyettesítést, így elmenthetjük későbbre, hogy majd újra utánanézzünk.

Windows esetében a panelt a jobb felső gombok közül a középsővel fogjuk tudni elérni, a törlés és a frissítés gombok között.



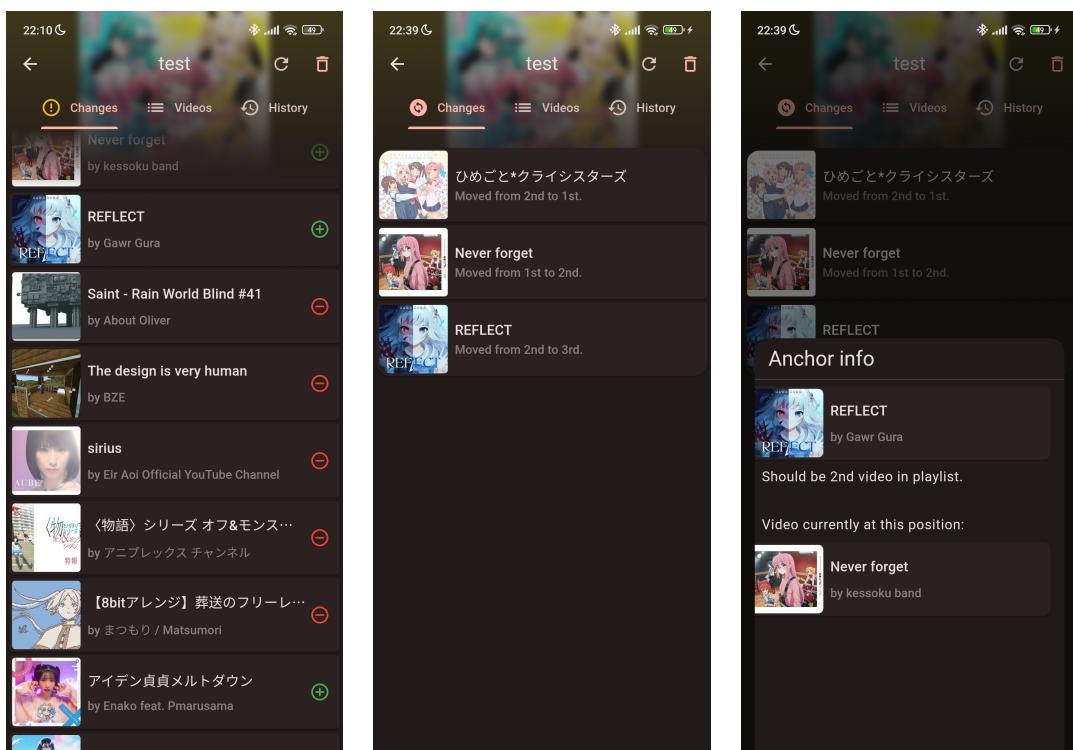
A 'Videók' fül

A 'Tervezett' panel Windowson,
1. jelöli a gomb helyét, amivel
nyitható.

2.8. ábra. A lejátszási lista oldal indítófájle, a 'Videók'.

2.4.2. Változások

Ezen a fülön találhatóak a lejátszási listával kapcsolatos legújabb változások. Ellenőrzés után, hogyha az alkalmazás törölt vagy új videókat észlel, azokat itt a megfelelő ikonnal ellátva megjeleníti. Ezek után egy nyomással megerősíthetjük ezeket a változásokat; hozzáadott videó esetén a videó bekerül a lokális lejátszási lista videói közé is, törölt videó esetén pedig az ellenkezője; kitöröljük a videót a likális lejátszási listából is. Abban az esetben, hogyha nem történt változás a lejátszási lista tartalmában, akkor ellenőrzéskor az alkalmazás a Rögzített ('Anchored') videókat fogja ellenőrizni, hogy a helyükön vannak-e. Hogyha nem, akkor erről is ezen a fülön fogunk információt kapni; tudni fogjuk hogy a rosszul elhelyezett videónak hol kéne lennie. Hogyha rányomunk, akkor pedig egy részletesebb panel fog felugrani; itt megtudjuk, hogy melyik videóval kellene megcsérálni.



Változások, piros ikonnal jelezve a törölt videókat, zölddel pedig a hozzáadottakat.

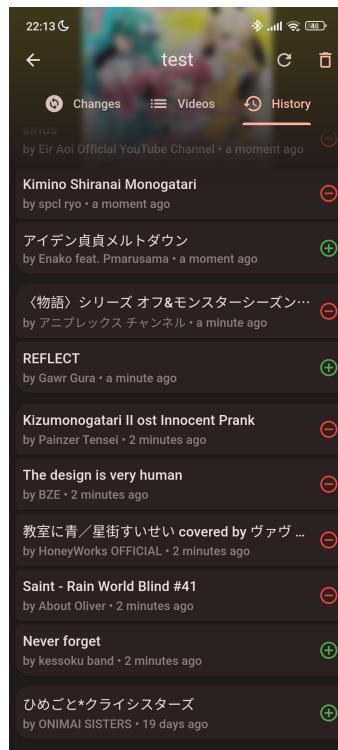
(a) Ha nincsenek változások, akkor a rosszul rögzített videók jelennek meg.

(b) A rosszul rögzített videókra nyomva részletesebb infót tudhatunk meg.

2.9. ábra. Változások fül

2.4.3. Előzmények

Erre a fülre kerülnek a lejátszási lista tartalmával történt változások. Hogyha egy videót változásként érzékel az alkalmazás, azt ellátja egy dátummal, és eltárolja az előzményekbe. Így bármikor visszanézhető, mikor milyen videókkal történtek változások egy adott lejátszási listában. Fontos azonban megjegyezni, hogy míg ellenőrzéskor a változások azonnal megjelennek az előzményekben is, azokat itt nem lehet megerősíteni, illetve a nem megerősített változások az előzményekben sem lesznek véglegesítve.



2.10. ábra. Az Előzmények fül, dátum alapján rendezve, illetve azonos időpontok alapján csoportosítva.

2.5. Személyre szabhatóság

Most nézzük az alkalmazás személyre szabhatóságát. Ehhez a főoldalon lévő felületi menüsávban található (ld. 2.5 ábra) gombra kell nyomni, vagy mobilon egy a képernyő széléről történő balra húzás is ugyanezt teszi. Ekkor a bal oldalról elő fog jönni egy menü (Drawer), ahol a felhasználó személyre szabhatja az alkalmazás kinézetét/viselkedését.

Ezek a személyre szabhatóságok a következőkben merülnek ki:

- **Dark mode** (sötét mód): beállítható vele az alkalmazás témája; világos, vagy sötét mód. Hosszú lenyomásra pedig fekete mód.
- **Color scheme** (színséma): az alkalmazás színe
 - dynamic: az alkalmazás az operációs rendszer színsémájához próbál igazodni: Windowson a rendszer színséma alapján, Androidon a **Material You** [1] alapján.
 - mono: monokromatikus, szürke színséma
 - red,orange...: egyéb színsémák
- **Confirm deletions** (törlések megerősítése): a felhasználó beállíthatja, hogy minden egyes törlésnél az alkalmazás megerősítést kérjen, vagy ne.
- **Hide '- Topic'** ('- Topic' szöveg elrejtése): a Youtube csatornák neveinek végéről elrejthető a '- Topic' kifejezés.
- **Split view** (osztott nézet): elég nagy képernyőméret esetén beállítható, hogy az alkalmazás egyszerre két oldalt legyen képes megjeleníteni.
 - Disabled (kikapcsolva)
 - Even (egyenlő): az osztott nézet eloszlásának aránya a két képernyő között 1:1
 - Uneven (egyenletlen): az osztott nézet eloszlásának aránya 2:3; 2 a bal oldal, 3 a jobb oldal
- **Run in background** (futtatás háttérben): csak mobilon elérhető funkció; az alkalmazás bezárás után is fut a háttérben, és magától ellenőrzi a lementett listák állapotát.

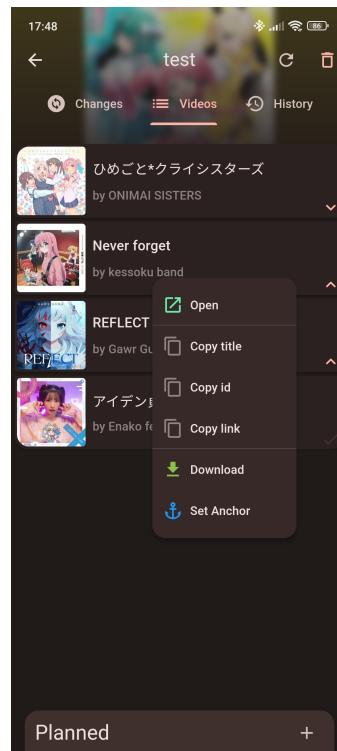
- **Reorder playlists** (listák átrendezése): segítségével a lementett lejátszási listák sorrendjét tudjuk megváltoztatni tetszés szerint.

Ugyanitt érhető el az exportálásra / importálásra szolgáló gombok, melyekkel az alkalmazás összes felhasználói adatát lementhetjük egy fájlba, illetve betölthetünk hasonló fájlokat.

Emellett a menü legalján található gombbal az információs oldalra juthatunk (3.6.4), mely egy alkalmazáson belüli kisebb használati útmutatóként szolgál, illetve a különböző licenszkről olvashatunk.

2.6. Kontextus menük

Az alkalmazásban minden megjelenített Média elemhez (Lejátszási lista, Videó, ...) (3.3.2) tartozik egy kontextusmenü. Ez Windows esetében jobb gombbal, Android esetében pedig hosszan nyomva érhető el.



2.11. ábra. Egy videó kontextusmenüje

Minden Média listaelemhez elérhetők az alábbi kontextusmenü opciók:

- **Megnyitás** (Open): Az adott médiát megnyitjuk Youtube-on.
- **Copy title** (Cím másolása): Az adott média címének vágólapra másolása.
- **Copy id** (Azonosító másolása): Az adott média egyedi Youtube azonosítójának vágólapra másolása.
- **Copy link** (Link másolása): Az adott média Youtube url linkjének vágólapra másolása.

Emellett a legtöbb Médiához tartoznak egyedi kontextus opciók is, melyek a következő bekezdésekben vannak felsorolva:

Lejátszási lista

- **Törlés** (Delete): Az adott lejátszási lista törlése az alkalmazásból.

Videó

- **Letöltés** (Download): Az adott videó letöltése.
- **Rögzítés** (Set Anchor): Az adott videó pozicionális rögzítésének beállítása.

Előzmény videó

- **Törlés** (Delete): Az adott előzmény törlése a lejátszási listából.

Videó változás

- **Hozzáadás** (Add): Az adott videó hozzáadása a lejátszási listához. Így erősítjük meg a változást. Csak hozzáadott videó esetén érhető el.
- **Törlés** (Remove): Az adott videó törlése a lejátszási listából. Így erősítjük meg a változást. Csak törölt videó esetén érhető el.
- **Hozzáadás a tervezett listához** (Add to Planned): Az adott videót hozzáadjuk a lejátszási lista Tervezett (Planned) (3.3.6) listájához. Csak törölt videó esetén érhető el.

Rögzített videó

- **Rögzítés** (Set Anchor): a rögzített videó rögzítésének megváltoztatása.

3. fejezet

Fejlesztői dokumentáció

Ebben a fejezetben lesz az alkalmazás működése kódszinten elmagyarázva, diagramokkal kiegészítve. Az alkalmazás a **Flutter** keretrendszerben íródott, mely a **dart** nyelvet használja. Így a dokumentáció ennek a nyelvnek a sajátosságait fogja használni, melyekből a fontosabbak itt vannak felsorolva:

- alap esetben minden adattag **publikus**, hogyha **privát** adattagot szeretnénk, akkor annak a neve `_` karakterrel kell kezdődjön.

```
1  class Explanation{  
2      int count = 3; // public  
3      int _length = 6; // private  
4  }
```

- lehet definiálni egyedi **getter** függvényeket, melyek valójában csak paraméter nélküli függvények. Ezeknek a szintaxisa:

```
1  class Explanation{  
2      [Object?] get customGetter{  
3          ... logic  
4  
5          return [value];  
6      }  
7  }
```

Meghívni pedig a következőképpen tudjuk:

```
1  object.customGetter;
```

- lehet definiálni egyedi **setter** függvényeket is, melyeknek szintaxisa:

```
1  class Explanation{  
2      set customSetter(Object? param){  
3          ... logic  
4      }  
5  }
```

Meghívásuk:

```
1  object.customSetter = value;
```

- Létezik az ún. nyílfüggvény szintaxis, mely egysoros függvények gyorsírását segíti. A következő két függvény ekvivalens:

```
1  int get length => _list.length;  
2  
3  int get length{  
4      return _list.length;  
5  }
```

Gettereknél gyakori a használata.

- Nagy jelentősége van még a **lambda** függvényeknek is. Ezek szintaxisa a következő:

```
1  (Object? param1, ...) {  
2      // logic  
3  }
```

A dart nyelvet és a Flutter keretrendszeret **deklaratívnak** nevezik. Ez azt jelenti, hogy a hangsúly nem azon van, hogy *hogyan* valósítunk meg valamit, mint **imperatív** esetben, hanem azon, hogy *mit* akarunk megvalósítani. Egy egyszerű példával szemléltetve, amikor meg akarunk találni egy elemet egy listában, akkor nem egy for ciklust írunk rá, hanem a listához tartozó *find* névvel rendelkező metódust használjuk, melybe beleírhatjuk a feltételt.

A deklaratív vs imperatív programozás részletesebb összehasonlítása: [\[2\]](#)

Dart nyelvi irodalom: [\[3\]](#).

Teljes dart nyelvi dokumentáció: [\[4\]](#)

Teljes Flutter dokumentáció: [\[5\]](#)

3.1. Fordítás, futtatás

Az alkalmazás futtatható a 'flutter run' parancs segítségével. Ekkor az alkalmazás *debug* módban fog futni. Hogyha **release** módban akarjuk futtatni, ahhoz a 'flutter run -release' parancsot kell használnunk.

Android platformon ún. **flavor** típusok is vannak definiálva, melyekkel könnyebb az alkalmazás tesztelése. Röviden, Androidon az alkalmazásból 2 darab is futhat egyszerre. Két **flavor** érhető el: *dev* és *prod*. Alapesetben a **dev** flavorban fog indulni az alkalmazás, ekkor az eszközre telepített program neve 'ytp+ dev' lesz. Hogyha **prod** módban akarjuk futtatni az alkalmazást, ahhoz az alábbi parancsot kell használnunk: 'flutter run -flavor prod'. Ekkor az alkalmazás neve 'Youtube Playlists+'-ként fog megjelenni. Windows platformon nincs lehetőség **flavor** használatára.

Hogyha telepíthető / futtatható állományt szeretnénk készíteni a kódátról, azt platformról függően az alábbi parancsokkal tehetjük meg:

- **Android:** flutter build apk -release
- **Windows:** flutter build windows -release

A generált fájlok a *build* mappában lesznek elérhetőek.

Android platformon **flavor**-tól függően is generálhatunk telepíthető fájlt, kombinálva a -flavor kapcsolóval. Ezt használnunk is kell, hogyha új release-t akarunk github-ra rakni. Tehát, új github release-nél az alábbi parancsokat futtatva generálhatjuk ki a feltöltendő fájlokat:

- **Android:** flutter build apk -flavor prod -release
- **Windows:** flutter build windows -release

Majd az alábbi fájlokra lesz szükségünk:

- **Android:** build/app/outputs/flutter-apk/app-prod-release.apk
- **Windows:** build/windows/x64/runner/Release mappa teljes tartalma

3.2. Architektúra

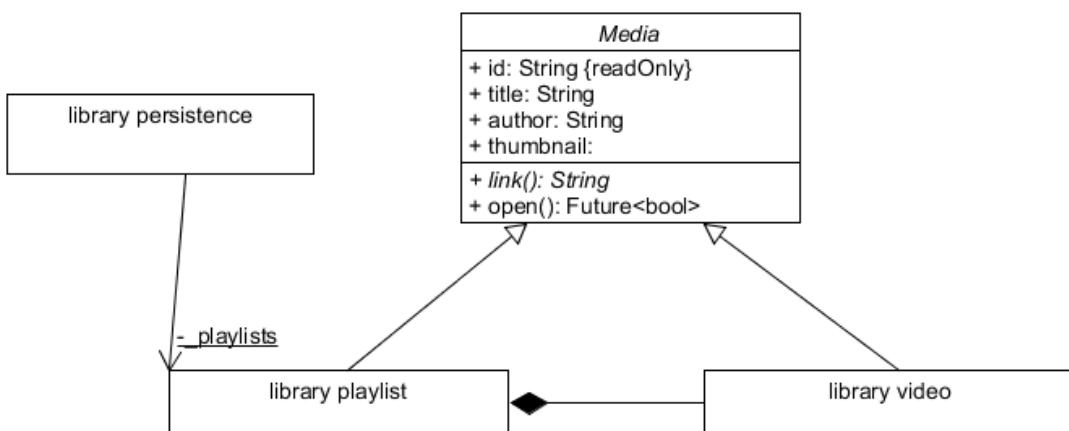
Az alkalmazás 4 fő rétegből épül fel:

- **Model** (3.3): Az alkalmazás fő logikája itt helyezkedik el.
- **Persistence (Perzisztencia)** (3.4): Az alkalmazás perzisztenciája, ahol az adatok tárolva vannak.
- **Provider, avagy NézetModel** (3.5): Az alkalmazás logikáját és nézetét összekötő réteg. Ezen keresztül kommunikál a Nézet a Modellel, ezen keresztül frissíti a Modell a Nézetet.
- **View (Nézet)** (3.6): Az alkalmazás megjelenítő rétege, ezen keresztül történik a felhasználó interakciója az alkalmazással.

Emellett fontos még megjegyezni a **Services / Szolgáltatások** (3.7) meglétét, mely különböző segédfeladatokat lát el, mint a Youtube-al való kommunikáció, illetve az Android-specifikus tevékenységek felállítása.

3.3. Model

Ez a réteg végzi az alkalmazás adatainak kezelését, illetve itt található a fő logika. Itt definiáljuk a legalapabb osztályokat, melyek az alkalmazáshoz szükségesek. A dokumentáció a legalapabb osztályokból indul ki, fokozatosan az egyre összetettebbek felé.



3.1. ábra. A modell egyszerűsített "könyvtár" diagramja. Az egyes könyvtárak részletesebb diagramjai a megfelelő fejezetekben szerepelnek.

3.3.1. Anchor (Rögzített pozíció)

Osztálydiagram: (ld. 3.2 ábra)

Az **Anchor** osztály egy módosíthatatlan osztály, mely 4 adattagot tartalmaz:

- final String **playlistId**: Annak a lejátszási listának az azonosítója, amelyhez az a videó tartozik, amelyhez az adott Anchor tartozik.
- final String **videoId**: Annak a videónak az azonosítója, amelyhez az adott Anchor tartozik.
- final AnchorPosition **position**: Az Anchor pozíciója, ahonnan számoljuk az eltérést. Ez egy enum 3 lehetséges értékkel:
 - **Start**: az eltérés a lejátszási lista elejétől számolódik.
 - **Middle**: az eltérés a lejátszási lista közepétől van számolva.
 - **End**: az eltérés a lejátszási lista végétől van számolva.
- final int **offset**: az eltérés az adott Anchor **position** adattaggjától

Az osztály ezeket a metódusokat definiálja:

- int get **index**: Visszaadja, hogy az adott Anchor az **offset** és **position** adattagok alapján a **playlistId** azonosítójú lejátszási listában hanyadik indexű elem helyét jelöli.

A különböző **position** értékek különböző intervallumot határoznak meg az **offset** adattagnak is:

- Start: [0.._playlistLength]
- Middle: [-_playlistLength / 2.._playlistLength / 2]
- End: [-_playlistLength..0]

Ezek alapján az implementáció:

```
1 int get index => switch (position) {  
2     AnchorPosition.start => offset,  
3     AnchorPosition.middle => _playlistLength ~/ 2 + offset,  
4     AnchorPosition.end => _playlistLength + offset,  
5 };
```

- int get _playlistLength: Visszaadja az Anchor lejátszási listájának hosszát.

3.3.2. Media

A **Media** osztály egy absztrakt osztály, mely definiálja az építőköveit a **Video** és **Playlist** osztályoknak.

4 adattagot tartalmaz, melyek az előzőek alapján a **Video** és **Playlist** osztályoknak is adattagjai lesznek:

- final String **id**: az adott Youtube tartalom azonosítója.
- String **title**: az adott Youtube tartalom címe.
- String **author**: az adott Youtube tartalom tulajdonosának neve.
- String **thumbnail**: az adott Youtube tartalom borítóképének az url-címe.

A **Media** osztály 2 metódust definiál:

- String get **link**: visszaadja a Youtube tartalom url-címét. Nem implementált, az alosztályoknak felül kell írni.

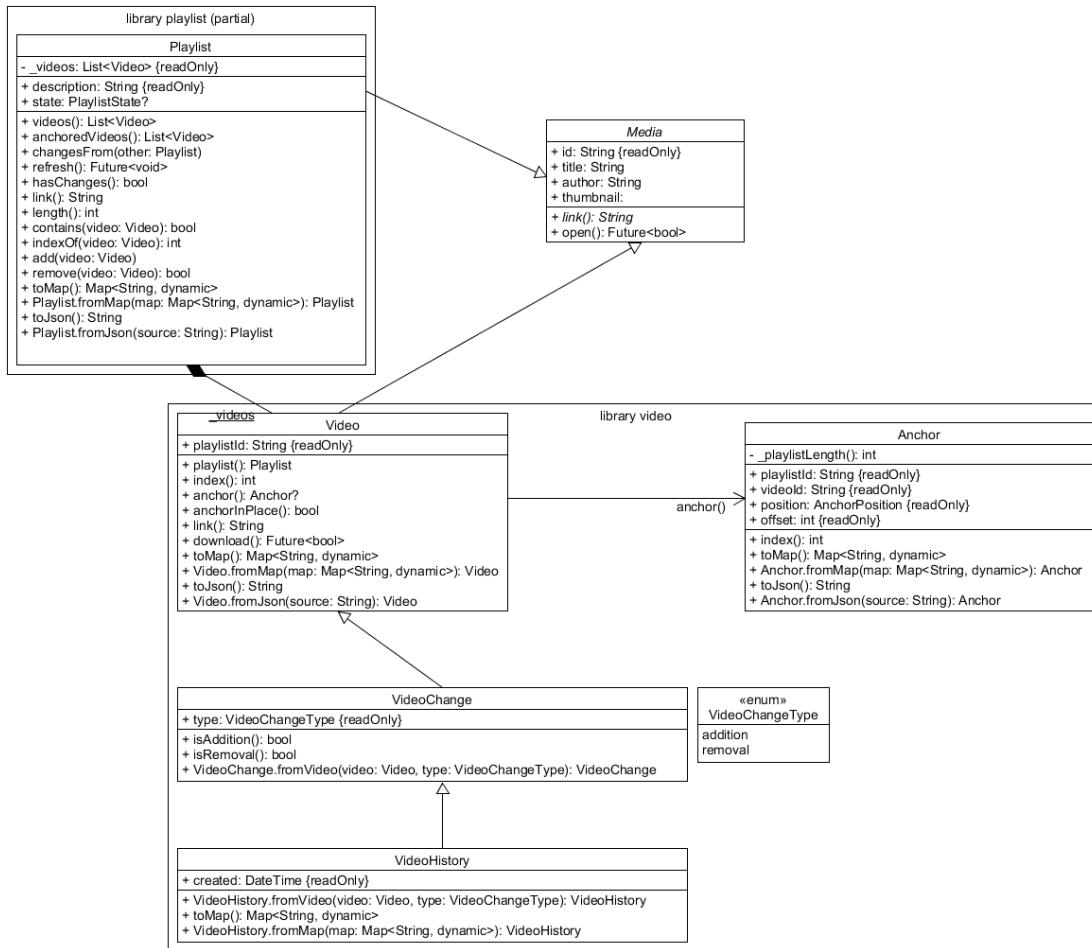
- Future<bool> **open()**: megnyitja az adott tartalmat Youtube-on.

```

1  /// Opens a 'Media' externally on 'Youtube'
2  Future<bool> open() async
3      => await launchUrl(Uri.parse(link));

```

3.3.3. Video



3.2. ábra. A Video osztály diagramja. (Az asszociált könyvtárak minimális részletezettséggel.)

A **Video** osztály a **Media** (3.3.2) osztályból terjeszt ki, így annak minden adattagját és metódusát tartalmazza.

Emellett 1 extra adattaggal egészíti ki a **Media** osztályt:

- final String **playlistId**: az adott videó lejátszási listájának azonosítója.

Továbbá ezekkel a metódusokkal rendelkezik:

- Playlist get **playlist**: visszaadja azt a lejátszási listát, melynek eleme a videó.
- int get **index**: visszaadja a videó indexét a lejátszási listájában.
- Anchor? get **anchor**: visszaadja a videóhoz tartozó anchor-t (pozicionális rögzítést), hogyha rendelkezik vele. Ellenkező esetben **null** a visszatérési érték.
- bool get **anchorInPlace**: visszaadja, hogy a videó az **anchor** adattagja alapján a megfelelő **index**-en helyezkedik el. Hogyha nem rendelkezik **anchor**-al (**null**), akkor **false** a visszatérési érték.
- String get **link**: visszaadja a videó url-címét. Felüldefiniált metódus a **Media** 3.3.2 ősosztályból.
- Future<bool> **download()**: letölti az adott videót.
- bool **operator==**(covariant Video): az == operátor felüldefiniálása, két videó akkor ekvivalens, hogyha az **id** egyenlő.
- int get **hashCode**: az object.hashCode felüldefiniálása, az **id** hashCode-ja.
- Map<String, dynamic> **toMap()**: átkonvertálja az adott videót egy json-elhető Map objektummá.

```

1  Map<String, dynamic> toMap() => <String, dynamic>{
2      'id': id,
3      'playlistId': playlistId,
4      'title': title,
5      'author': author,
6      'thumbnail': thumbnail,
7  };

```

- factory **Video.fromMap**(final Map<String, dynamic>): átkonvertál egy megfelelő Map objektumot egy **Video** objektummá.

```

1  factory Video.fromMap(final Map<String, dynamic> map) =>
2      Video(
3          id: map['id'] as String,
4          playlistId: map['playlistId'],
5          title: map['title'] as String,
6          author: map['author'] as String,
7          thumbnail: map['thumbnail'] as String,
8      );

```

- String **toJson()**: az adott videót egy **json** Stringé konvertál, a **toMap**-nek megfelelően.
- factory **Video.fromJson**(final String): egy megfelelő json Stringből létrehoz egy **Video** objektumot.

3.3.4. VideoChange (VideóVáltozás)

Az osztály egy kiterjesztése a **Video** (3.3.3) osztálynak, így annak minden adattagját és metódusát tartalmazza. Az osztály feladata egy lejátszási lista videó változásának a modellezése.

1 extra adattaggal egészíti ki a **Video** osztályt:

- final VideoChangeType **type**: a változás típusa, egy enum típus, 2 értékkel:
 - **addition**: a változás jelentése: az adott videót hozzáadták a lejátszási listához.
 - **removal**: a változás jelentése: a videót törölték a lejátszási listából.

Az enumnak 2 adattagja van, melyek a megjelenítésben játszanak szerepet; egy ikon (icon), és annak színe (color).

Példa egy ikonra a Nézetben:

```
1  const Icon(  
2      icon: videoChange.type.icon,  
3      color: videoChange.type.color,  
4  )
```

Két getter metódussal rendelkezik:

- bool get **isAddition**: igazzal tér vissza, hogyha a változás a videó hozzáadása a listához.
- bool get **isRemoval**: igazzal tér vissza, hogyha a változás a videó törlése a listából.

Rendelkezik egy factory metódussal is, mely a **type** megadásával egy **Video** objektumból **VideoChange** objektumot csinál:

```

1  factory VideoChange.fromVideo(
2      final Video video,
3      final VideoChangeType type,
4  ) =>
5
6      VideoChange(
7          id: video.id,
8          playlistId: video.playlistId,
9          title: video.title,
10         author: video.author,
11         thumbnail: video.thumbnail,
12         type: type,
13     );

```

3.3.5. VideoHistory (VideóElőzmény)

Ez az osztály a **VideoChange** (3.3.4) egy kiterjesztése, így annak minden adattagját és metódusát tartalmazza. Mivel törölt videóknál van rá esély, hogy a Youtube-ról törölték a videót, így ebben az esetben a videónak nem létezne url-címe, és borítóképje sem. Emiatt ennek az osztálynak a thumbnail adattagját nem használjuk.

1 extra adattaggal egészíti ki az ősosztályát:

- final DateTime **created**: az előzmény létrehozásának dátuma.

Metódusai:

- bool **operator==**(covariant VideoHistory): az == operátor felüldefiniálása a **Video** ősosztályból, két videóelőzmény akkor ekvivalens, hogyha az **id** egyenlő, illetve ugyanakkor készültek, azaz a **created** adattagok egyenlők.
- int get **hashCode**: a video.hashCode felüldefiniálása, az **id** és **created** hashCode-ja.
- Map<String, dynamic> **toMap()**: átkonvertálja az adott videóelőzményt egy json-elhető Map objektummá.

```

1      Map<String, dynamic> toMap() => <String, dynamic>{
2          'id': id,
3          'playlistId': playlistId,

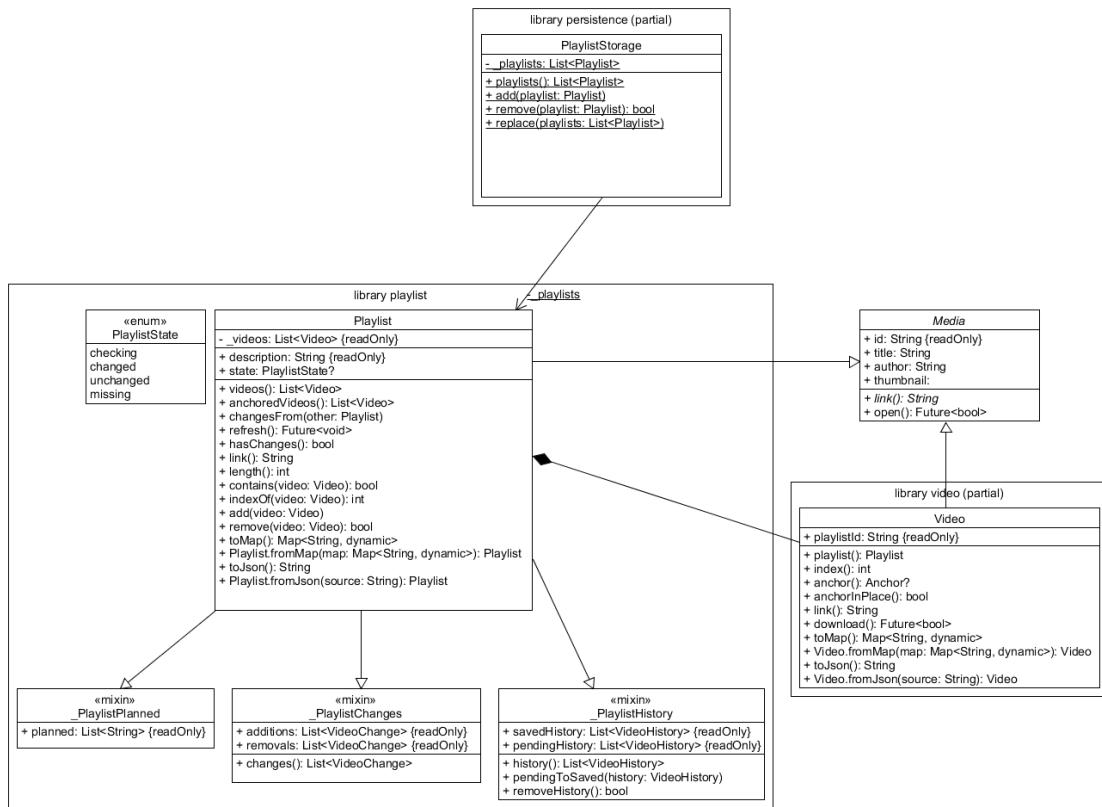
```

```
4     'title': title,
5     'author': author,
6     'type': type.index,
7     'created': created.millisecondsSinceEpoch,
8   };
```

- factory **VideoHistory.fromMap**(final Map<String, dynamic> map): átkonvertál egy megfelelő Map objektumot egy **VideoHistory** objektummá.

```
1   factory VideoHistory.fromMap(final Map<String, dynamic> map
2     ) => VideoHistory(
3       id: map['id'] as String,
4       playlistId: map['playlistId'] as String,
5       title: map['title'] as String,
6       author: map['author'] as String,
7       type: VideoChangeType.values[map['type'] as int],
8       created: DateTime.fromMillisecondsSinceEpoch(map['
9         created']) as int),
10    );
```

3.3.6. Playlist (Lejátszási Lista)



3.3. ábra. A Playlist osztály diagramja. (Az asszociált könyvtárak minimális részletességgel.)

Ez az osztály reprezentál és kezel egy Youtube lejátszási listát. A **Media** osztályból terjeszt ki, így annak minden adattagjával és metódusával rendelkezik.

Fontos itt megemlíteni a **Mixinek**-et. Ezek újrahasználható önálló kódrészleteket definiálnak, ezáltal használhatóak úgy, mint egy osztály önálló része. Mivel a **Playlist** osztály viszonylag sok adatot tartalmaz, fontosnak éreztem az önállóbb, szétszedhető részeket külön **Mixinek**be helyezni, hogy jobban átlátható legyen a különböző kódrészek feladata.

Emiatt a **Playlist** osztály a **Media** kiterjesztése mellett 3 **Mixin** osztállyal is kiegészülve alkot egy egészet:

- **_PlaylistChanges**: adott lejátszási lista változásait tárolja.
- **_PlaylistHistory**: adott lejátszási lista előzményeit tárolja és kezeli.
- **_PlaylistPlanned**: adott lejátszási lista tervezett listáját kezeli.

Innen től a **Playlist** osztályt a 3 **Mixin**nel együtt értelmezzük. Ezek alapján a **Playlist** osztály az **Media** ősosztályból örökolt adattagjain kívül ezekkel rendelkezik:

- final String **description**: a lejátszási lista leírása, Youtube-ról származó adat.
- PlaylistState? **state**: a lejátszási lista állapota, egy enum, mely a következő értékeket veheti fel:
 - **checking**: a lejátszási lista ellenőrzés alatt van éppen.
 - **unchanged**: a lejátszási lista le lett ellenőrize, és az alkalmazás nem talált változást.
 - **changed**: a lejátszási lista le lett ellenőrizve, és az alkalmazás talált változást.
 - **missing**: a lejátszási listát az alkalmazás nem találta meg ellenőrzés közben.

Ez az érték azonban **null** értéket is felvehet; ez azt jelenti, hogy a lejátszási lista még nem lett ellenőrize. A **PlaylistState** enum 3 adattaggal is rendelkezik, melyek a megjelenítésben segítenek:

- final IconData **icon**: az állapot ikonja
- final Color **color**: az állapot színe
- final String **message**: az állapothoz tartozó üzenet
- final List<Video> **_videos**: a lejátszási lista videóinak listája
- final List<VideoChange> **additions**: a lejátszási lista azon videót változásainak listája, melyek hozzáadottként vannak feltüntetve. Szükséges hozzá, hogy a lejátszási lista ellenőrizve legyen előtte.
- final List<VideoChange> **removals**: a lejátszási lista azon videót változásainak listája, melyek töröltként vannak feltüntetve. Szükséges hozzá, hogy a lejátszási lista ellenőrizve legyen előtte.
- final List<VideoHistory> **savedHistory**: a lejátszási lista azon videó előzményei, melyek már véglegesítve lettek a **Perzisztenciában** (3.4).

- final List<VideoHistory> **pendingHistory**: a lejátszási lista azon videóelőzémenyei, melyek még nem lettek véglegesítve a **Perzisztenciában** (3.4).
- final List<String> **planned**: a lejátszási lista tervezett videóinak listája. Fontos, hogy nem **Video** (3.3.3) objektumokat tárolunk itt, hanem Stringeket, melyekkel felismerhetőek a videók a felahsználó szerint.

Az alábbi metódusokkal rendelkezik:

- List<Video> get **videos**: visszaadja a privát **_videos** adattag egy módosíthatatlan változatát.
- List<Video> get **anchoredVideos**: visszaadja a **videos** lista azon Videóit, melyek Rögzítve vannak.
- void **changesFrom**(Playlist): összehasonlítja az adott lejátszási listát a paraméterben kapottal. Feltölti az **additions** és **removals** listákat megfelelően, illetve a **pendingHistory** listát is. Hogyha a paraméterben kapott lista nem egyezik meg a példánnyal (nem azonos az id), akkor a függvény nem ellenőriz, egyből visszatér. Hogyha nem talál változást a metódus, akkor a paraméterben kapott videók listájára cseréli ki a példány listáját, egységesítve a sorrendet.
- Future<void> **refresh()**: frissíti az adott lejátszási listát. Lekéri Youtube-ról a legfrissebb állapotot, majd a **changesFrom** metódussal összehasonlítja önmagával.
- bool get **hasChanges**: a **changes** adattag üres-e.
- bool **operator==**(covariant Playlist): az == operátor felüldefiniálása, két lejátszási lista akkor ekvivalens, hogyha az **id** egyenlő.
- Map<String, dynamic> **toMap()**: átkonvertálja az adott lejátszási listát egy json-elhető Map objektummá.

```

1  Map<String, dynamic> toMap() => <String, dynamic>{
2      'id': id,
3      'title': title,
4      'author': author,
5      'description': description,
6      'thumbnail': thumbnail,

```

```

7         'videos': [..._videos.map((final video) => video.
8             toMap())],
9         'history': [
10             ...(_savedHistory
11                 ..sort(
12                     (final fst, final snd) => snd.created.
13                     compareTo(fst.created),
14                     ))
15             .take(500)
16             .map((final history) => history.toMap())
17         ],
18         'planned': planned,
19     };

```

- factory **Playlist.fromMap**(final Map<String, dynamic>): átkonvertál egy megfelelő Map objektumot egy **Playlist** objektummá.

```

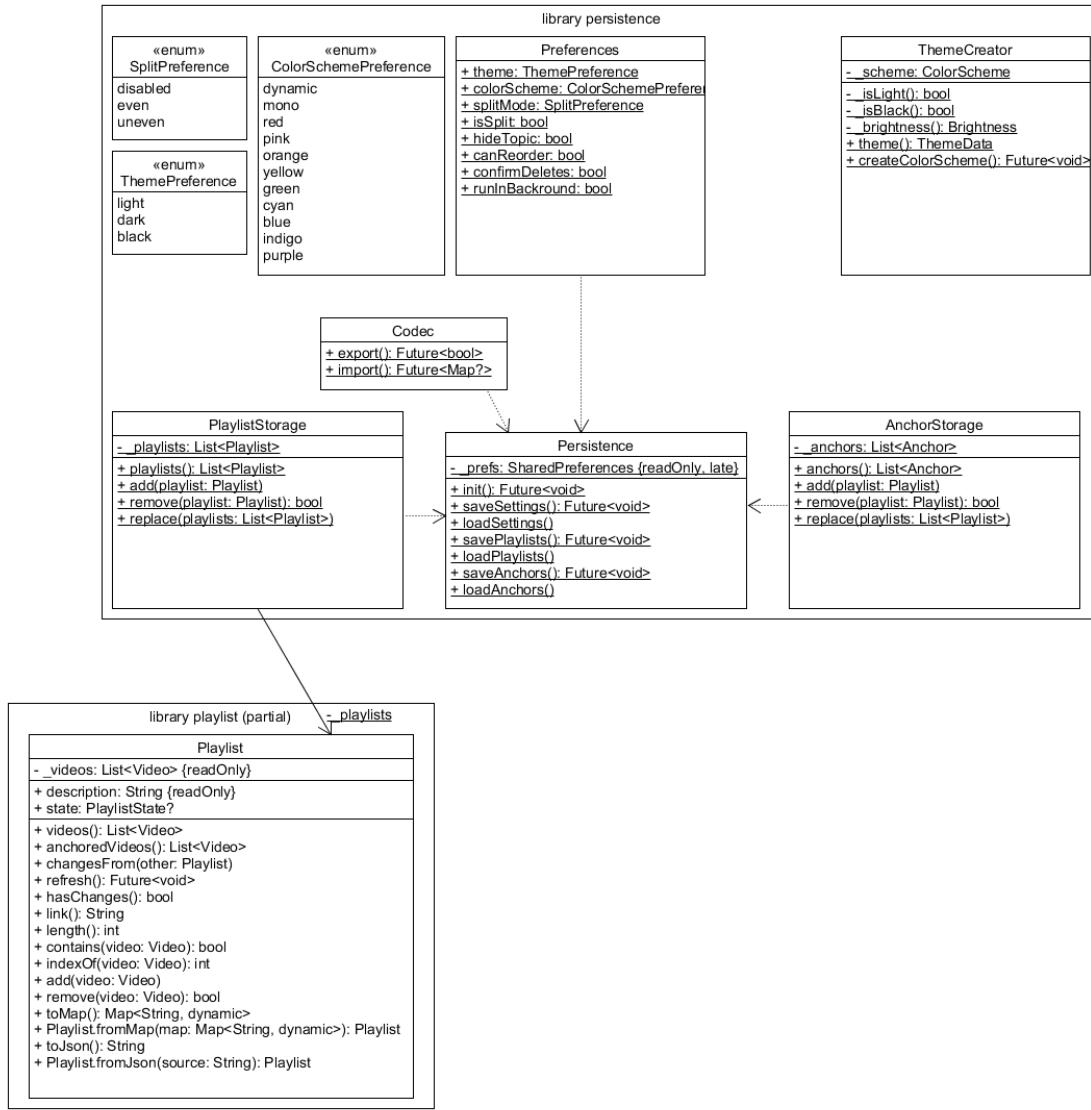
1     factory Playlist.fromMap(Map<String, dynamic> map) =>
2         Playlist(
3             id: map['id'] as String,
4             title: map['title'] as String,
5             author: map['author'] as String,
6             description: map['description'] as String,
7             thumbnail: map['thumbnail'] as String,
8             videos: List<Video>.from(
9                 (map['videos'] as List<dynamic>).map<Video>(
10                    (final map) => Video.fromMap(map as Map<String,
11                        dynamic>),
12                    ),
13                    ),
14                    ),
15                    .._savedHistory.addAll(
16                        List<VideoHistory>.from(
17                            (map['history'] as List<dynamic>).map<VideoHistory>(
18                                (final map) => VideoHistory.fromMap(map as Map<
19                                    String, dynamic>),
20                                ),
21                                ),
22                                ),
23                                .._planned.addAll(

```

```
21     List<String>.from(
22         (map['planned'] as List<dynamic>)
23         .map<String>((final map) => map as String),
24         ),
25     );
```

- String **toJson()**: az adott lejátszási listát egy **json** Stringé konvertál, a **toMap**-nek megfelelően.
- factory **Playlist.fromJson**(final String): egy megfelelő json Stringből létrehoz egy **Playlist** objektumot.

3.4. Persistence (Perzisztencia) könyvtár



3.4. ábra. A Perzisztencia osztály diagramja. (Az asszociált könyvtárak minimális részletezetességgel.)

A Perzisztencia, mint réteg 6 osztályból áll:

- **Persistence** (3.4.1): ő végzi az adatok mentését és betöltését.
- **Codec** (3.4.2): ez az osztály végzi az adatok exportálását / importálását a felhasználó által választott fájlba/-ból.
- **Preferences** (3.4.3): ez az osztály tárolja a felhasználó preferenciáit, mint például az alkalmazás színsémája.

- **ThemeCreator** (3.4.4): előkészíti és tárolja az alkalmazás témáját és színsémaját, amit a Nézet (3.6) használhat.
- **PlaylistStorage** (3.4.5): tárolja és kezeli az alkalmazás lejátszási listáit.
- **AnchorStorage** (3.4.6): tárolja és kezeli az alkalmazás pozicionálisan rögzített videóinak a rögzített pozícióját.

3.4.1. Persistence (Perzisztencia) osztály

Az alábbi adattagokkal rendelkezik:

- static String? **currentlyShowingPlaylistId**: az alkalmazás által jelenleg aktív lejátszási lista oldal (2.4) lejátszási listájának az azonosítója, vagy **null**, hogyha nincs aktív lejátszási lista oldal. Ez az adattag arra szolgál, hogy a nézet bezárja az éppen aktív lejátszási lista oldalt, hogyha annak lejátszási listáját éppen kitöröltük.
- static late final SharedPreferences **_prefs**: platform-specifikus implementációja az adatok tárolásának. Ezen keresztül lehet az adatokat kulcs-érték párok-ként lementeni. Web platformon például a localStorage-t használja.

Az osztály a következő metódusokkal rendelkezik:

- static Future<void> **init()**: inicializálja a **_prefs** adattagot későbbi használatra. Ennek a metódusnak a meghívása nélkül az osztály minden másra LateInitializationError-t fog dobni.
- static Future<void> **savePrefereces()**: elmenti a **Preferences** (3.4.3) osztály minden adattagját.
- static void **loadPreferences()**: betölti a **Preferences** (3.4.3) osztály minden adattagját, és át is állítja a **Preferences** osztály értékeit. Hogyha még nem voltak lementve adott adattagok, azok az eredeti értékükön maradnak.
- static Future<void> **savePlaylists()**: elmenti a **PlaylistStorage** **_playlists** adattagját.
- static void **loadPlaylists()**: betölti a **savePlaylists()** metódus által lementett lejátszási listákat a **PlaylistStorage**.**_playlists** tömbbe. Hogyha nem talál betölthető adatot, nem módosítja a tömböt.

- static Future<void> **saveAnchors()**: elmenti az **AnchorStorage._anchors** adattaggát.
- static void **loadAnchors()**: betölti a **saveAnchors()** metódus által elmentett adatokat, és behelyezi őket a **AnchorStorage._anchors** tömbbe. Hogyha nem talál betölthető adatot, nem módosítja a tömböt.

3.4.2. Codec

A **Codec** osztály végzi az alkalmazás adatainak fájlba való eltárolását és onnan betöltését. A különbség a **Persistence** osztálytól az, hogy a **Codec** osztályt a felhasználó irányítja. Exportálhatja az alkalmazás egy adott időbeli állapotát, majd egy másik eszközön a kiexportált fájlt beimportálhatja, és az alkalmazás állapota ugyanaz lesz minden eszközön

2 osztályszintű metódust definiálunk itt:

- static Future<bool> **export()**: a felhasználónak választania kell egy mappát az eszközén, ahol exportálhatja az adatait, majd az eszköz ebbe a mappába egy "ytp_export[dátum].json" nevű fájlt hoz létre, amibe exportálja az adatokat. A [dátum] az Epochtól kezdve eltelt micromásodpercek.
- static Future<Map?> **import()**: a felhasználónak ki kell választania egy ".json" kiterjesztésű fájlt, melyet ezután beolvas a metódus, és egy Map kulcsérték adatszerkezetben visszaadja a beolvasott alkalmazás állapotot. Hogyha bárhol elhasalna a beolvasás, akkor **null** a visszatérési érték.

3.4.3. Preferences (Preferenciák)

A **Preferences** osztály tartalmazza a felhasználó preferenciáit az alkalmazással kapcsolatban. Ezeket a **Preferences** (2.5) panelen tudja módosítani a **Főoldalon**. Az egyes preferenciák általában 2 opciónális: egy **bool**-ként, több opciónális esetén pedig egy **enum**-ként vannak ábrázolva.

Az osztály ezek alapján az alábbi osztályszintű adattagokkal rendelkezik:

- static ThemePreference **theme**: Az alkalmazás témája, mely a világosságát vagy sötétségét jelenti. A **ThemePreferences** enum típus a következő értékeket veheti fel:

- **light**: világos mód
- **dark**: sötét mód
- **black**: fekete mód, OLED képernyők esetén kímélőbb.
- static ColorSchemePreference **colorScheme**: Az alkalmazás színsémája, ez adja, hogy a **téma** mellett milyen színben jelenjen meg az alkalmazás. A **ColorSchemePreference** enum típus a következő értékeket veheti fel:
 - **dynamic**: ekkor az alkalmazás dinamikusan igazodik az eszközhöz, hogyha az támogatja. Ez Windows esetén az operációs rendszer színéből keveri ki, Android esetén pedig a Material You alapján az eszköz hátteréből keveri ki a színsémát. Hogyha nem sikerül beállítani a dinamikus színt, akkor kék színt állít be.
 - **mono**: Az alkalmazás monokromatikus színsémát állít be, azaz fekete, fehér illetve szürke színek lesznek a dominánsak.
 - **red** (piros): az alkalmazás domináns színe a piros lesz.
 - **pink** (rózsaszín): az alkalmazás domináns színe a rózsaszín lesz.
 - **orange** (narancs): az alkalmazás domináns színe a narancssárga lesz.
 - **yellow** (citromsárga): az alkalmazás domináns színe a citromsárga lesz.
 - **green** (zöld): az alkalmazás domináns színe a zöld lesz.
 - **cyan** (ciánkék): az alkalmazás domináns színe a ciánkék lesz.
 - **blue** (kék): az alkalmazás domináns színe a kék lesz.
 - **indigo** (indigókék): az alkalmazás domináns színe a indigókék lesz.
 - **purple** (lila): az alkalmazás domináns színe a lila lesz.

Az enum rendelkezik egy **color** adattaggal, mely az enumnak megfelelő színt rendeli hozzá. Dynamic enum esetén ez az érték null, Mono esetén pedig Colors.grey. minden más esetben az enum nevével azonos szín.

3.4.4. ThemeCreator

A **ThemeCreator** osztály egy statikus osztály, mely összeállítja az alkalmazás megjelenítési stílusát. Ez a világosság és színséma mellett bizonyos beépített kom-

ponensek tulajdonságait is jelenti. Ehhez a **ThemeData** osztály alapján állítja be a keretrendszer az alkalmazás stílusait, lényegében felfogható egy alap CSS-nek is.

Egyetlen statikus privát adattaggal rendelkezik, a `_scheme` ColorScheme típusú adattaggal, mely az alkalmazás színsémáját tárolja. Ez a változó kezdetben nincs inicializálva, így az alkalmazás indításakor kell ezt megtennünk, mielőtt bármilyen megjelenítési komponenst (Widget-et) kirajzolnánk.

Ehhez meghívjuk a statikus `createColorScheme()` Future<void> visszatérésű metódust, mely inicializálja a `_scheme` adattagot, a **Preferences**.colorScheme alapján. Az osztály egy Future, aszinkron függvény, mivel **dynamic** színséma választása esetén aszinkron metódusokkal kell dolgoznunk. Mivel ez a függvény keveri ki az alkalmazás színeit, az alkalmazás indításán kívül minden alkalommal meghívjuk, amikor megváltoztatjuk az színsémát.

Az osztály 3 db statikus privát gettert tartalmaz:

- static bool `_isLight`: visszaadja, hogy a **Preferences**.theme alapján világos-e a téma.
- static bool `_isBlack`: visszaadja, hogy a **Preferences**.theme alapján fekete-e a téma.
- static Brightness `_brightness`: visszaadja, hogy az alkalmazás témája világos vagy sötét

Egyetlen publikus getterrel rendelkezik:

- static ThemeData get `theme`: visszaadja az alkalmazás témáját, melynek világosságát a `_brightness` és `_isBlack` alapján, színsémáját pedig a `_scheme` alapján állítja össze. Működésének feltétele a `_scheme` adattag inicializáltsága. Az adattagot az alkalmazás komponensfájának gyökerében, a **MaterialApp**.theme adattagjaként használjuk.

3.4.5. PlaylistStorage

Ez az osztály egy tároló osztály, egyetlen statikus privát adattaggal: `_playlists`. Ez egy static final List<Playlist> típusú adattag, melyben az alkalmazás összes eltárolt lejátszási listája (3.3.6) található.

Az adattagot egy getteren keresztül érjük el, ez a **playlists**. A getter egy módosíthatatlan listát ad vissza, garantálva ezzel, hogy ne lehessen rosszul hozzájárni a listához.

Mivel csak egy módosíthatatlan listával tudjuk visszakapni a lejátszási listákat, így 3 metódust is definiál az osztály, melyekkel a szükséges módosításokat tudjuk végrehajtani.

- static void **add**(Playlist): hozzáad egy lejátszási listát a **_playlist** listához.
- static bool **remove**(Playlist): kivesz egy elemet a **_playlist** listából, hogyha az egyezik a paraméterben kapottal (azaz egyezik az **id**). Hogyha sikeres volt a törlés, igazzal tér vissza, ellenkező esetben hamissal.
- static void **replace**(List<Playlist>): kicséríti a **_playlist** lista minden elemét a paraméterben kapott elemekre. Ehhez kiüríti az adattagot, majd feltölti a paraméter elemeivel.

3.4.6. AnchorStorage

Hasonlóan tároló osztály, mint a **PlaylistStorage** (3.4.5), viszont lejátszási listák helyett **Anchor** objektumok listáját tárolja.

Így hasonló módon rendelkezik egy statikus privát List<Anchor> **_anchors** adattaggal, melyet egy publikus getterrel, **anchors** néven érünk el egy módosíthatatlan tömbként.

Emellett, 3 függvényel rendelkezik, melyek módosítják a privát adattagot:

- static void **change**(Anchor): berak a **_anchors** listába egy új **Anchor** (3.3.1) elemet. Hogyha már volt ilyen elem a listában, akkor azt pedig kiveszi.
- static bool **remove**(Anchor): kiveszi a paraméterben kapott elemet a listából, hogyha benne van. Visszatér azzal, hogy az elem benne volt-e a listában.
- static void **replace**(List<Anchor>): kicséríti a privát adattag elemeit a paraméterben kapott lista elemeivel.

3.5. Provider (ViewModel / NézetModell)

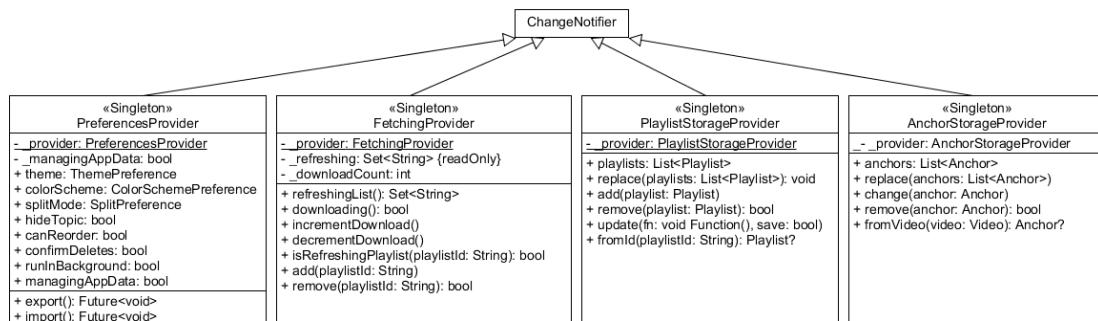
A Provider réteg adja a Modelt és Nézetet összekapcsoló réteget, így nevezhető ez a réteg a NézetModell rétegnak is. Mivel viszont a keretrendszer Provider néven használja ezt az állapot-kezelő megoldást, így a dokumentációban a továbbiakban is így lesz hivatkozva.

A Provider réteg tehát kezeli a modellt, melyben történt változásokat visszaad a nézetnek. Visszafele pedig, a Nézet a Provideren keresztül lép kapcsolatba közvetett módon a modellel. Erre azért van szükség, mert a modell csupán önmagáért felelve, nem értesíti a Nézetet, amikor megváltoztatja az állapotát. A Provider réteg viszont miután megváltoztatta a Modelt, értesíti a Nézetet a változásról, hogy az frissíteni tudja magát az új adatokkal. Így teljesen elkülöníthető a logika és a megjelenítés.

Az értesítéseket az összes Provider osztály a **ChangeNotifier** osztály kiterjesztésével éri el, melynek **notifyListeners()** metódusa értesíti az összes feliratkozott Nézet komponenst, akik ez után frissülnek. A komponensek feliratkozásáról a **Nézet** 3.6 részben lesz kitérve.

Mivel a **notifyListeners()** függvény egy példányszintű metódus, nekünk pedig minden Provider osztályunk statikus adattagokkal dolgozik, így minden Provider osztályunk a **Singleton** programozási sémát használja, hogy az alkalmazás minden pontján elérve ugyanazokat az adatokat érjük el. Emellett, ellenkező esetben azzal is probléma lenne, hogy ugyanaz a Provider osztály két külön példányára van feliratkozva két komponens, viszont csak az egyik frissül, pedig minden kellene.

Minden Provider osztály, mely a modellt egy osztályát kezeli, az [osztálynév]Provider konvencióval lett elnevezve. Emellett minden Provider osztály vége "Provider" szóval végződik.



3.5.1. AnchorStorageProvider

Az első Provider osztályunk, melynek feladata az **AnchorStorage** 3.4.6 osztályval való kommunikáció. Az osztály lényegében körbefoglalja a modellből származó osztályát.

Egyetlen gettere visszaadja az **AnchorStorage.anchors** gettert.

Emellett úgyanúgy definiálja az **AnchorStorage** 3 módosító függvényét, a **change**, **remove** és **replace** függvényeket, azzal a különbséggel, hogy miután meghívta a megfelelő függvényekben a modellosztály megfelelő függvényét, értesíti a Nézetet, majd elmenti a Perzisztenciában a módosításokat. Egy példa a kódból:

```

1  /// Removes an [Anchor]
2  bool remove(final Anchor anchor) {
3      final result = AnchorStorage.remove(anchor);
4      notifyListeners();
5      Persistence.saveAnchors();
6      return result;
7  }

```

Emellett egyetlen funkcióval bővül a modellosztályához képest:

- **Anchor? fromVideo(Video):** visszaadja a paraméterben kapott videóhoz tartozó Anchor objektumot, hogyha létezik. Ellenkező esetben null-al tér vissza.

3.5.2. PlaylistStorageProvider

A **PlaylistStorage** 3.4.5 osztály Provider osztálya, hasonlóan az **AnchorStorageProvider** osztályhoz, definiál egy gettert a **PlaylistStorage.playlists** getterhez, illetve minden módosítófüggvényét felüldefiniálja.

Emellett 2 új metódussal egészül ki:

- **void update(void Function(), bool):** mivel az Anchor objektumokkal ellentétben a Playlist objektumok módosíthatóak, így szükség volt egy olyan metódusra, mellyel az egyes Playlist objektumok módosításairól is tudunk értesítést küldeni. Hasonló megoldás a keretrendszer beépített **setState()** metódusához, egy függvényt adunk át, melyben a módosításokat végezzük, majd a módosítások után értesítjük a feliratkozott komponenseket. A bool paraméterben megadhatjuk, hogy a függvény elmentse-e a változtatásokat vagy sem.

```

1  /// Notifies after calling a callback
2  ///
3  /// Used to update [Playlist]s with
4  void update(final void Function() fn, {bool save = false}) {
5    fn();
6    notifyListeners();
7
8    if (save) {
9      Persistence.savePlaylists();
10   }
11 }

```

- Playlist? **fromId**(String): visszaadja a paraméterben kapott **id**-vel rendelkező lejátszási listát, hogyha az létezik. Ellenkező esetben **null** a visszatérési érték.

3.5.3. PreferencesProvider

A **Preferences** 3.4.3 osztályt kezelő Provider osztály. Feladata, hogy a felhasználó által tett személyre szabhatósági beállításokat elmentse és megjelenítse. Ehhez a **Preferences** osztály alábbi tagjaihoz definiál publikus getter és settert:

- ThemePreference **theme**: az alkalmazás témája ¹
- ColorSchemePreference **colorScheme**: az alkalmazás színsémája ¹
- SplitPreference **splitMode**: az alkalmazás osztott módban jelenjen-e meg, hogyha elég nagy a képernyő, illetve milyen arányban legyen elosztva a képernyő. ¹
- bool **hideTopic**: a Youtube csatornák nevéről levágja-e a ' - Topic' jelzőt ¹
- bool **canReorder**: jelenleg az alkalmazás lejátszási listái rendezhetőek-e ²
- bool **confirmDeletes**: megerősítést vár minden törlés előtt: lejátszási lista törlése, videóelőzmény törlése éstervezett videó törlése esetén ¹.
- bool **runInBackground**: az alkalmazás futhat-e a háttérben. ¹

¹Módosításkor értesíti a nézetet és elmenti az értéket a Perzisztenciában.

²Változáskor értesíti a nézetet, elmenti a lejátszási listákat, hogyha hamisra vált

Egy saját adattagja a **managingAppData** bool. Ennek feladata, hogy ki-bekapcsolja az exportálást/importálást kezelő gombokat a nézeten, míg egy fájlba exportáljuk az adatainkat.

Emellett rendelkezik 2 függvénnyel is:

- Future<void> **export()**: meghívja a **Codec.export()** 3.4.2 függvényt.
- Future<void> **import()**: átállítja a **managingAppData** értékt igazra, majd meghívja a **Codec.import()** függvényt. Miután visszakapott egy valid értéket, hamisra állítja a **managingAppData** értéket, majd beállítja/betölti az importált adatokat.

3.5.4. FetchingProvider

Feladata, hogy kezelje a frissítés és letöltés alatt lévő listák számát, és erről értesítse a nézetet. Ezt 2 privát adattagjával teszi:

- final Set<String> **_refreshing**: a jelenleg frissítés alatt lévő lejátszási listák **id**-jét tartalmazó halmaz.
- int **_downloadCount**: a jelenleg letöltés alatt lévő lejátszási listák számát tárolja.

A fenti adattagokat pedig az alábbi publikus getterek segítségével használhatjuk:

- Set<String> get **refreshingList**: visszaadja egy módosíthatatlan másolatát a **_refreshing** adattagnak.
- bool get **downloading**: visszaadja, hogy van-e jelenleg letöltés alatt lévő lejátszási lista, azaz a **_downloadCount > 0**.

A **_downloadCount** értékét az alábbi publikus függvényekkel változtathatjuk:

- void **incrementDownload()**: megnöveli a **_downloadCount** értékét 1-el.
- void **decrementDownload()**: csökkenti a **_downloadCount** értékét 1-el, hogyha 0-nál nagyobb

Mindkét metódus sikeres módosítás esetén jelez a Nézetnek, mely a **downloading** getter által észlelhető.

A **_refreshingList** tartalmát az alábbi publikus metódusok használják:

- bool **isRefreshingPlaylist**(String playlistId): visszaadja, hogy egy adott azonosítójú lejátszási lista éppen frissül-e
- void **add**(String playlistId): hozzáadja a `_refreshingList` halmazhoz a playlistId Stringet, majd értesíti a Nézetet.
- void **remove**(String playlistId): kiveszi a `_refreshingList` halmazból a playlistId Stringet, majd értesíti a Nézetet.

3.6. View (Nézet)

Ebben a részben a Nézet réteg lényegesebb elemei lesznek kifejtve. Itt nem lesz minden osztály (komponens) részletezve, helyette az átfogóbb, nagyobb összefüggő egységek együttesen lesznek leírva.

A Nézet építőkövei a **Widget**-ek, melyek a komponensekként foghatóak fel. minden Widget egy külön osztály, melyek főleg 2 osztályból származnak: StatelessWidget vagy StatefulWidget; az előbbi olyan komponenseknél alkalmazzuk, melyeknek nincs változó belső állapota, utóbbi pedig olyanokra, melyeknek van változó állapota, és időnként frissíteni kell miatta a nézetet.

Komplexebb állapot-kezelés (state-management) esetén azonban hamar nehézségekbe ütközhetünk, ha csak ezekre az eszközökre szorítkozunk. A **Provider** [6] csomag ad segítséget arra, hogy egy komponens feliratkozzon állapotváltozásokra, és frissítse magát.

Egy komponens Provider segítségével az alábbi módon képes feliratkozni egy ChangeNotifierból kiterjesztett osztályra: a komponens **build()** metódusában meg-hívjuk a `context.watch<ProviderClassName>()`; függvényt. Ezzel minden alkalommal, amikor a **ProviderClassName** osztály meghívja a `notifyListeners()` függvényét, az adott komponens újraraírja önmagát.

A továbbiakban a **vastagbetűs** szavak az egyes komponenseket fogják jelölni.

3.6.1. HomePage (Főoldal)



3.5. ábra. A főoldal mintaterve

A **HomePage** az alkalmazás kezdőoldala. Innen érhető el az összes lementett lejátszási lista. Ezt egy **PlaylistListView** jeleníti meg, ahol minden egyes lejátszási listát egy **PlaylistItem** reprezentál.

Az oldal rendelkezik qgy **PreferencesDrawer** komponenssel, mely a bal oldalról húzható ki, vagy a bal-felső gombbal előhívható. Ebben a komponensben lehet a felhasználói preferenciákat beállítani. minden egyes beállítás külön komponenre van szedve, minden egyes beállítás a lib/view/pages/home_page/drawer/preferences mappában található. Ugyanitt érhetők az exportálásra és importálásra használható **CodecButtons** gombok.

Az oldal jobb alsó részén található egy lebegő gomb, mellyel a **SearchPage**-re, a keresőoldalra juthatunk. Ugyanez a gomb átváltozik, hogyha éppen átrendezzük a lejátszási listáinkat; ekkor az átrendezés befejezésére szolgál.

3.6.2. SearchPage (Keresőoldal)

Ezen az oldalon tudunk lejátszási listákat keresni Youtube-on, majd azokat hozzáadni. A keresést a **SearchEngine** osztály hajtja végre. Ennek publi-

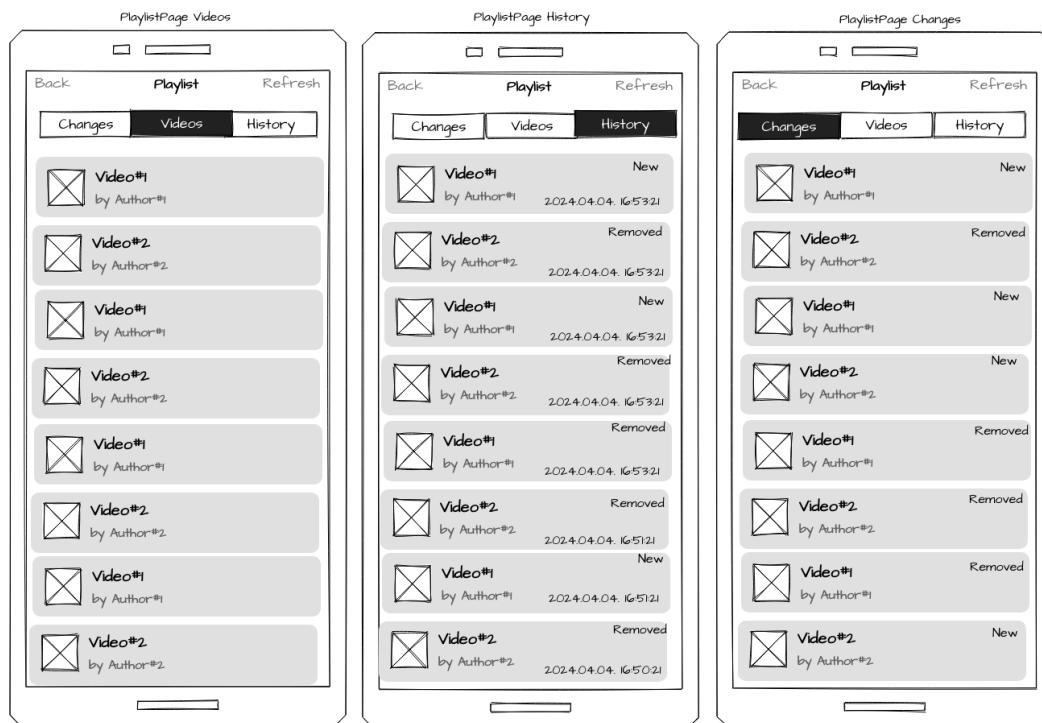
kus `search()` metódusát használja az oldal. Ez a metódus pedig visszatér egy `Future<List<Playlist>>-el`. Ennek a `Future`-nak az eredményét bevárva, az összes listaelemet megjelenítjük egy-egy **SearchResult** komponensként.

A **SearchResult** komponens kattintásra letölти az adott lejátszási lista adatait, és hozzáadja azt a Perzisztenciához, mely megjelenik a főoldalon is.

3.6.3. PlaylistPage (Lejátszási lista oldal)

Ezen az oldalon található egy adott lejátszási listának az összes adata. Ez az oldal egy 3 füllel rendelkező oldal, jobbról haladva a **ChangesTab**, **VideosTab** és a **HistoryTab**. Ezeket a füleket a **DefaultTabController** beépített flutter komponens kezeli.

A fülek felépítése nagyon hasonló, mindegyik egy-egy lista nézetet jelenít meg. Lényegi különbség a megjelenített listaelemek között van. A **ChangesTab** megjeleníthet **ChangeItem** vagy **AnchorItem** komponenseket, attól függően, hogy milyen változások történtek a lejátszási listával. A **VideosTab** **VideoItem** elemeket jelenít meg, a lejátszási lista összes videóját. A **HistoryTab** pedig **HistoryItem**-eket, melyek megjelenítik a dátumot és a változás típusát.



3.6. ábra. A lejátszási lista oldal mintaterve

3.6.4. AboutPage (Információs oldal)

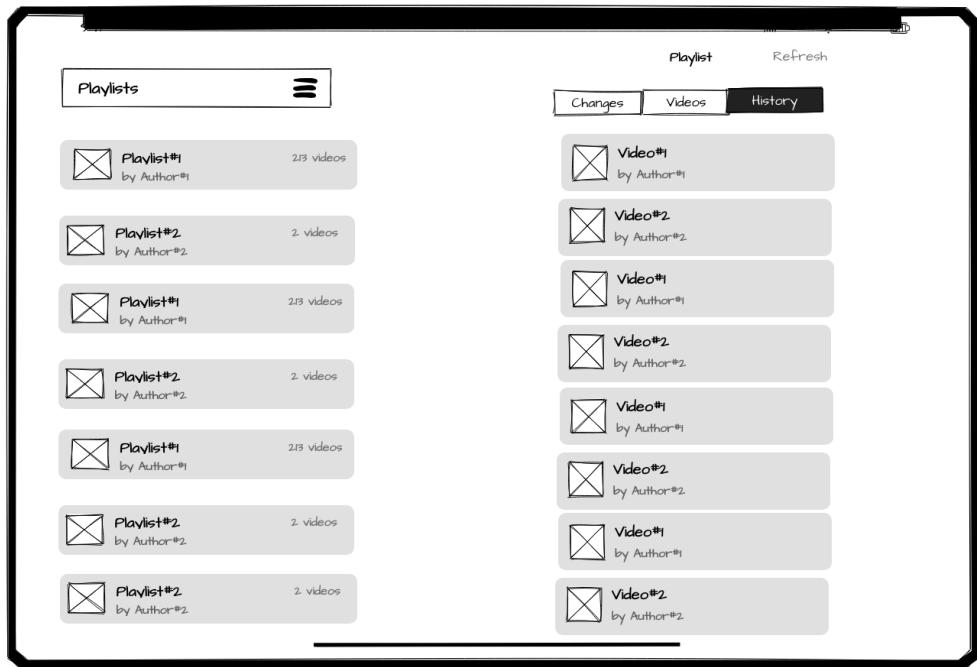
Ez az oldal segítséget próbál nyújtani a kezdő felhasználónak. Itt ugyanis le van írva egy egyszerű útmutató az alkalmazás használatáról és funkcióiról.

Ugyanezen az oldalon érhetők el az alkalmazás licenszei is; a jobb alsó sarokban található gombbal lehet oda elnavigálni. Itt az összes package és használt kód licensze megtalálható.

3.6.5. Other widgets (Egyéb komponensek)

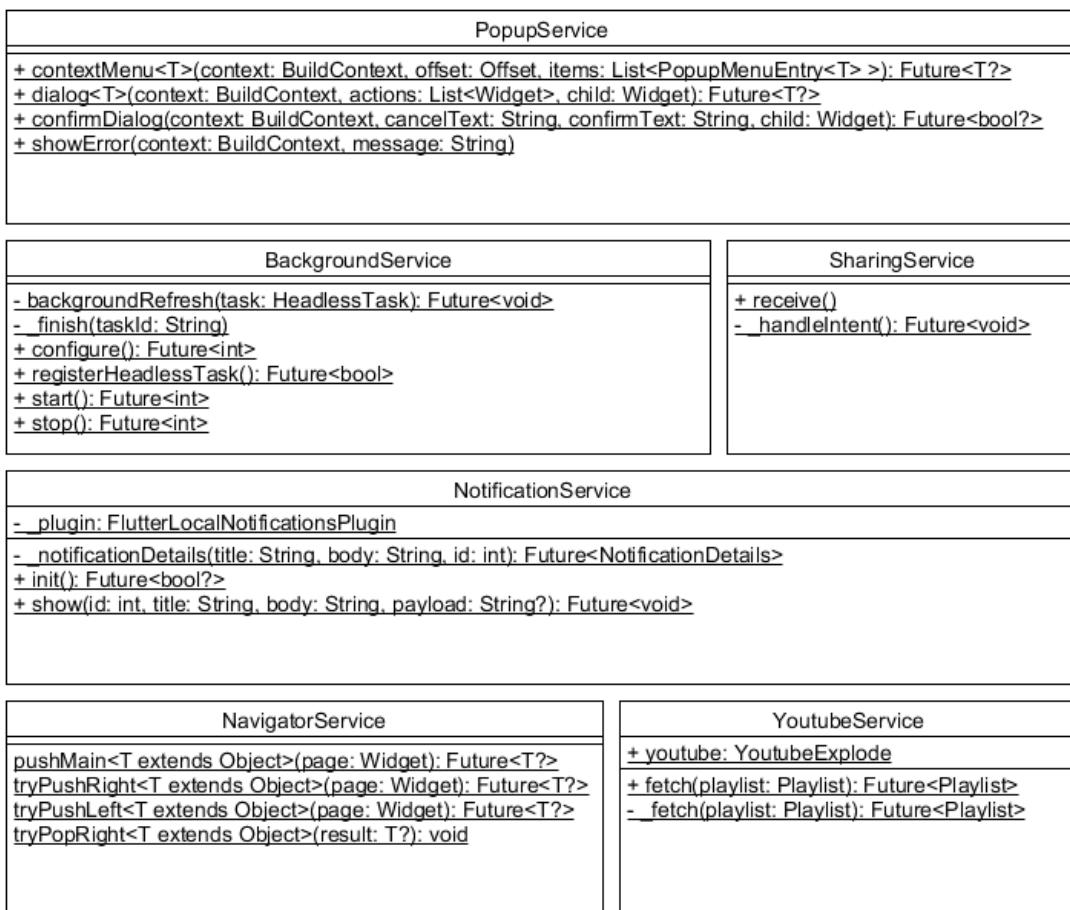
Ebben a részben lesznek kifejtve olyan komponensek, melyek nem feltétlenül tartoznak egy-egy adott oldalhoz. Ezek a komponensek a lib/widget mappában találhatóak.

- **Thumbnail:** ez a komponens videók és lejátszási listák borítóképeként használható.
- **AdaptiveSecondaryInkWell:** az InkWll bepített komponens egy átalakítása, mely egy **onPrimary** és **onSecondary** akcióval rendelkezik. Az előbbi kattintásra/érintésre aktiválható, utóbbi pedig jobb-kliikk / nyomvatartás hatására, platformtól függően (Windows/Android).
- **MediaItemTemplate:** egy alap vázat képez a legtöbb Media-alapú komponensnek, mint a **PlaylistItem** vagy a **VideoItem**. A hátteret és funkcionálitást építi ki.
- **FadingListView:** egy olyan listanézet, mely görgetés hatására elhalványítja a lista alsó és/vagy felső részét, beállítás alapján.
- **SplitView** és **Responsive:** azon 2 komponens, melyek a képernyő kettéosztását végzik. a **Responsive** a képernyő mérete és beállítás alapján jeleníti meg a **HomeScreen** oldalt, vagy a **SplitView** nézetet, mely bal oldalán a **HomePage**, jobb oldalán pedig egy ideiglenes komponens, vagy egy **PlaylistPage** kerülhet.



3.7. ábra. Az osztott nézet mintaterve

3.7. Services (Szolgáltatások)



3.8. ábra. A Szolgáltatások osztálydiagramja

A **Services** réteg az alkalmazás azon osztályait tartalmazza, melyek a többi rétegbe nem illettek be, mivel egy azoktól különálló, általában segédfeladatot végeznek. Emellett közös tulajdonságuk, hogy mind csak osztályszintű elemekkel rendelkeznek, és a nevük 'Service'-el végződik.

3.7.1. BackgroundService

A **BackgroundService** osztály kezeli az alkalmazás háttérben való futását. Mivel a funkció csak Android platformon érhető el, így az osztály függvényeinek meghívása előtt fontos ellenőriznünk, hogy a megfelelő platformon vagyunk-e. Ezt a következő módon tehetjük:

```
1  if(Platform.isAndroid){  
2      BackgroundService.configure();  
3  }
```

Az alábbi 6 függvényel kezeljük a folyamatot:

- static Future<int> **configure()**: beállítja a háttérben való futáshoz szükséges opciókat, mint az időzítést (1440 perc), hogy eszköz újraindítás után a folyamat is elinduljon, és hogy milyen hálózati kapcsolat szükséges.
- static Future<bool> **registerHeadlessTask()**: beállítja a **_backgroundRefresh()** függvényt, mint háttérben futtatható metódust.
- static void **_finish(String taskId)**: befelyez egy megadott taskId-vel rendelkező háttérfolyamatot.
- static Future<int> **start()**: elindítja a háttérben való futást.
- static Future<int> **stop()**: leállítja a háttérben való futást.
- static Future<void> **_backgroundRefresh(HeadlessTask task)**: ez a függvény végzi a háttérfolyamatot; lekéri a **Persistence**-ből az összes lejátszási listát, leellenőrzi őket, majd ha változást észlel, küld egy értesítést a **NotificationService** (3.7.2) segítségével. A metódus a **@pragma('vm:entry-point')** annotációval van ellátva, mellyel a fordító nem fogja eldobni, mint nem használt metódust. Ennek részletei innen [7] elérhetőek.

3.7.2. NotificationService

Az osztály feladata Android platformon az értesítések küldése. Ehhez 2 függvénytel rendelkezik:

- static Future<bool?> **init()**: ez a függvény inicializálja az osztályt, és a plugin-t, amivel értesítéseket tudunk küldeni.
- static Future<void> **show(int id = 0, String title, String body, String? payload)**: a függvény segítségével küldhetünk az Android eszközre egy értesítést **id** azonosítóval, **title** címmel és **body** tartalommal. A **payload** adattag pedig arra használható, hogy speciális utasítás adjunk az alkalmazásnak, hogyha az értesítésen keresztül nyitjuk meg. A metódus a **@pragma('vm:entry-point')** annotációval van ellátva, mellyel a fordító nem fogja eldobni, mint nem használt metódust. Ennek részletei innen [7] elérhetőek.

3.7.3. PopupService

Az osztály feladata az alkalmazásban való különböző felugró ablakokat egy közös egységbe foglalni és kezelní.

Metódusok:

- static Future<T?> **contextMenu<T>(BuildContext context, Offset offset, List<PopupMenuEntry<T> > items)**: egy kontextusmenüt hoz elő az alkalmazásban **offset** pozícióban, **items** opciókkal. A függvény visszatérése **T?**, ahol T generikus, és null, hogyha a menüből nem választottunk semmit.
- static Future<T?> **dialog(BuildContext context, List<Widget> actions, Widget child)**: a képernyő közepére hoz fel egy dialógusablakot.
- static Future<bool?> **confirmDialog(BuildContext context, String confirmText = "Proceed", String cancelText = "Cancel", Widget child)**: a **dialog** egy szűkebb változata, egy olyan dialógusablak, melyet elfogadni/elutasítani lehet csak
- static void **showError(BuildContext context, String message)**: a képernyő aljáról felugró hibaüzenetet jelenít meg **message** üzenettel

3.7.4. SharingService

Az osztály az alkalmazásnak küldött megosztások kezeléséért felel.

- static void **receive()**: fogadja a megosztás tartalmát, majd továbbküldi a **_handleIntent()**-nek
- static Future<void> **_handleIntent()**: feldolgozza a megosztott tartalmat

3.7.5. YoutubeService

Az osztály feladata a Youtube-al való kommunikáció lebonyolítása.

Egyetlen adattagja a statikus **youtube**, mely egy youtube kliens, melyen keresztül történik az adatok lekérése.

Emellett egy privát **_fetch()** metódussal rendelkezik, mely egy lejátszási lista adatait tölti le.

Az előbbi metódust pedig a publikus **fetch()** metódussal érjük el, mely visszatér a **_fetch()** értékével, viszont a függvényt egy külön izolált környezetben (Isolate) futtatja le, elkerülve ezzel a Nézet réteg blokkolását és befagyasztását.

3.7.6. NavigatorService

Ez a szolgáltatás kezeli az alkalmazás oldalak közti navigálását. Mivel az alkalmazás egyszerre képes több oldal megjelenítésére az osztott nézet beállítással 3.7, így a beépített **Navigator** osztály már körülményes kezdett lenni.

Ezért a **NavigatorService** ezt a beépített osztályt alkalmazva definiál olyan metódusokat, melyekkel egyszerű az alkalmazás navigálása akár egy-oldalas, akár osztott módban.

Az alábbi 4 metódust definiálja az osztály:

- static Future<T?> **pushMain**(Widget page): egy adott komponensre (oldalra) navigáltatja az alkalmazást, minden esetben az egész képernyőt elfedve. Tehát osztott módban mind a bal, mind a jobb oldalt kitölti az új oldal.
- static Future<T?> **tryPushRight**(Widget page): osztott mód esetén a képernyő jobb oldalára helyezi az új komponens oldalt, egy-oldalas nézetben pedig a teljes képernyőre.

- static void **tryPopRight**(T? result): osztott mód esetén a képernyő jobb oldalán található oldalt próbálja meg levenni, egy-oldalas mód esetén pedig a jelenleg megjelenített legfelső oldalt. Emellett, hogyha egyetlen oldalt tartalmat a Navigátor, akkor azt nem vesszük le. A result paramétert pedig visszaadja a Navigátornak, mely push metódusok bevárása esetén kapható meg
- static Future<T?> **tryPushLeft**(Widget page): a **tryPushRight** metódus megvalósításra az osztott mód bal oldalára

Láthatóan ezek a metódusok nem fednek le minden szükséges esetet, nem lehet például a bal oldalról, sem osztott mód esetén a teljes képernyőn levenni oldalt. Ezekre a redundancia miatt nincsenek függvények, az alkalmazás jelenlegi állapotában nem lenne szükség ezek használatára.

3.8. Tesztelés

Az alkalmazás **Model**, **Persistence** és **Provider** rétegei rendelkeznek egységes tesztekkel. Ezeket a rétegeket lehetett ugyanis kiszámítható módon tesztelni. Ezek a tesztek az alkalmazás gyökér-mappájából, a 'test' mappán belül érhetőek el. Elindítható parancssorból a 'flutter test' parancssal.

A Nézetre és Szolgáltatásokra nincs külön egységes tesztelés, így az alábbi tesztelési kézikönyv szolgál ezeknek a rétegeknek a teszteléséért:

3.8.1. Szolgáltatások

BackgroundService		
Minden tesztet megelőz a configure() és registerHeadlessTask() meghívása		
Teszt	Elvárt eredmény	Kapott eredmény
start() függvény meghívásával az alkalmazás 1 nap elteltével lefuttatta a _backgroundRefresh() függvényt	Igaz	Igaz, bár az operációs rendszernek nem egy pontos időzítőjét használja az alkalmazsát, ezért inkább valamivel több, mint egy nap az eltelt idő.
stop() függvény meghívásával az alkalmazás nem küld hívja meg a háttérben a _backgroundRefresh() függvényt	Igaz, több nap után is.	Igaz, több nap után is.

NotificationService		
Minden tesztet megelőz a init() meghívása		
Teszt	Elvárt eredmény	Kapott eredmény
A show(int id, String title, String body, String? payload) függvény küld egy értesítést az eszközre title címmel és body tartalommal	Igaz	Igaz

YoutubeService		
Teszt	Elvárt eredmény	Kapott eredmény
A fetch(String id) metódus egy létező youtube játszási lista azonosítót kapva letölti és visszaadja annak a listának a jelenlegi legfrissebb állapotát	Igaz	Igaz, viszont lehet akár fél percnyi késés, míg a youtube valóban eltárol egy lista változtatást.

PopupService			
Teszt	Elvárt eredmény	Kapott eredmény	
A contextMenu (BuildContext, Offset, List) metódus Offset pozícióban List elemekkel megjelenít egy menüt.	Igaz	Igaz, ahogy az a Média típusú listaelemek hosszan nyomásával / jobb kattintásával is tesztelhető.	
A dialog (BuildContext, List, Widget) metódus megjelenít egy dialógus ablakot List-ben megadott gombokkal és Widget tartalommal	Igaz	Igaz	
A confirmDialog (BuildContext, String confirmText, String cancelText, Widget) hasonlóan a dialog (...) metódushoz megjelenít egy dialógusablakot, egy elfogadó és elutasító gombbal és Widget tartalommal.	Igaz	Igaz	
A showError (BuildContext, String) metódus megjelenít az alkalmazás alján egy felügrő értesítést a megadott String hibaüzenettel	Igaz	Igaz	

SharingService			
Előfeltétele	a	YoutubeService.fetch(...)	és
a PopupService.showError(...) helyes működése			
Teszt		Elvárt eredmény	Kapott eredmény
Youtube lejátszási lista linket megosztva az alkalmazás képes azt letölteni és lementeni az alkalmazásba		Igaz	Igaz
Helytelen megosztott link esetén (videó linkje, vagy teljesen random szöveg) az alkalmazás jelzi azt nekünk egy alul megjelenő hibaüzenettel		Igaz	Igaz

NavigatorService egy-oldalas mód			
Teszt		Elvárt eredmény	Kapott eredmény
A pushMain(Widget) metódus egy-oldalas nézet esetén egy új oldalt helyez az alkalmazásba		Igaz	Igaz
A tryPushRight(Widget) metódus egy-oldalas nézet esetén egy új oldalt helyez az alkalmazásba		Igaz	Igaz
A tryPushLeft(Widget) metódus egy-oldalas nézet esetén egy új oldalt helyez az alkalmazásba		Igaz	Igaz
A tryPopRight(Widget) metódus egy-oldalas nézet esetén kiveszi a legfelső oldalt, hogyha az nem az utolsó		Igaz	Igaz
A tryPopRight(Widget) metódus egy-oldalas nézet esetén nem veszi ki a legfelső oldalt, hogyha az az utolsó		Igaz	Igaz

NavigatorService osztott mód		
Teszt	Elvárt eredmény	Kapott eredmény
A pushMain (Widget) metódus osztott nézet esetén egy új oldalt helyez az alkalmazás teljes képernyőjére	Igaz	Igaz
A tryPushRight (Widget) metódus osztott nézet esetén egy új oldalt helyez az alkalmazás jobb oldalára	Igaz	Igaz
A tryPushLeft (Widget) metódus osztott nézet esetén egy új oldalt helyez az alkalmazás bal oldalára	Igaz	Igaz
A tryPopRight (Widget) metódus osztott nézet esetén kiveszi a legfelső jobb oldalt, hogyha az nem az utolsó	Igaz	Igaz
A tryPopRight (Widget) metódus osztott nézet esetén nem veszi ki a legfelső jobb oldalt, hogyha az az utolsó	Igaz	Igaz

3.8.2. Nézet

HomePage		
Teszt	Elvárt eredmény	Kapott eredmény
Ha nincsenek lementett lejátszási listák, akkor az oldalon egy segédszöveg jelenik meg.	Igaz	Igaz
Az oldalon az összes lementett lejátszási lista megjelenik.	Igaz	Igaz
Bármely lementett lejátszási listára nyomva az adott lista PlaylistPage oldalára navigálunk.	Igaz	Igaz
Hogyha van jelenleg letöltés alatt lévő lejátszási lista, akkor azt jelzi az alkalmazás.	Igaz	Igaz, az összes lejátszási lista után szereplő töltőkör segítségével.
A lejátszási lista elemeken látható a címük, a készítőjük, a videók száma és a jelenlegi állapotuk.	Igaz	Igaz
Az összes frissítése gomb megnyomására az összes lementett lejátszási lista frissíti magát.	Igaz	Igaz, viszont egyszerre 3 lista tud lekérni adatot.
A bal menü máshogyan néz ki Android és Windows platformon	Igaz	Igaz
A bal menüben elérhető preferenciák beállításra kerülnek valós időben.	Igaz	Igaz
A lejátszási listák átrendezhetőek a bal menüvel, és befejezés után véglegessé válik az új sorrend.	Igaz	Igaz
Az exportáló / importáló gombok működnek.	Igaz	Igaz
A bal menüből elérhető az AboutPage.	Igaz	Igaz

PlaylistPage		
Teszt	Elvárt eredmény	Kapott eredmény
A Videos fül a lista összes videóját tartalmazza az adott állapotban.	Igaz	Igaz
A Changes fül a lista összes változását tartalmazza, ha vannak.	Igaz	Igaz
A Changes fül a lista összes nem jó Anchor értékkel rendelkező videóját tartalmazza, hogyha nincs más változás.	Igaz	Igaz
A History fül tartalmazza az összes változtatást, ami a listával történt, időrendi sorrendben, legfelül a legújabbal.	Igaz	Igaz
Az oldal a Videos fülön nyílik meg, hogyha nincs változás.	Igaz	Igaz
Az oldal a Changes fülön nyílik meg, hogyha van változás.	Igaz	Igaz
A frissítés gombra nyomva csak az adott lista frissül.	Igaz	Igaz
A törlés gombra nyomva a lista törölhető.	Igaz	Igaz, beállítástól függően rákérdez az alkalmazás.
A Planned panel Androidon a Videos fülön alul érhető el, Windowson pedig a navigáló sávban található extra gombbal.	Igaz	Igaz

SearchPage		
Teszt	Elvárt eredmény	Kapott eredmény
A keresősávval kereshetőek lejátszási listák.	Igaz	Igaz
Valid lejátszási lista linket beírva a keresőmezőbe csak 1 lista jelenik, meg ami az ugyanaz mint a linkben szereplő.	Igaz	Igaz, ezzel adhatóak hozzá az alkalmazáshoz nemlistázott lejátszási listák.
Szöveget beírva a keresőbe az alkalmazás maximum 20 lejátszási listát megjelenít.	Igaz	Igaz
Ugyanazt a szöveget keresve ugyanazt az eredményt kapjuk.	Igaz	Hamis, mivel a Youtube maga randomizálja a keresési eredményeket.

3.9. Összegzés

Összegezve, az alkalmazás célja a Youtube lejátszási listák nehézkes kezelésének a megkönnyítése, a legnagyobb hangsúly a videókra fektetve. Mivel Youtube-on letörölhetőek a videók, így a lejátszási listáinkból is eltűnhetnek videók, viszont ezt nem tudjuk érzékelni csak manuálisan. Ezt automatizálja az alkalmazás, ellenőrizzetjük, hogy töröletk-e videót a listából, és lecserélhetjük azt egy másikra, hogyha lehetőség van rá.

3.9.1. További lehetőségek

Az alkalmazás a jelenlegi állapotában egy működőképes egészet alkot. Ezt azonban nem jelenti azt, hogy nincsenek további ötletek, funkciók, melyekkel bővíteni lehetne. Ezeket az ötleteket az alábbi felsorolás tartalmazza:

- Mivel az alkalmazás Material3 Design sémát követett, így Android platformra illik legjobban. Viszont lehetőség van a Windows Fluent UI nevezetű Design sémáját is implementálni, ennek implementálása a Windowsos verzióra lenne a terv.
- Jelenleg az alkalmazás csak Windows és Android platformon alérhető, mivel csak ezek az eszközök álltak a rendelkezésemre teszteléshez. A jövőben azonban szeretném több platformra is elérhetővé tenni az alkalmazást, mint iOS, macOS, Linux platformokra.
- Egy olyan funkciót is szeretnék implementálni, mellyel tagolni tudjuk majd a lejátszási listáinkat. Ez hasonló lenne, mint a PowerPointos szekciók, melyeket diákok közé lehet beszúrni.
- Egy olyan funkció is hasznos lenne, mellyel videókat az alkalmazáson belül lehetne lejátszani. Elsősorban csak a hangsávot, majd a teljes videót.

Irodalomjegyzék

- [1] *Material You: The next Stage in Material Design.* <https://material.io/blog/announcing-material-you>. Accessed: 2024-04-20.
- [2] *Declarative vs. Imperative Programming.* <https://codefresh.io/learn/infrastructure-as-code/declarative-vs-imperative-programming-4-key-differences/>. Accessed: 2024-04-27.
- [3] Gilad Bracha. *The Dart programming language.* Addison-Wesley Professional, 2015.
- [4] *Dart Documentation.* <https://dart.dev/guides>. Accessed: 2024-04-20.
- [5] *Flutter Documentation.* <https://docs.flutter.dev/>. Accessed: 2024-04-20.
- [6] *Provider package documentation.* <https://pub.dev/documentation/provider/latest/>. Accessed: 2024-04-23.
- [7] *The vm:entry-point pragma annotation.* https://github.com/dart-lang/sdk/blob/master/runtime/docs/compiler/aot/entry_point_pragma.md. Accessed: 2024-04-27.