ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra počítačů

Diplomová práce

Vývoj aplikace na chytré televize pro neziskovou TV

Bc. Viktor Sinelnikov

Leden 2022
Vedoucí práce: Ing. Tomáš Vondra, Ph.D.

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Sinelnikov** Jméno: **Viktor** Osobní číslo: **495794**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Vývoj aplikace na chytré televize pro neziskovou TV

Název diplomové práce anglicky:

Smart TV application development for a nonprofit TV

Pokyny pro vypracování:

CATVUSA je nezisková TV propagující ČR v USA. Nabízí tři internetové televizní kanály (cestopisný, jazykovědný a kuchařský) a dvě radiové stanice.
Ve spolupráci s producentem a technikem televizní stanice sepište požadavky na vzhled a funkčnost aplikace. Požadavky analyzujte a navrhněte jejich řešení pro chytré televize. Aplikace by měla splňovat doporučení pro tvorbu grafického rozhraní Android TV a použít API na hlasové ovládání. Bude se napojovat na Wordpressové API webu CATVUSA. Aplikaci implementujte. Napište jednotkové testy a funkční testy, které půjdou periodicky automatizovaně spouštět a odhalí i změny webu, které by rozbily aplikaci.
Funkčnost zahrne úvodní stránku s upoutávkami na aktuální pořady synchronizovanou s webem. Oznámí uživateli existenci nového obsahu. Hlavní funkcí bude streaming videa ze zmíněných televizních kanálů a rádio se zvýhodněním podporovatelů nadace. Veškerý obsah je již připraven v databázi webu.

Seznam doporučené literatury:

1. Android TV | Android Developers [online]. Google, 2020 [cit. 2020-12-14]. Dostupné z: https://developer.android.com/tv
2. Test apps on Android | Android Developers [online]. Google, 2020 [cit. 2020-12-14]. Dostupné z: https://developer.android.com/training/testing
3. DRDLÍČEK, Michael. Aplikace pro mobily a chytré televize pro neziskovou TV. 2017. Diplomová práce. České vysoké učení technické v Praze, Fakulta informačních technologií.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Tomáš Vondra, Ph.D., katedra počítačových systémů FIT

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **09.02.2021** Termín odevzdání diplomové práce: **04.01.2022**

Platnost zadání diplomové práce: **30.09.2022**

Ing. Tomáš Vondra, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Poděkování / Prohlášení

Chtěl bych poděkovat vedoucimu Ing. Tomáši Vondrovi, Ph.D. za vedení této práce a cenné rady poskytnute v průběhu tvorby diplomové práce. Dále bych chtěl poděkovat zadavateli Johnu Honneroví za možnost připojení k tomuto projektu, testování a zpětnou vazbu. Nakonec bych rád poděkoval své rodině za podporu během studia.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č.121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust.§2373 odst.2 zákona č.89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen "Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

Abstrakt / Abstract

Tato diplomová práce popisuje proces návrhu a implementace aplikaci pro Android TV. Popisuje nejmodernější architektonické přístupy ve světě Android a jejich implementaci za pomoci sady knihoven Android Jetpack a Android Leanback. Aplikace se vytváří pro project CATVUSA. CATVUSA je nezisková TV propagující ČR v USA. Nabízí tři internetové televizní kanály (cestopisný, jazykovědný a kuchařský) a dvě radiové stanice.

Klíčová slova: Android TV, CATVUSA, Wordpress, Android Jetpack

This thesis describes the process of design and implementation of the Android TV application. It describes the most modern architectural patterns in Android World, and their implementation using a suite of libraries Android Jetpack and Android Leanback. The application is made for the CATVUSA project. CATVUSA is a non-profit TV that propagates the Czech Republic in the USA. It offers 3 internet TV channels (travel, language, and cooking) and 2 radio stations.

Keywords: Android TV, CATVUSA, Wordpress, Android Jetpack

Title translation: Smart TV application development for a nonprofit TV

Obsah /

Úvod

Czech-American TV (CATVUSA) je nezisková organizace, která se zabývá propagací České republiky, její kultury a tradice v USA. Jednou z hlavních aktivit je vysílání pořadů o různých krajích a jejich tradicích na americké kabelové televizi. Zaměřuje se na komunitu etnických Čechů žijících v USA a jiné zájemce o českou kulturu. Každý týden tento pořad sleduje více než 2,5 milionu domácností v 60 amerických městech. Czech-American TV má webový portál catvusa.com, který poskytuje přístup k archivu těchto pořadů a dalších videí, například k lekcím češtiny a pořadům o vaření tradičních českých jídel. Kromě videa jsou k dispozici internetová rádia s českou hudbou, podcast o české kultuře, kvíz o českých krajích a řada článků na podobná témata.

Oblast chytrých televizí se rychle rozvíjí především v posledních deseti let. Celosvětovým trendem je přechod z kabelového vysílání do vysílání přes internet, takové vysílací služby dnes poskytují např. společnosti Netflix, HBO Go a jiné. I pořady CATVUSA patří mezi delší a divákům se lépe sledují na obrazovkách televize než na monitoru počítače, proto je důležité jim poskytnout možnost přístupu k obsahu portálu právě z velké obrazovky.

Tato diplomová práce se zabývá analýzou webového portálu CATVUSA, návrhem a implementací aplikací pro chytré televize řízené operačním systémem Android TV určené právě pro tuto neziskovou organizaci.

Práce je rozdělena do čtyř kapitol. V první kapitole je zformulován cíl práce a definovány funkční požadavky na aplikace. Ve druhé kapitole jsou uvedeny teoretické údaje o technologiích využitých v projektu. Začíná se popisem systému řízení obsahu Wordpress, na kterém se zakládá webový portál CATVUSA. Dále je krátce popsána architektura rozhraní REST a její základní metody, použité v aplikaci. Na konci druhé kapitoly jsou vysvětleny směrnice k podobě uživatelského rozhraní aplikací pro Android TV a vysvětleno, jak aplikace musí spolupracovat s operačním systémem a jaké požadavky musí být splněny, aby aplikace prošla kontrolou moderátorského týmu Google.

Ve třetí kapitole je provedena analýza zadavatele, datového modelu webového portálu, API webového portálu a je navrženo uživatelské rozhraní aplikací. Čtvrtá a poslední kapitola je věnována implementaci. Skládá se z popisu použitých nástrojů s příklady použití těchto nástrojů v aplikaci a popisu implementace jednotlivých obrazovek.

Výsledek práce umožní divácky mnohem komfortnější využití obsahu a představí CATVUSA na stránkách elektronického obchodu Google Play, čímž potenciálně zvýší počet diváků a podporovatelů CATVUSA.

Kapitola 1
Cíl

Cílem této diplomové práce je navrhnout a vytvořit aplikace určené pro chytrou televizi a řízené operačním systémem Android TV a modifikovat klientův systém správy obsahu (WordPress) pro snadnější práci s obsahem této aplikace. Aplikace bude vytvořena podle standardu a návrhových pokynů pro Android TV dostupných z [1]. Co se týče programové části, budou použity nejmodernější postupy a nástroje pro vývoj aplikací Android: sada knihoven Android Jetpack a knihovna Android Leanback.

1.1 Požadavky na aplikace

Ve spolupráci se zákazníkem byly sepsány následující požadavky:

1.1.1 Řízení ze systému pro správu obsahu

Na rozdíl od jiných aplikací pro portál CATVUSA (aplikace Android tvůrce Michaela Drdlicka [2] a aplikace iOS tvůrce Davida Lenskeho [3]), by obsah této aplikace měl být přímo řízen z administrátorského desk-systému pro správu obsahu Wordpress, takže při přidání nového videa by správce mohl zvolit, zda toto bude v aplikaci viditelné. Vztahuje se to nejen na jednotlivá videa, ale i na kategorii obsahu (takže když správce bude chtít uschovat všechna videa z kategorie „Recepty", musí takovou možnost vůbec mít).

1.1.2 Přehrávání videí

Aplikace musí přehrávat videa z portálu. Videa jsou rozdělena do kategorií, které mají být zobrazené odděleně na hlavní obrazovce. Některá z videí musí být zdůrazněna za účelem propagace, zdůraznění určuje správce systému. Před každým videem musí být přehráno speciální krátké video (intro) a po každém videu musí být přehráno další krátké video (outro). Intro a outro jsou nastaveny pro každé video zvlášť správcem systému při přidání nového videa. Je nutno zobrazit též detaily, textový doprovod k videu. Rovněž má být umožněno hledání podle názvu, uvnitř aplikace a při pomoci aplikace Google Assistant.

1.1.3 Přehrávání hudby

Webový portál má bohatou kolekci českých hudebních děl – klasických a lidových. Tyto kompozice jsou zařazené do playlistu a vysílají se ve formě rádia. Kromě hudebních dílů na třetím rádiu je vysílán podcast o české kultuře. Aplikace má umožnit přehrávání těchto playlistů.

1.1.4 Členství

Webový portál má program členství a finanční podpory projektu, který dává členům řadu výhod:

- Přístup k některým videím z omezené sekce.

- Možnost neomezeného poslouchání playlistu (nepřihlášení uživatelé mají k dispozici lhůtu deseti minut).

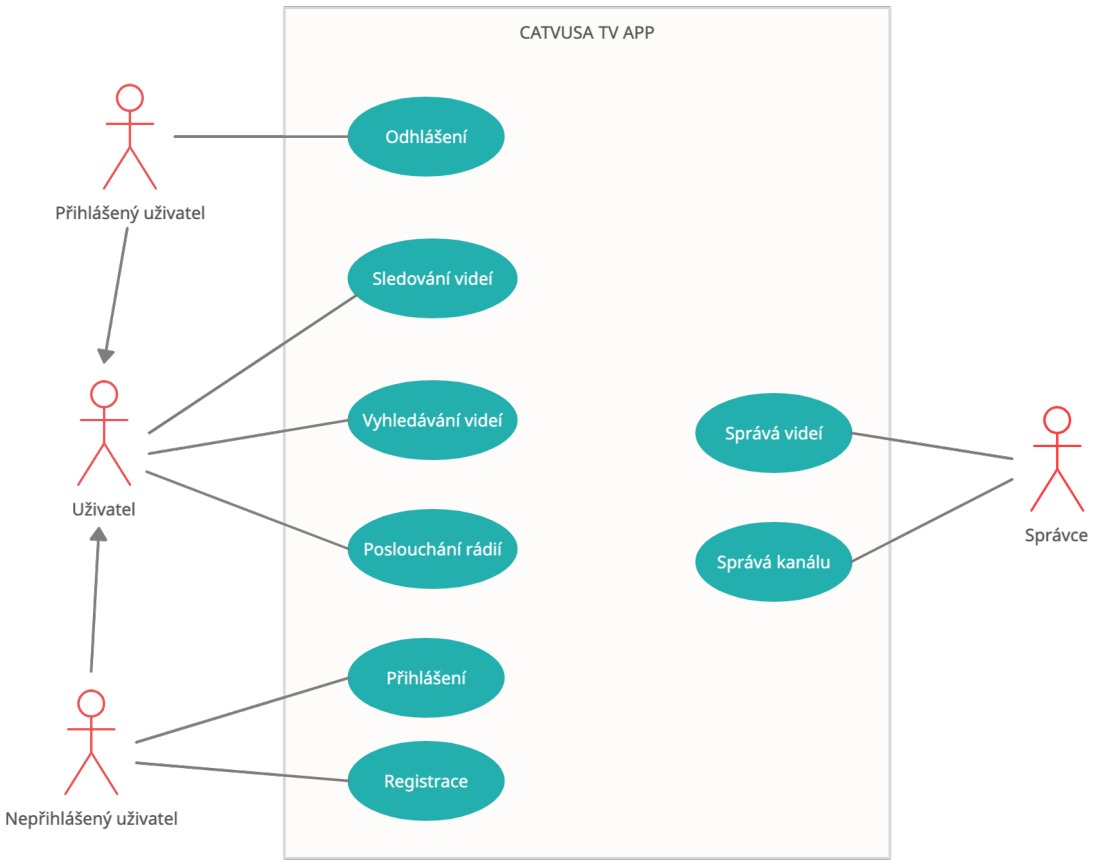
Aplikace má umožnit přihlášení a registraci nových členů, brát v úvahu roli uživatele, omezovat přístup nepřihlášených uživatelů k obsahu určenému jenom pro členy a omezovat dobu poslouchání hudby.

1.1.5 Hlasové ovladaní

Aplikace musí byt integrována v programu Google Assistant a podporovat následující funkce:

- Přehrání náhodného videa.
- Přehrání videa s názvem.
- Přehrání videa z kategorií.
- Přehrání posledního videa z kategorií.
- Zapnutí klasické hudby.
- Zapnutí lidové hudby.
- Zapnutí podcastu o české kultuře.
- Řízení přehrávání – spuštění a zastavení, převíjení vpřed a zpět.

Výše uvedené body mají fungovat, i když je aplikace uzavřena, takže metadata veškerého obsahu mají být dostupná pro Google Assistant ve správném formátu.



Obrázek 1.1. Use case diagram

Kapitola 2
Teorie

2.1 Wordpress

WordPress je bezplatný a otevřený systém pro správu obsahu; napsaný v PHP; databázový server – MySQL; vydáno pod licencí GNU GPL verze 2. Rozsah použití – od blogů po poměrně složité zpravodajské zdroje. Vestavěný systém „témat“ a „pluginů“ spolu s vhodnou architekturou umožňuje navrhovat projekty široké funkční složitosti. V rámci této práce hrají důležitou roli pojmy vlastního příspěvkového typu a vlastní taxonomie.

2.1.1 Post Types

WordPress obsahuje velké množství různých typů obsahu a to je rozděleno do tzv. Post Types (typy příspěvku). Jedna položka se nazývá post, ale toto je také název standardního Post Type nazývaného posts. Ve výchozím nastavení WordPress přichází s několika různými typy příspěvků, které jsou všechny uloženy v databázi pod tabulkou wp_posts. [4]

Výchozí typy příspěvků, které jsou vždy součástí instalace WordPressu, pokud není nastaveno jinak, jsou:

- Posts (příspěvky).
- Pages (stránky).
- Attachments (přílohy).
- Revisions (revize).
- Navigation Menus (navigační panely).
- Custom CSS (vlastní CSS).
- Changesets (sady změn).

Post

Post ve WordPressu je typ příspěvku, který je typický pro blogy a je též nejpoužívanější.

Page

Stránka je podobná příspěvkům, ale jsou zde některé velmi důležité rozdíly. Stránky se nezobrazují v obráceném časovém pořadí. Mohou být také umístěny do hierarchického pořadí, kde stránka může být rodičem nebo potomkem jiné stránky vytvářející strukturu stránky. Stránky také tradičně nevyužívají kategorie a tagy jako příspěvky.

Attachments

Přílohy jsou dalším typem příspěvků, které jsou zvláštní, protože obsahují informace o jakémkoli médiu, které je nahráno na web WordPress. Nejenže informace o hlavních příspěvcích jsou uloženy tam, kde jsou ostatní příspěvky, ale přílohy také využívají tabulku wp_postmeta k ukládání dalších informací, jako jsou metadata pro obrázky a videa.

Revisions

Revize jsou zvláštní speciální typ příspěvků, protože se používají k vytvoření historie jiných typů příspěvků v případě, kdy se uživatel bude chtít vrátit k předchozí verzi. Ačkoliv technicky revize nemohou být upraveny přímo, pokud nebudou vrácené, lze je upravovat stejně jako příspěvky a jsou uloženy v tabulce wp_posts jako jakýkoliv jiný typ příspěvku.

Menu

Menu ve WordPressu jsou seznamy odkazů, které lze použít k navigaci na webu. Umožňuje to vytvářet vlastní seznamy odkazů na různá místa na webu, které používají návštěvníci a které jsou upravovány v tematické části řídícího panelu, tedy mimo tradiční typy příspěvků, jako jsou příspěvky nebo stránky.

Custom CSS

Custom CSS je typ příspěvku specifický pro téma, které se používá k ukládání CSS.

Changesets

Sady změn jsou podobné revizím, ale speciálně pro Customizer (nástroj pro vytvoření a editace obsahu). Účelem je udržet Customizer v trvalém stavu. WordPress se pokusí zachovat změny obsahu provedené prostřednictvím Customizeru během uživatelské relace v příspěvku customize_changeset a pokusí se je obnovit, pokud aktuální relace bude ukončena.

2.1.2 Vlastní příspěvkové typy

Vlastní typ příspěvku (Custom Post Type) ve WordPressu je vlastní definovaná část obsahu. Použitím vlastních příspěvkových typů lze ve WordPressu definovat jakýkoliv typ obsahu a správce už není nucen používat pouze výchozí typy příspěvků uvedené v předchozí části. To otevírá dveře do nekonečného množství možností. [5]

2.1.3 Taxonomie

Taxonomie je definována jako způsob, jak seskupit podobné položky dohromady. Ve výchozím nastavení bude mít standardní příspěvek dva typy taxonomie, které se nazývají kategorie a tagy, což je praktický způsob, jak zajistit, aby související obsah na webu návštěvníci snadno našli. Tyto dva typy taxonomií jsou ve WordPressu ve výchozím nastavení zahrnuty, ale stejně jako jakoukoli jinou taxonomii je lze odstranit nebo změnit, anebo dokonce přidat další. Stejně jako příspěvky lze vytvořit i vlastní taxonomii. WordPress obsahuje schopnost automaticky zobrazit metabox s vlastní taxonomií na obrazovce pro přidání nebo úpravu příspěvku. [6]

2.2 API

Zkratka API (angl. Application Programming Interface) označuje univerzální rozhraní pro vzájemnou komunikaci (webových) aplikací. V praxi obsahuje API sbírku tříd, metod, funkcí a protokolů, díky kterým může jedna aplikace komunikovat s jinou, vzájemně synchronizovat uživatelská či systémová data. [7]

Ve webovém prostředí se pro komunikaci webového řešení se vzdáleným API používá standardní komunikační protokol HTTP/S, přes který dále obě strany většinou komunikují v platformě nezávislými formáty typu XML, CSV či JSON. Uvedené formáty posílají data ve standardizovaném zápisu a obě dvě strany jsou schopny tyto zápisy číst (parsovat) a vytvářet (generovat). [7]

2.3 REST

REST (Representational State Transfer) – je architektura rozhraní, navržená pro distribuované prostředí. Pojem REST byl představen v roce 2000 v disertační práci Roye Fieldinga, jednoho ze spoluautorů protokolu HTTP. Proto nepřekvapí, že REST má s HTTP hodně společného. [8]

REST je na rozdíl od známějších XML-RPC či SOAP, orientován datově, nikoli procedurálně. Webové služby definují vzdálené procedury a protokol pro jejich volání, REST určuje, jak se přistupuje k datům. [8]

Rozhraní REST je použitelné pro jednotný a snadný přístup ke zdrojům (resources). Zdrojem mohou být data, stejně jako stavy aplikace (pokud je lze popsat konkrétními daty). Všechny zdroje mají vlastní identifikátor URI a REST definuje čtyři základní metody pro přístup k nim. [8]

2.3.1 Metody pro přístup ke zdrojům

REST implementuje čtyři základní metody, které jsou známé pod označením CRUD, tedy vytvoření dat (Create), získání požadovaných dat (Retrieve), změnu (Update) a smazání (Delete). Tyto metody jsou implementovány pomocí odpovídajících metod HTTP protokolu. [8]

GET

Metoda GET se používá ke čtení reprezentace zdroje. V „šťastné“ (nebo nechybové) cestě vrací GET reprezentaci v XML nebo JSON a kód odpovědi HTTP 200 (OK). V případě chyby nejčastěji vrací 404 (NOT FOUND) nebo 400 (BAD REQUEST).

Podle návrhu specifikace HTTP slouží požadavky GET (spolu s HEAD) pouze ke čtení dat a nikoli k jejich změně. Proto, když jsou používány tímto způsobem, jsou považovány za bezpečné. To znamená, že je lze volat bez rizika modifikace nebo poškození dat – volání jednou má stejný účinek jako volání 10x nebo vůbec žádné. GET (a HEAD) je navíc idempotentní, což znamená, že vytvoření více identických požadavků má nakonec stejný výsledek jako jeden požadavek. [9]

Příklady:

- GET http://www.example.com/customers/12345
- GET http://www.example.com/customers/12345/orders
- GET http://www.example.com/buckets/sample

POST

Metoda POST se nejčastěji používá k vytváření nových zdrojů. Zejména se používá k vytváření nových prvků kolekce. Požadavek neobsahuje identifikátor nového prvku, tento identifikátor generuje služba a vrátí v odpovědi (ve hlavičce Location nebo v tělu).

Po úspěšném vytvoření vrátí stav HTTP 201 a hlavičku Location s odkazem na nově vytvořený prostředek.

POST není bezpečný ani idempotentní. Proto se doporučuje pro neidempotentní požadavky na zdroje. Provedení dvou identických požadavků POST s největší pravděpodobností povede ke dvěma zdrojům obsahujícím stejné informace. [9]

Příklady:

- POST http://www.example.com/customers
- POST http://www.example.com/customers/12345/orders

PUT

PUT se nejčastěji používá pro funkce „aktualizace“, volání PUT na URI známého zdroje s tělem požadavku obsahujícím nově aktualizovanou reprezentaci původního zdroje.

PUT však lze také použít k vytvoření prostředku v případě, že ID prostředku volí klient místo serveru. Jinými slovy, pokud je PUT na URI, které obsahuje hodnotu neexistujícího ID prostředku, tělo požadavku opět obsahuje reprezentaci zdroje.

Po úspěšné aktualizaci vrátí kód 200 (nebo 204, pokud nevrátí žádný obsah v těle). Pokud metoda PUT byla použita pro vytvoření, při úspěšném vytvoření vrátí stav HTTP 201. V případě vytvoření není nutné vracet odkaz prostřednictvím záhlaví Location, protože klient již nastavil ID zdroje.

PUT není bezpečná operace, protože upravuje (nebo vytváří) stav na serveru, ale je idempotentní. Jinými slovy, pokud vytvoříte nebo aktualizujete zdroj pomocí PUT a poté provedete stejné volání znovu, zdroj tam stále bude a bude mít stále stejný stav jako při prvním volání. [9]

Příklady:

- PUT http://www.example.com/customers/12345
- PUT http://www.example.com/customers/12345/orders/98765
- PUT http://www.example.com/buckets/secret_stuff

PATCH

PATCH se používá pro upravení zdrojů. Požadavek PATCH musí obsahovat pouze změny prostředku, nikoli celý prostředek.

Toto se podobá PUT, ale tělo obsahuje sadu instrukcí popisujících, jak by měl být zdroj aktuálně umístěný na serveru upraven, aby vytvořil novou verzi. To znamená, že tělo PATCH by nemělo být jen upravenou částí zdroje, ale mělo by mít v nějakém jazyku opravy, jako je JSON Patch nebo XML Patch.

PATCH není bezpečný ani idempotentní. Požadavek PATCH však může být vydán tak, aby byl idempotentní, což také pomáhá předcházet špatným výsledkům způsobeným kolizemi mezi dvěma požadavky PATCH na stejném zdroji v podobném časovém rámci.

Příklady:

- PATCH http://www.example.com/customers/12345
- PATCH http://www.example.com/customers/12345/orders/98765
- PATCH http://www.example.com/buckets/secret_stuff

DELETE

DELETE se používá ke **smazání** zdroje identifikovaného pomocí URI.

Po úspěšném odstranění vrátí stav HTTP 200 (OK) spolu s tělem odpovědi, nebo stav HTTP 204 (NO CONTENT) bez těla odpovědi.

Pokud jde o HTTP specifikace, operace DELETE jsou idempotentní. Opakované volání DELETE u tohoto zdroje skončí stejně: zdroj je odstraněn.

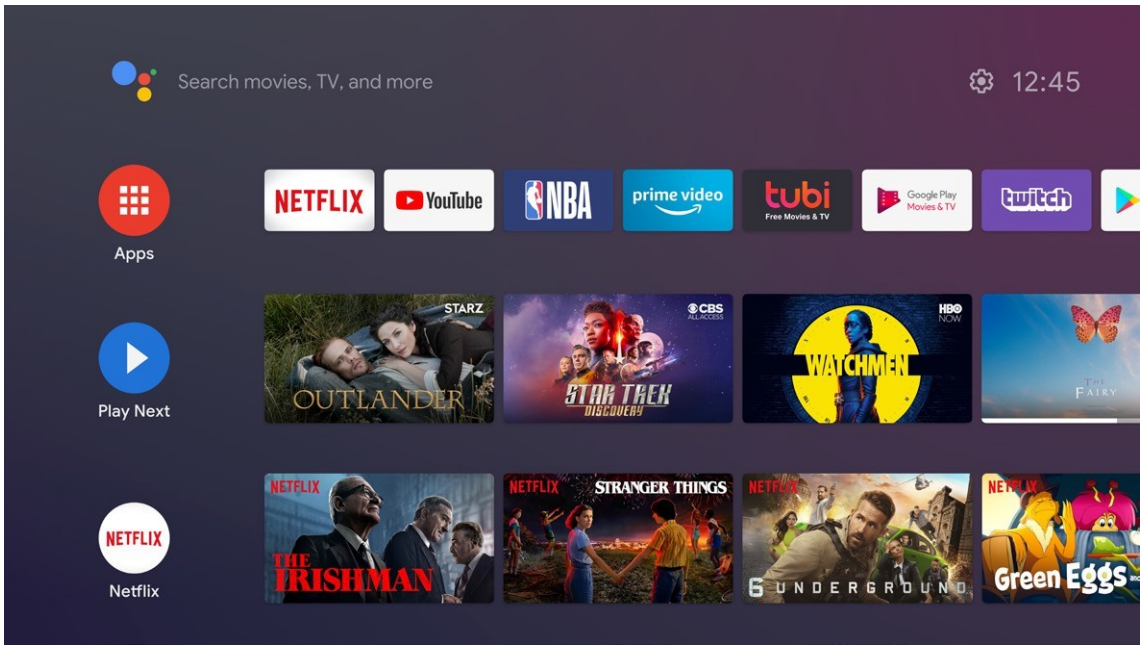
Existuje však varování ohledně idempotence DELETE. Volání DELETE na zdroji podruhé často vrátí 404 (NOT FOUND), protože zdroj již byl odstraněn, a proto jej již nelze najít. To podle některých názorů způsobuje, že operace DELETE již nejsou idempotentní, avšak konečný stav zdroje je stejný. Vrácení 404 je přijatelné a sděluje přesně stav zdroje.

2.4 Android TV

Android TV je operační systém chytré televize založený na Androidu a vyvinutý společností Google pro televize, přehrávače digitálních médií, set-top boxy a zvukové panely.

Z technického hlediska se neodlišuje od Androidu pro mobilní zařízení, ale má mnohem striktnější požadavky na design a správu vstupu od uživatele.

Velice doporučováno bývá nevyvíjet vlastní UI, ale použít hotové prvky z knihovny Android Leanback. Obsah aplikace měl by být dostupný nejen z aplikace, ale i z různých míst operačního systému – z hlavní obrazovky, přes globální vyhledávání, přes hlasového průvodce (Google Assistant, Amazon Alexa).



Obrázek 2.1. Hlavní obrazovka operačního systému Android TV

Kontrolní seznam od Google doporučuje vzít v úvahu následující body [10]:

2.4.1 Uživatelské rozhraní aplikace

Je třeba:

- Ujistit se, že text a ovládací prvky jsou dostatečně velké, aby byly viditelné z dálky.
- Poskytnout bitmapy a ikony ve vysokém rozlišení pro obrazovky HDTV.
- Ujistit se, že ikony a logo odpovídají specifikacím Android TV.
- Při přehrávání médií nebo ve hře se ujistit, že obrazovka zůstává zapnutá.
- Ujistit se, že každý prvek uživatelského rozhraní funguje jak s D-padem, tak s herními ovladači.
- Měnit obrázek na pozadí, když uživatelé procházejí obsah.
- Přizpůsobit si barvu pozadí tak, aby odpovídala značce ve fragmentech Leanback.
- Ujistit se, že uživatelské rozhraní nevyžaduje dotykovou obrazovku.
- Postupovat podle pokynů pro účinnou reklamu.
- Použít API knihovny Leanback s průvodcem, je-li potřeba provést uživatele řadou rozhodnutí.

2.4.2 Vyhledávání a odhalování obsahu

Je třeba:

- Poskytovat výsledky vyhledávání ze své aplikace v globálním vyhledávacím poli Android TV.
- Poskytovat pro vyhledávání datová pole specifická pro TV.
- Ujistit se, že aplikace zobrazuje nalezený obsah na obrazovce s podrobnostmi, které uživateli umožní začít obsah okamžitě sledovat.

2.4.3 Hlavní obrazovka Android OS

Musí být splněno, že:

- Každý kanál musí mít smysluplný název, který představuje obsah kanálu. Nedoporučuje se používat název aplikace jako název kanálu.
- Název kanálu se nesmí měnit, pokud nedojde k nějaké interakci s uživatelem.
- Každý kanál musí mít přiřazenou ikonu. Ikona nemusí být přesnou ikonou aplikace; může to být značková reprezentace obsahu v kanálu.
- Každý kanál musí být jedinečný a nesmí napodobovat funkce řádku Play Next. Například kanál, který uživatelům umožňuje pokračovat ve sledování videa tam, kde skončili, není pro kanál platným použitím.
- Každý program v kanálu musí mít jedno video. Program nesmí obsahovat sbírku videí.
- Program nesmí být propagačním sdělením nebo reklamou.
- Program musí mít vhodný popis a jeho metadata musí být správně mapována. Například hodnocení obsahu se nesmí objevit tam, kde se očekává název.
- Náhledové obrázky představující obsah nesmí být oříznuté ani roztažené. Musí odpovídat jednomu z dostupných poměrů stran.
- Program se musí začít přehrávat, jakmile jej uživatel vybere.

Kapitola 3
Analýza a navrh

3.1 Zadavatel

Czech-American TV (dále jen „CATVUSA“) vysílá na území Spojených států amerických jedinečný vzdělávací pořad o českých a moravských regionech a jejich kulturním dědictví již od roku 2003. Toto veřejné a nekomerční vysílání sleduje pravidelně každý týden na kabelové televizi více než 2,5 milionu domácností v šedesáti amerických městech. A díky internetovému streamu a vysílání v anglickém jazyce oslovuje další desítky tisíc diváků z různých koutů světa. [11]

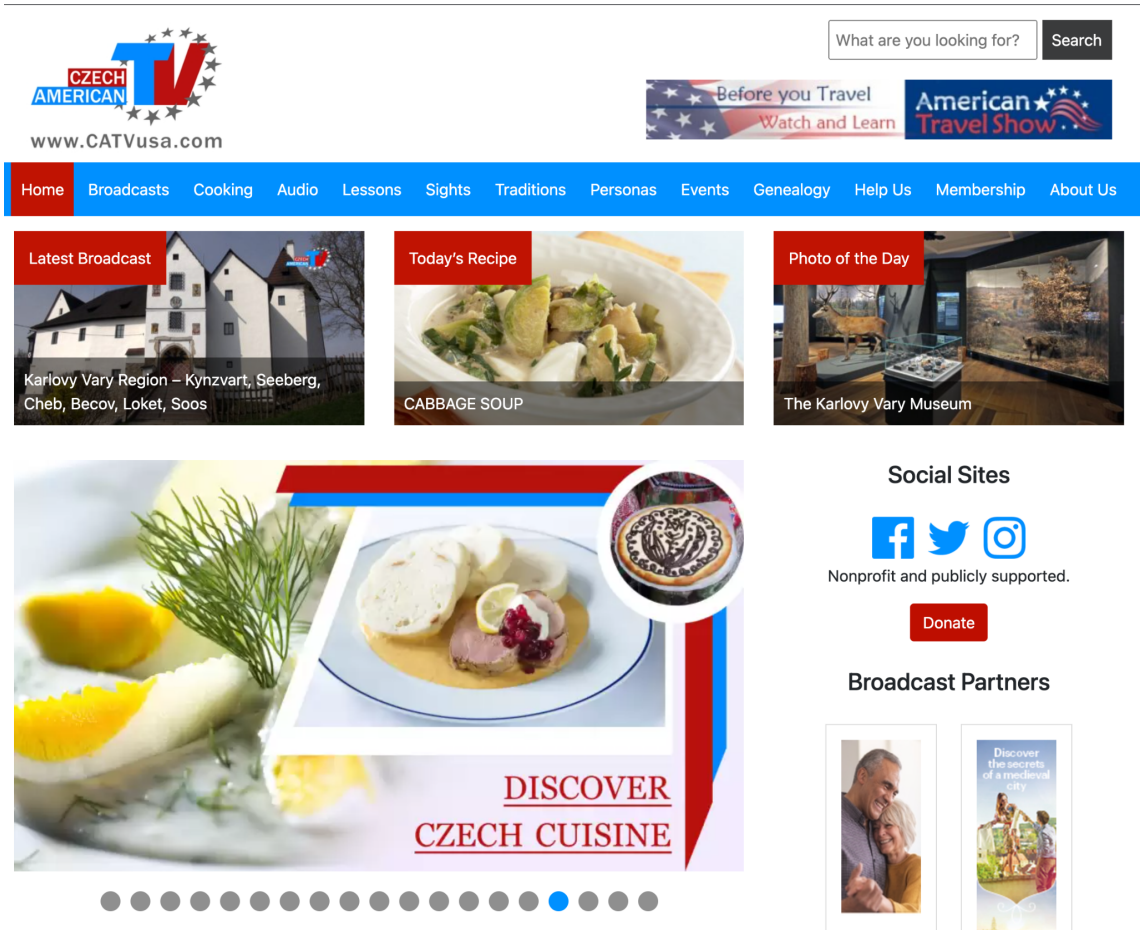
Obrázek 3.1. Logo CATVUSA

Cílem televizní stanice je rozšíření povědomí o České republice, jejích regionech, památkách a kultuře. Vysílání má formu videodokumentů, které jsou doprovázeny výkladem v anglickém jazyce. Kromě reportáží o zajímavých místech se televizní stanice věnuje také českým tradicím, folklóru nebo vaření typických českých pokrmů. [11]

Dnes se o produkci televizních pořadů stará více než čtyřicetičlenný tým dobrovolníků z řad mediálních profesionálů, studentů i pedagogů z USA i ČR. [11]

Vysílání vzniká především za pomoci jednotlivých krajů, které vítají příležitost zviditelnit svůj region v zahraničí. [11]

Kromě vysílání programů na kabelové televizi má televizní stanice i své webové stránky přístupné na adrese catvusa.com. Na těchto webových stránkách lze najít archiv videozáznamů programu kabelové televize, lekce češtiny, recepty české kuchyně, informace o českých tradicích, informace o významu českých příjmení, dvě internetová rádia s klasickou a lidovou hudbou a také podcast o české kultuře. [12]



Obrázek 3.2. Hlavní stránka webového portálu CATVUSA

Na vývoji a správě těchto webových stránek už více než deset let pracují studenti českých vysokých škol včetně ČVUT. Bylo vyvinuto několik aplikací pro Android a iOS, pluginy a šablony Wordpressu použité na portálu.

3.2 Datový model CATV USA

Portál CATVUSA používá následující strukturu obsahu: Videa na různá témata jsou shromážděna do různých typů příspěvků (tzv. custom post types) – Broadcasts, Cooking Videos, Lessons atd. Každý typ příspěvku má svoji taxonomii – Food Types, Tradition Types atd.

Správa videí se odlišuje od standardního přístupu ve Wordpressu – používá se plugin WPlyr. Místo nahrávání videa do média katalogu Wordpress se videa nahrávají do katalogu pluginu a integrují se do obsahu za vytvoření příspěvku. Každý příspěvek obsahuje typicky tři videa – intro (krátké představení CATVUSA a pořadů, které se vysílají na tomto kanálu), video s cílovým obsahem a outro.

Zadavatel uvádí, že seznam typu příspěvků se bude měnit, budou se přidávat a odebírat nové typy obsahu s videem a jen některé musí být v aplikaci zobrazeny. Rozhodl jsem vyřešit tento požadavek zavedením nové taxonomie – potřebné příspěvkové typy budou přiřazeny do této taxonomie a jejich seznam bude přístupný z API přes tuto taxonomii. Rádiové stanice se řídí pluginem „Radio Manager“, vyvinutém v rámci bakalářské práce studenta ČVUT Karla Vrabce. V rámci pluginu byl zaveden nový příspěvkový typ rm_musician a nová taxonomie rm_genre. Každý hudebník je přidán jako

příspěvek typu rm_musician do příslušného prvku taxonomie rm_genre. Audionahrávky se přidávají přes standardní katalog medií Wordpressu a ukládají se pod příslušným hudebníkem.

Uživatelé jsou řízeni přes plugin S2Member Pro. Tento plugin povoluje přiřadit každému uživateli úroveň přístupu (členství) a nastavovat požadovanou úroveň u příspěvku. Má širokou možnost automatizace, například na portálu realizované automatické nastavení úrovně členství, kdy uživatel podpoří organizaci prostřednictvím systému PayPal.

3.3 Práce s API

3.3.1 Wordpress REST API

Wordpress REST API Od verze Wordpress 4.7.0 obsahuje distribuce Wordpress REST API.

WordPress REST API poskytuje REST koncové body (URL) představující příspěvky, stránky, taxonomie a další vestavěné datové typy WordPressu.

V rámci této diplomové práce potřebujeme pracovat s taxonomií, typy příspěvků a s příspěvky pro stahování údajů o videích a audionahrávkách.

Zaprvé potřebujeme získat potřebné typy příspěvků. Jak je popsáno v předchozí části, vytvořil jsem taxonomii s názvem „android_tv“, kam bude správce systému přidávat potřebné příspěvkové typy. Seznam příspěvkových typů lze získat přes požadavek

GET https://www.testcatv.site/wp-json/wp/v2/types.

Tento požadavek vrátí JSON object, kde klíče jsou identifikátory (slugs) jednotlivých datových typů a hodnoty jsou JSON s údaji o příslušném příspěvkovém typu. Dále je uveden příklad takové struktury – příspěvkový typ broadcast.

```
"broadcast": {
    "description": "Travel show broadcasts from Czech towns",
    "hierarchical": false,
    "name": "Broadcasts",
    "slug": "broadcast",
    "taxonomies": [
        "post_tag",
        "android_tv"
    ],
    "rest_base": "broadcasts",
    "_links": {
        "collection": [
            {
            "href": "https://www.testcatv.site/wp-json/wp/v2/types"
            }
        ],
        "wp:items": [
            {
            "href": "https://www.testcatv.site/wp-json/wp/v2/broadcasts"
            }
        ],
        "curies": [
            {
                "name": "wp",
                "href": "https://api.w.org/{rel}",
                "templated": true
```

```
            }
        ]
    }
},
```

- description – uživatelsky příjemný popis příspěvkového typu. Bude využit jako popis kanálu na hlavní obrazovce.
- hierarchical – ukazuje, jestli tento příspěvek tohoto typu může mít dítě (nebo být dítětem) jiného příspěvku stejného typu. V rámci našeho úkolu to není důležité.
- name – jméno příspěvkového typu. Bude využito jako jméno kanálu na hlavní obrazovce a v aplikaci.
- taxonomies – seznam identifikátorů taxonomií navázaných na tento příspěvkový typ. Pro nás je důležité, aby obsahoval taxonomii „android-tv“.
- rest_base – RESTový koncový bod představující kolekce příspěvků tohoto typu.
- links – objekt s odkazy na související entity, používá se pro proces API Discovery. V rámci našeho úkolu není potřeba.

Bohužel tento koncový bod nepodporuje filtrování podle obsahování taxonomie android_tv v seznamu taxonomií, ale to není problém z důvodu malého počtu příspěvkových typů, které můžeme dostat jedním požadavkem a vyfiltrovat na straně klientů.

Za pomoci pole rest_base můžeme zformovat požadavek na seznam příspěvků příslušného typu (v tomto příkladu – broadcast). Bez parametru vypadá takto:

GET https://www.testcatv.site/wp-json/wp/v2/broadcasts

A vrátí následující datovou strukturu:

```
[
{
    "id": 1188764,
    "date": "2020-02-09T13:55:14",
    "date_gmt": "2020-02-09T18:55:14",
    "guid": {
"rendered": "https://www.catvusa.com/?post_type=broadcast&#038;p=1188764"
    },
    "modified": "2021-12-11T14:19:11",
    "modified_gmt": "2021-12-11T19:19:11",
    "slug": "liberec-region-jablonec-bohemian-paradise-castles-chateaus",
    "status": "publish",
    "type": "broadcast",
    "link": "https://www.testcatv.site/broadcast/liberec-region-...",
    "title": {
        "rendered": "Liberec Region &#8211; Jablonec, Bohemian..."
    },
    "content": {
        "rendered": "<div class=\"wplyr_container\">\n...",
        "protected": false
    },
    "excerpt": {
        "rendered": "<p>Discover Czech Regions: </p>\n...,
        "protected": false
    },
    "featured_media": 1188766,
    "template": "",
    "meta": {
```

```
        "_expiration-date-status": "",
        "_expiration-date": 0,
        "_expiration-date-type": "",
        "_expiration-date-categories": [],
        "_expiration-date-options": []
    },
    "tags": [],
    "android_tv": [
        151,
        139
    ],
    "acf": [],
    "_links": {
       …
    "wp:featuredmedia": [
    {
        "embeddable": true,
        "href": "https://www.testcatv.site/wp-json/wp/v2/media/1188766"
    }
    ],
       …
        }
    },
    { … },
    { … },
    { … },
]
```


Tato datová struktura je příliš rozsáhlá, obsahuje spoustu zbytečných položek a neobsahuje obrázek přidělený tomuto příspěvku (jenom jeho id). Avšak, jak vidíme v objektu _links.wp:featured_media, tento obrázek má flag „`embeddable`: true“, takže můžeme dostat podrobnější údaje o tomto obrázku za pomoci parametru požadavku _embed=wp:featuredmedia. Navíc můžeme požádat server o vrácení pouze potřebných polí za pomoci parametru požadavku _fields.

Filtrování podle taxonomie android_tv je možné za pomoci parametru požadavku android_tv=id, kde id je identifikátorem prvku taxonomie android_tv. Ale protože různá prostředí portálu (testovací/produkční) mohou mít různé identifikátory a identifikátory se mohou změnit po smazání a přidání prvku show, je lepší filtrovat podle slugu tohoto prvku. Wordpress REST API nepodporuje filtrování příspěvků podle slugu prvku taxonomie, ale umožňuje to plugin „WP REST API – Filter parameter for posts endpoints“ [13] – za pomoci parametru požadavku filter[taxonomy_id]=taxonomy_element_id.

Příspěvkový koncový bod vrátí data postupně za pomoci stránkování. Počet vrácených položek určuje argument požadavku per_page (výchozí hodnota – 10, maximální – 100). Počáteční prvek se určuje buď parametrem page (číslo stránky), nebo offset (počet překročených prvků). [14]

Kompletní URL se všemi požadavky bude vypadat následovně:
https://www.testcatv.site/wp-json/wp/v2/broadcasts?filter[android_tv]=show&_fields=id,title,content,
a vrátí následující datovou strukturu:


```
{
        "id": Int,
        "date_gmt": String,
        "modified_gmt": String,
```

```
        "title": {
            "rendered": String
        },
        "content": {
            "rendered": String,
            "protected": Bool
        },
        "excerpt": {
            "rendered": String,
            "protected": Bool
        },
       "android_tv": Array<Int>,
        "_embedded": {
            "wp:featuredmedia": [
                {
                    "id": Int,
                    "date": String,
                    "slug": String,
                    "type": String,
                    "link": String,
                    "title": {
                        "rendered": String
                    },
                    "author": Int,
                    "smush": { … },
                    "caption": {
                        "rendered": ""
                    },
                    "alt_text": "",
                    "media_type": String,
                    "mime_type": String,
                    "media_details": {
                        "width": Int,
                        "height": Int,
                        "file": String,
                        "sizes": {
                            "thumbnail": { … },
                            "medium": { … },
                            "medium_large": { … },
                            "large": { … },
                            "list": { … },
                            "big": { … },
                            "full": {
                                "file": String,
                                "width": Int,
                                "height": Int,
                                "mime_type": String,
                                "source_url": String
                            }
                        },
                        "image_meta": { … }
                    },
                    "source_url": String,
```

```
                }
            ]
        }
    }
```

- id – unikátní identifikátor příspěvku.
- date_gmt – datum vytvoření příspěvku ve formátu ISO 8601 (víc o formátu ve zdroji [15]).
- modified_gmt – datum poslední změny příspěvku ve formátu ISO 8601.
- title.rendered – název příspěvku.
- excerpt.rendered – krátký popis příspěvku ve formátu HTML.
- content.rendered – obsah příspěvku ve formátu HTML. Obsahuje blok videí playeru WPlyr, odkud můžeme dostat URL videí.
- android_tv – seznam identifikátorů prvků taxonomie android_tv, přidělených tomuto příspěvku.
- _embedded.wp:featuredmedia – metadata obrázku přiřazené tomuto příspěvku jako hlavní. Jako hlavní médium může být přiřazen jenom obrázek, ve kterém nás zajímají jenom objekty sizes uvnitř objektu media_details, které reprezentují instanci tohoto obrázku v různých velikostech.

Objekt size obsahuje následujici položky:

- file – jméno souboru.
- width – šířka v pixelech.
- height – výška v pixelech.
- mime_type – typ a podtyp obrázku. image/jpeg nebo image/png.
- source_url – přímý odkaz na obrázek.

Odpověď kromě těla obsahuje i hlavičky s údaji o stránkování:

- X-WP-Total: celkový počet příspěvků.
- X-WP-TotalPages: celkový počet stránek pro použitý parametr per_page.

Podobným způsobem lze dostat i seznam hudebníků (příspěvkový typ rm_musician). Katalog médií Wordpressu je zpřístupněn přes URI
GET https://www.testcatv.site/wp-json/wp/v2/media
S tímto koncovým bodem mohou být použity jak parametry už zmíněné _fields, page, per_page, tak i řada dalších – například media_type (pro filtrování podle typu médií, například image, audio nebo video) a parent (pro filtrování podle rodičovského příspěvku).

V aplikaci se tento koncový bod používá s následujícími parametry pro stahování seznamu audionahrávek rádiových stanic:
GET https://www.testcatv.site/wp-json/wp/v2/media?_fields=id,title,post,source_url&media_type=au
a vrátí seznam kompaktních datových struktur:

```
{
        "id": Int,
        "title": {
            "rendered": String
        },
        "post": Int,
        "source_url": String
}
```

id – unikátní identifikátor příspěvku. title.rendered – název médií. post – identifikátor rodičovského příspěvku (hudebníka). source_url – krátký popis příspěvku ve formátu HTML.

3.3.2 S2Member Pro API

Plugin S2Member Pro nabízí jednoduché API pro správu uživatele. [16] API je v experimentálním stavu a povoluje následující operace:

- auth_check_user (ověřuje platnost přihlašovacích údajů).
- get_user (vrátí data existujícího uživatele).
- create_user (vytváří uživatele, pokud neexistuje, nebo mění údaje).
- modify_user (mění údaje existujícího uživatele).
- delete_user (odstraňuje uživatele).

API je přístupné přes jediný koncový bod:
POST https://www.testcatv.site/?s2member_pro_remote_op=1

Tělo požadavku se předává ve formátu JSON jako pole x-www-form-urlencoded pod klíčem s2member_pro_remote_op. Tento JSON vždy musí obsahovat jméno operace (pod klíčem op) a API klíč (pod klíčem api_key). Vstupní data specifická pro operaci se předávají ve stejném JSON pod klíčem data.

V aplikaci budeme potřebovat 2 operace – auth_check_user a get_user.

Vstupní data pro auth_check_user mají následující formu:

```
{
"op": "auth_check_user",
"api_key": String,
"data": {
"user_login": String
"user_pass": String
}
}
```

V odpovědi se vrátí JSON objekt s jedním polem – „ID“ (identifikátor uživatele) v případě úspěchu, nebo „error“ (uživatelsky přijatelný popis chyby) v případě neúspěchu.

Vstupní data pro operaci get_user přijímají identifikátor uživatele a vrátí kompletní informaci o uživateli, jeho roli, času registrace atd.:

```
{
"op": "auth_check_user",
"api_key": String,
"data": {
"user_id": String
}
}
```

```
{
    "ID": 118,
    "role": "administrator",
    "level": 4,
    "ccaps": [],
    "data": {
        "ID": "118",
        "user_login": "sinelvik",
        "user_nicename": "sinelvik",
```

```
            "user_email": "sinelvik@fel.cvut.cz",
            "user_url": "",
            "user_registered": "2020-11-08 23:30:09",
            "user_activation_key": "",
            "user_status": "0",
            "display_name": "Viktor Sinelnikov"
        },
        "s2member_originating_blog": false,
        "s2member_subscr_gateway": "",
        "s2member_subscr_id": "",
        "s2member_custom": "",
        "s2member_registration_ip": "88.208.91.78",
        "s2member_notes": "",
        "s2member_auto_eot_time": "",
        "s2member_custom_fields": {
            "phone_number": ""
        },
        "s2member_paid_registration_times": {
            "level": 1604878209,
            "level4": 1604878209
        },
        "s2member_file_download_access_log": false
    }
```


Tento experimentální API neposkytuje způsob, jak zjistit požadovanou úroveň členství pro zobrazení určitých příspěvků, ale v PHP knihovně tohoto pluginu existuje metoda:

public array—bool is_post_protected_by_s2member (int post_id, bool check_user),

která vrátí nastavenou požadovanou úroveň podpory pro příspěvek [17]. S její pomocí jsem vytvořil koncový bod:

GET https://www.testcatv.site/is_post_protected_by_s2member.php,

který přijímá post_id jako parametr požadavku a vrátí požadovanou úroveň podpory, nebo „-1“, když je příspěvek přístupný pro všechny.

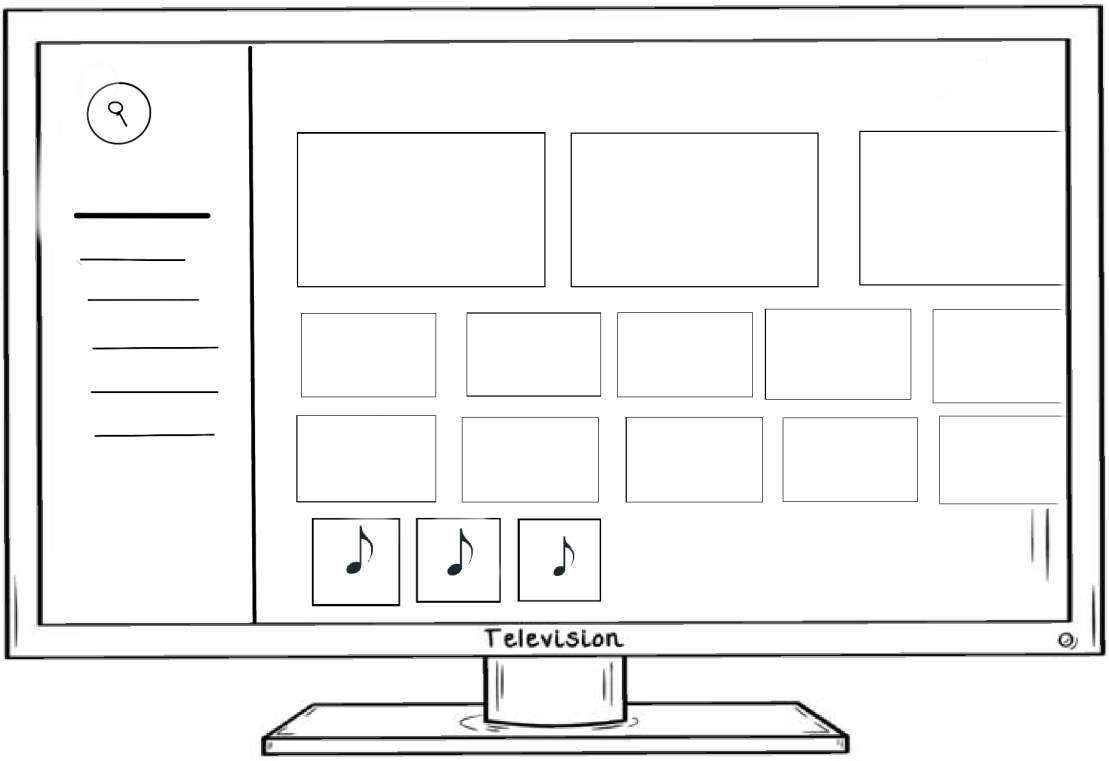
3.4 Návrh uživatelského rozhraní

3.4.1 Hlavní obrazovka

Hlavní obrazovka bude obsahovat následující prvky – vyjíždějící navigační menu se seznamem řádků, tlačítko vyhledávání a hlavní obsah po pravé straně. Hlavní obsah jsou následující řádky prvku (napojené na příslušné řádky navigačního menu):

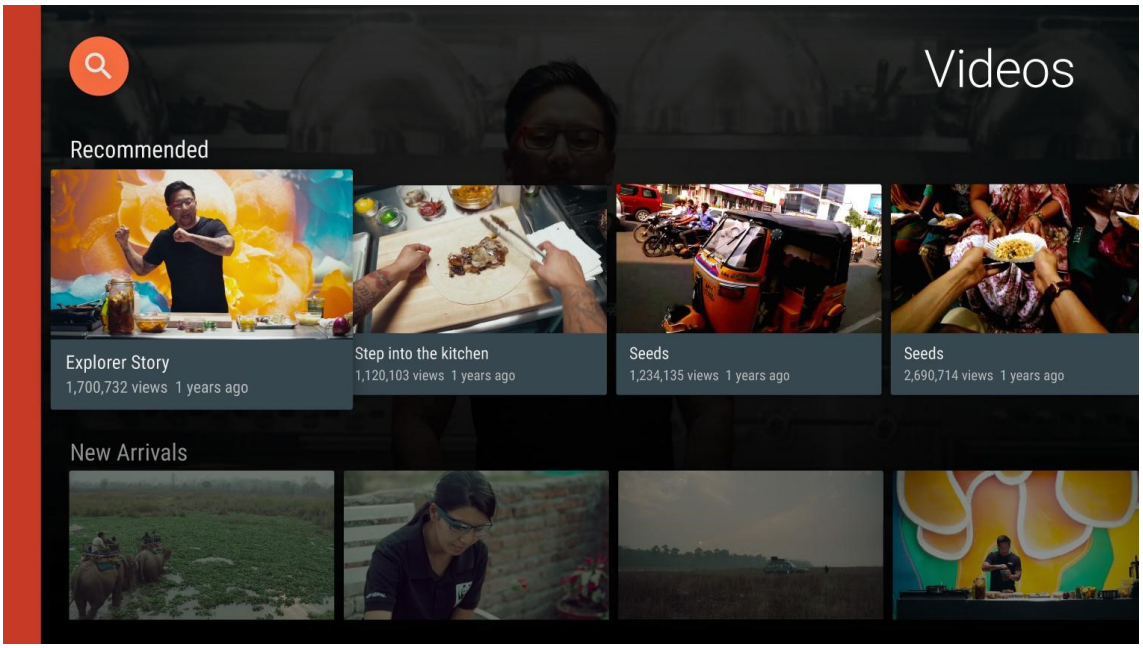
- Seznam doporučených videí (UI prvky v tomto řádku mají dvakrát větší velikost než ostatní).
- Několik řádků odpovídajících jednotlivým kanálům (příspěvkovým typům) základní velikosti.
- Seznam rádiových stanic.
- Seznam prvku nastavení – obsahuje jenom čtverec se stavem přihlášení uživatele.

Schematicky by hlavní obrazovka měla vypadat následujícím způsobem:



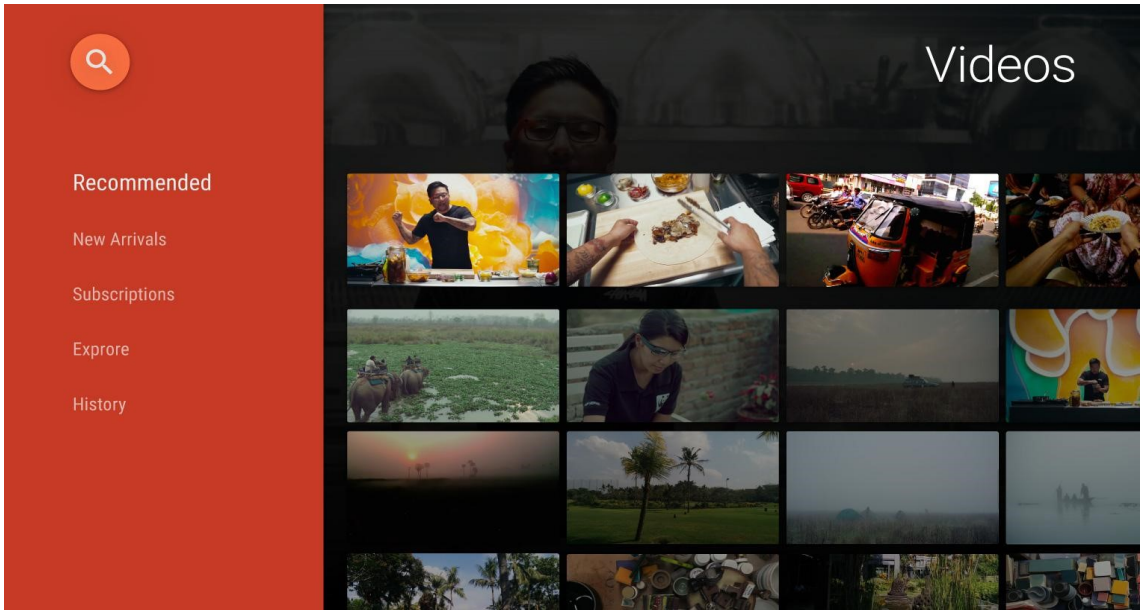
Obrázek 3.3. Návrh uživatelského rozhraní hlavní obrazovky

Základ tohoto návrhu byl převzat z obrazovky „Browse View“ oficiální návrhové příručky Android TV. [18] Podle uvedené příručky chování tohoto rozhraní musí vypadat následujícím způsobem:



Obrázek 3.4. Hlavní obrazkovka z oficialní přiručky

Browse Row organizuje obsah do kategorií. Tyto kategorie se zobrazují ve formě vodorovných seznamů, které jsou poskládány svisle na sebe. Ve výchozím nastavení se vodorovně posouvá vždy pouze jeden řádek. [18]



Obrázek 3.5. Browse Lane

Browse Lane je vertikální seznam všech položek v řádcích procházení. Položky se mapují přímo na názvy kategorií Browse Row. [18]

Browse Lane se vysune zleva a při vstupu vytlačí část obsahu z obrazovky. Procházením Browse Lane se současně posouvají odpovídající řádky Browse Row. [18]

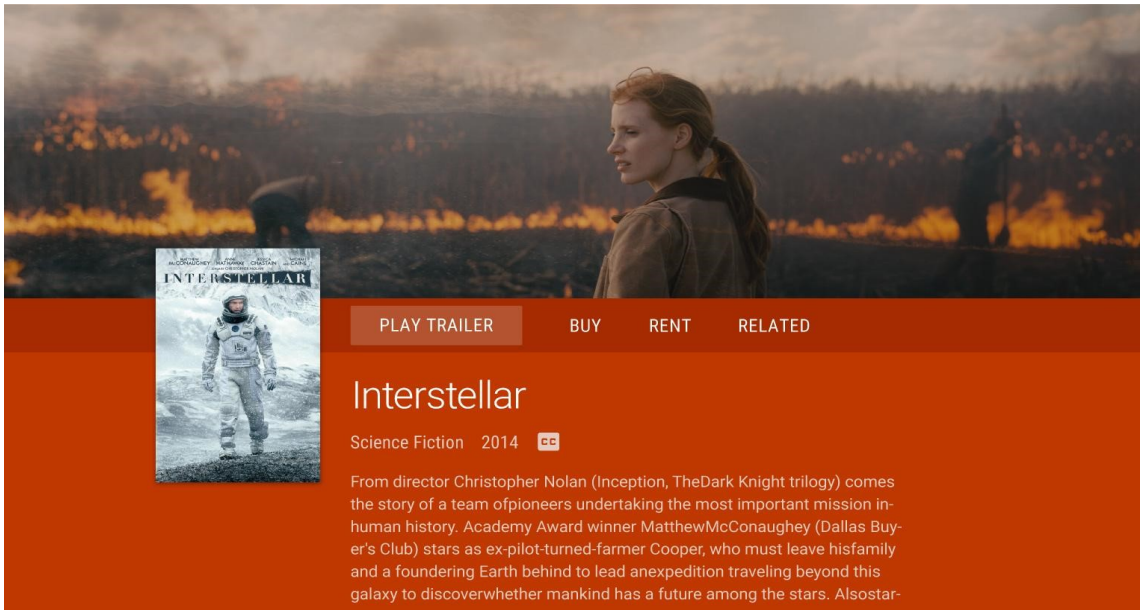
Browse Lane se doporučuje, pokud obsah má víc než pět řádků. [18]

3.4.2 Obrazovka detailů videa

Obrazovka detailů videa obsahuje

- Název.
- Popis videa.
- Seznam souvisejících videí (několik videí ze stejného kanálu).
- Tlačítko ke spuštění videa.

Tato obrazovka bude zcela odpovídat verzi z oficiální návrhové příručky:

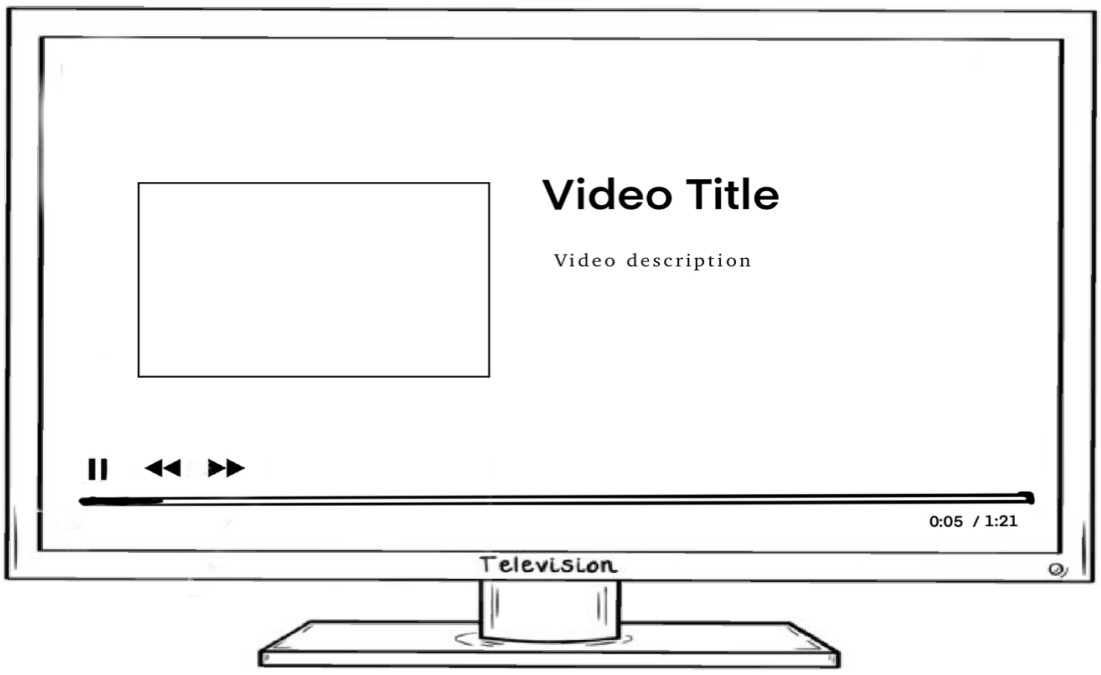


Obrázek 3.6. Obrazovka detailu videa z oficiální návrhové příručky

3.4.3 Obrazovka přehrávače videa

Obrazovka přehrávače bude obsahovat:

- Název videa.
- Krátký popis.
- Hlavní obrázek videa.
- Nástroje na kontrolu přehrávání – play/pause, rychlé převíjení.
- Časovou osu.



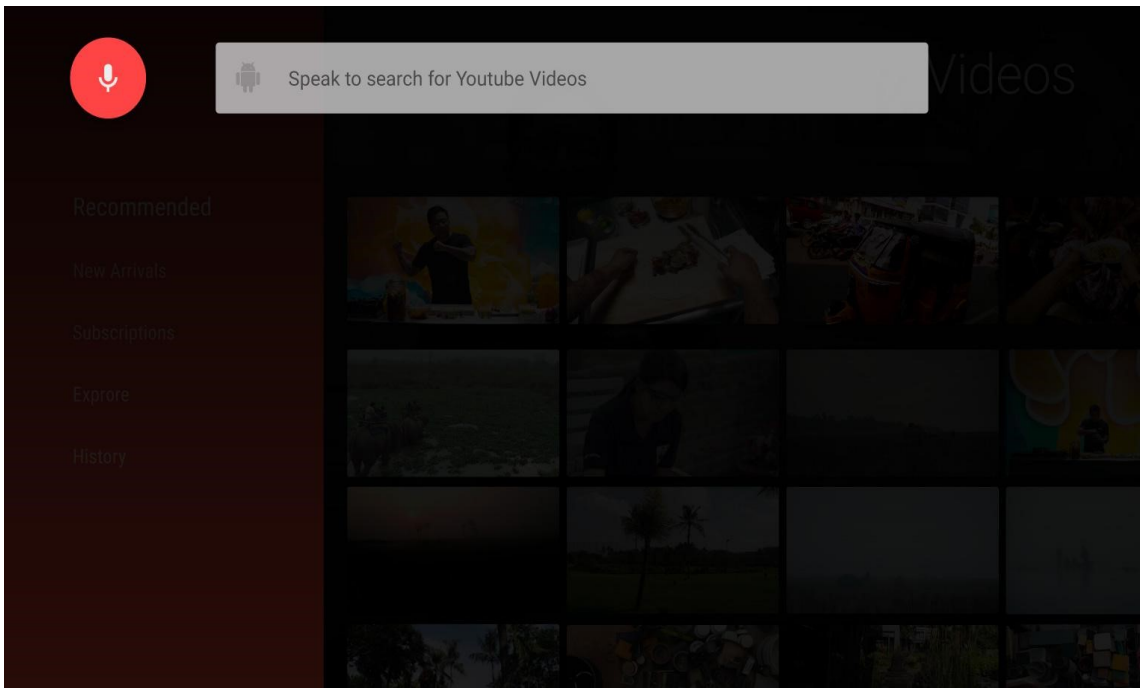
Obrázek 3.7. Návrh obrazovky přehrávače videa

3.4.4 Obrazovka radiové stanice

Úkolem této obrazovky je zobrazovat informace o přehrávané skladbě a poskytovat možnost přerušit přehrávání. Uživatelské rozhraní je stejné jako v přehrávači videí, ale na ploše videa bude zobrazen obrázek spojený s přehrávanou skladbou.

3.4.5 Obrazovka vyhledávání videa

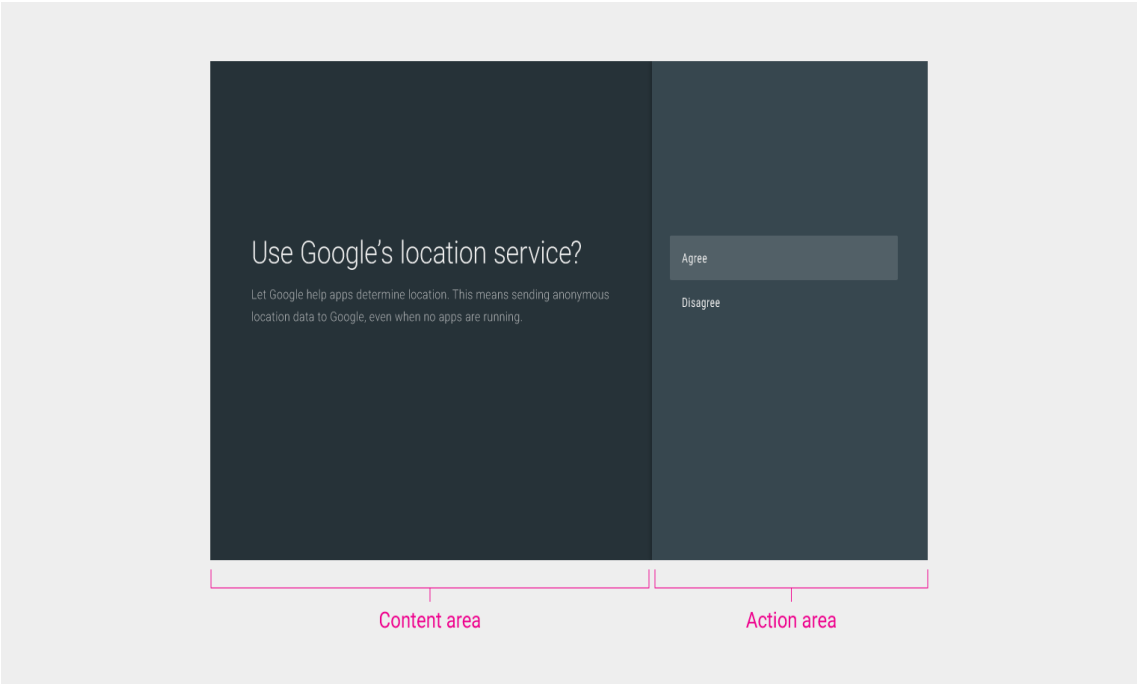
Obrazovka vyhledávání bude zcela odpovídat verzi z oficiální návrhové příručky [19] a obsahovat jenom vstupní pole pro vyhledávání slova, tlačítko pro hlasový vstup a seznam nalezených videí.



Obrázek 3.8. Obrazovka vyhledávání z oficiální návrhové příručky

3.4.6 Obrazovka přihlašování

Obrazovka přihlašování bude postavena podle vzoru „Dialog“. [20] V akční zóně bude obsahovat editovatelné pole pro uživatelské jméno a heslo.



Obrázek 3.9. Příklad vzoru „Dialog“

Kapitola 4
Implementace

Nativní aplikace pro Android OS se píší buď v jazyce Java, nebo v jazyce Kotlin. Od roku 2019 je doporučovaným jazykem Kotlin. Oficiálním vývojovým prostředím podporovaným autory tohoto systému a jediným podporujícím pokročilé ladění aplikace pro tento operační systém je Android Studio. Pro vývoj této aplikace jsem použil vývojové prostředí Android Studio a jazyk Kotlin.

4.1 Základní prvky aplikaci

4.1.1 Gradle

Pro sestavování aplikací pro Android se používá se systém Gradle. Gradle usnadňuje začlenění externích binárních souborů nebo jiných modulů/knihoven do sestavení formou závislostí. Závislosti mohou být umístěny na lokálním počítači nebo ve vzdáleném úložišti. [21]

V konfiguračním souboru build.gradle se zadávají důležité údaje jako identifikátor aplikace, číslo a jméno verze, minimální požadovaná verze OS Android a verze nástrojů pro sestavování. [22]

```
android {
    compileSdk 31

    defaultConfig {
        applicationId "com.sinelnikov.catvusa"
        minSdk 21
        targetSdk 31
        versionCode 1
        versionName "1.0"
    }
}
```

Gradle povoluje definovat různé varianty aplikací v jednom projektu. Toho dosáhneme za pomoci tzv. build types a product flavors. [23]

Build types se obyčejně používají k rozlišení různých stavů aplikace (například debug, test, release) a obyčejně se odlišují úrovní logování a použitým sběračem logu, použitím minimizátoru a obfuskátoru kódu a dalšími technickými záležitostmi.

Product flavors mají obyčejně obchodní význam, mohou představovat aplikace pro různé trhy, placenou a neplacenou verzi atd.

Typů rozlišení může být značné množství, proto product flavors mohou mít několik rozměrů.

Pro každý product flavors lze definovat vlastní hodnoty proměnných a používat je v kódu prostřednictvím objektu BuildConfig.

Pro tuto aplikaci jsem zavedl dimenzi „api“, která odpovídá použitému prostředí, a dva product flavors: stage (odpovídá testovacímu prostředí www.testcatv.site) a prod

(odpovídá produkčnímu – ostrému prostředí www.catvusa.com). Uvnitř jsou definovány odkazy na stránku registrace nových uživatelů, bázové URL a klíče pro API.

```
flavorDimensions "api"

productFlavors {
    stage {
        dimension "api"
        buildConfigField("String", "SIGNUP_URL",
            "\"http://www.testcatv.site/member-information/\""
        )
        buildConfigField("String", "WP_API_BASE_URL",
            "\"http://www.testcatv.site/wp-json/wp/v2/\""
        )
        buildConfigField("String", "S2_MEMBER_API_BASE_URL",
            "\"http://www.testcatv.site/\""
        )
        buildConfigField("String", "S2_MEMBER_API_KEY", "----")
    }
    prod {
        dimension "api"
        buildConfigField("String", "SIGNUP_URL",
            "\"http://www.catvusa.com/member-information/\""
        )
        buildConfigField("String", "WP_API_BASE_URL",
            "\"http://www.catvusa.com/wp-json/wp/v2/\""
        )
        buildConfigField("String", "S2_MEMBER_API_BASE_URL",
            "\"http://www.catvusa.com/\""
        )
        buildConfigField("String", "S2_MEMBER_API_KEY", "----")
    }
}
```

4.1.2 Knihovná Leanback

Knihovna Leanback je sada vysokoúrovňových widgetů, které odpovídají specifikacím Material Design a poskytují skvělou uživatelskou zkušenost v případě, že si lidé prohlížejí aplikaci na televizi. [24]

Knihovna Leanback je založena na vzoru zobrazení modelu MVP a mnoho z jejích widgetů umožňuje poskytovat vlastní prezentér. V MVP prezentér funguje jako spojení mezi modelem a zobrazením, ale je také zodpovědný za zvětšení zobrazení. Toto chování znamená, že poskytnutí vlastního prezentéru umožňuje úplnou kontrolu nad vzhledem widgetu. [24] Vzor MVP však se používá pouze na úrovni jednotlivých widgetů, pro architekturu aplikace lze použít jakýkoliv vzor.

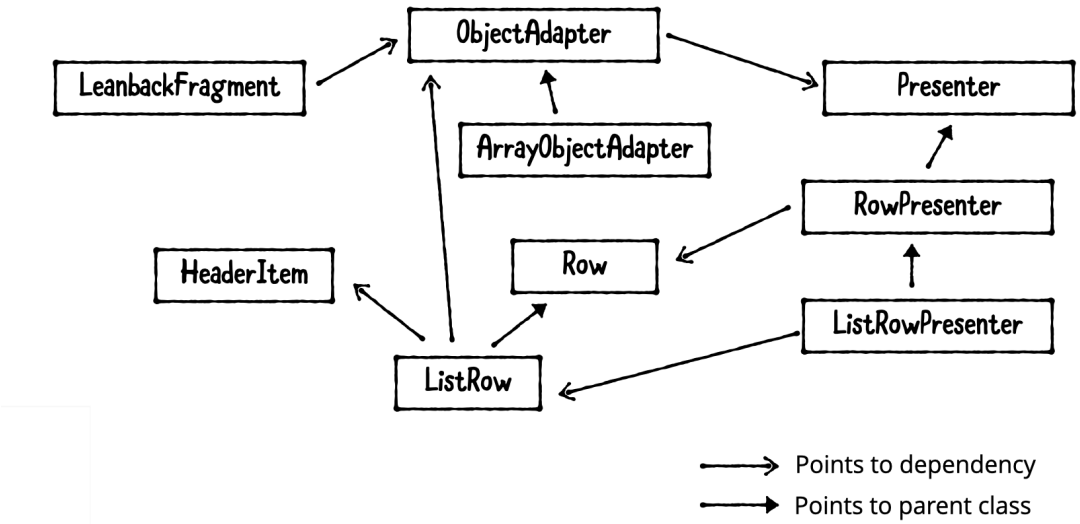
Pro implementaci vlastního prezentéru je potřeba předdefinovat následující funkce [25]:

- onCreateViewHolder, kde se vytváří zobrazení ze souboru náhledů.
- onBindViewHolder, kde se nastavuje obsah z předaného objektu.
- onUnbindViewHolder, kde se uvolňují zdroje.

Za reprezentace seznamů odpovídá adaptér. Přijímá seznam objektu s daty a předává je příslušným prezentérům.

ObjectAdapter je základní třída, která se používá ve knihovně Leanback. Poskytuje přístup k datovému modelu a je oddělena od prezentace položek prostřednictvím PresenterSelector. [26]

PresenterSelector mapuje třídy datového modelu k příslušným prezentérům, takovým způsobem jeden adaptér může podporovat a reprezentovat několik typů prvku.



Obrázek 4.1. Schéma souvislosti tříd knihovny Leanback

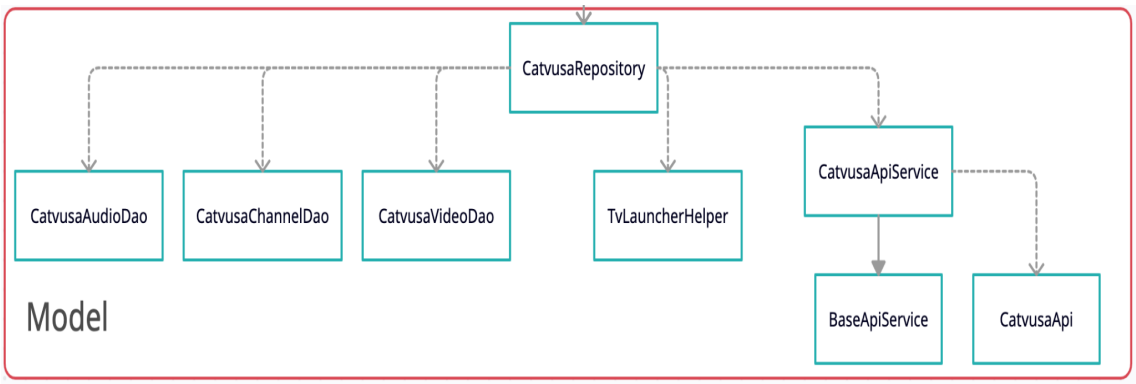
4.1.3 Datová vrstva

Architektura této aplikace se zakládá na rozdělení vrstev, použití návrhových vzorů a zapouzdření.

Vrcholem datové vrstvy jsou třídy implementující návrhový vzor Repository. Třídy ostatních vrstev komunikují jen s repozitářem a nevědí nic o vnitřním složení.

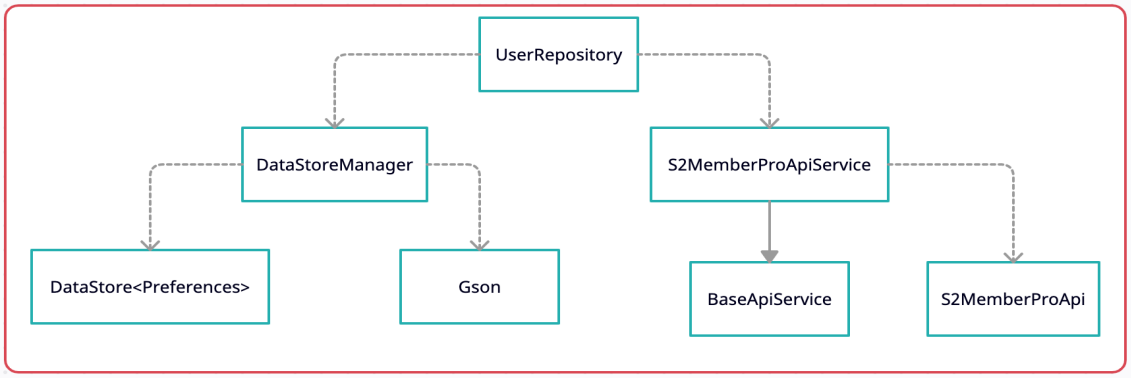
Každý repozitář odpovídá za svoji doménu. Takové domény jsou dvě: obsah webového portálu a uživatel v systému S2Member.

První repozitář komunikuje s databází přes rozhraní DAO (Data Access Object), komunikuje se serverem přes třídu CatvusaApiService a má přístup do hlavní obrazovky operačního systému přes pomocnou třídu TvLauncherHelper. Všechny závislosti se předávají do konstruktoru automaticky za pomoci frameworku pro vkládání závislosti.



Obrázek 4.2. Diagram závislosti CatvusaRepository

Repozitář UserRepository také komunikuje se serverem přes třídu ApiService, ale pro perzistenci dat používá DataStore – doporučené řešení na ukládání par klíč – hodnota ze sady knihoven Android Jetpack. V tomto úložišti se chrání informace o uživateli ve formě JSONového řádku. Serializace a deserializace se provádí ručně za pomoci knihovny GSON ve vlastní třídě DataStoreManager.



Obrázek 4.3. Diagram závislosti UserRepository

4.1.4 Komunikace se serverem

Pro komunikaci se serverem jsem použil knihovnu Retrofit. Tato knihovna generuje kód podle poskytnutého rozhraní, kde programátor specifikuje jenom metodu HTTP, relativní adresu koncového bodu, seznam parametrů požadavku a vrácenou datovou strukturu. Pro tuto knihovnu bylo vytvořeno značné množství nástaveb a integrace s jinými knihovnami pro jednodušší implementaci síťové vrstvy a snižování podílu šablonného kódu (tzv. boilerplate).

Jednotlivá metoda v tomto rozhraní vypadá takto:

```
@GET("rm_musician?_fields=id,title,rm_genre,_links,_embedded
    &_embed=wp:featuredmedia"
)
    suspend fun getMusicians(
        @Query("rm_genre[]") genres: List<Int>,
        @Query("page") page: Int,
        @Query("per_page") pageSize: Int,
    ): Response<List<WpRmMusician>>
```

V anotaci funkce specifikujeme HTTP metodu, RESTové jméno zdroje a (nepovinně) seznam konstantních parametrů požadavku. K parametrům metody lze také připojit sadu anotací, například @Header - hlavička požadavku @Query - parameter požadavku @Body - tělo požadavku a jiné.

Lze modifikovat i vrácenou hodnotu metody (například použít třídu Observable z konceptu Reactive Extensions), ale rozhodl jsem použít standardní třídu Response (která obsahuje všechny komponenty HTTP odpovědi – stavový kód, informační správu stavu, hlavičky a tělo odpovědi – a převést tyto komponenty na vlastní datové struktury (třídy Result, PagedResponse) ručně. Hlavním důvodem je to, že API vrátí informace o počtu stránek v hlavičkách místo těla odpovědi. Jediné, co je provedeno automaticky, je parsování JSON za pomoci knihovny GSON a její rozšíření pro Retrofit GsonConverterFactory.

Další vrstvou ve zpracování dat ze serveru je ApiService (CatvusaApiService.kt / S2MemberProApiService.kt). Odpovídá za správné zpracování požadavku, interpretaci chyb, práci s požadavky podporujícími stránkování.

Vytvořil jsem třídu Result, která představuje úspěšné nebo neúspěšné dokončení a povoluje řízení ve funkcionálním stylu (zprávu všech možných stavů):

```
sealed class Result<out T: Any> {
    data class Success<out T: Any>(val data: T) : Result<T>()
    data class Error(val error: Exception) : Result<Nothing>()
}
```

```
val musicians = when (
    val musiciansResult = apiService.getMusicians(genreIds)
) {
    is Result.Error -> throw musiciansResult.error
    is Result.Success -> musiciansResult.data
}
```

Stránkování se řeší za pomoci vytvořené třídy PagedResponse a následující metody, která parsuje hlavičky HTTP odpovědi.

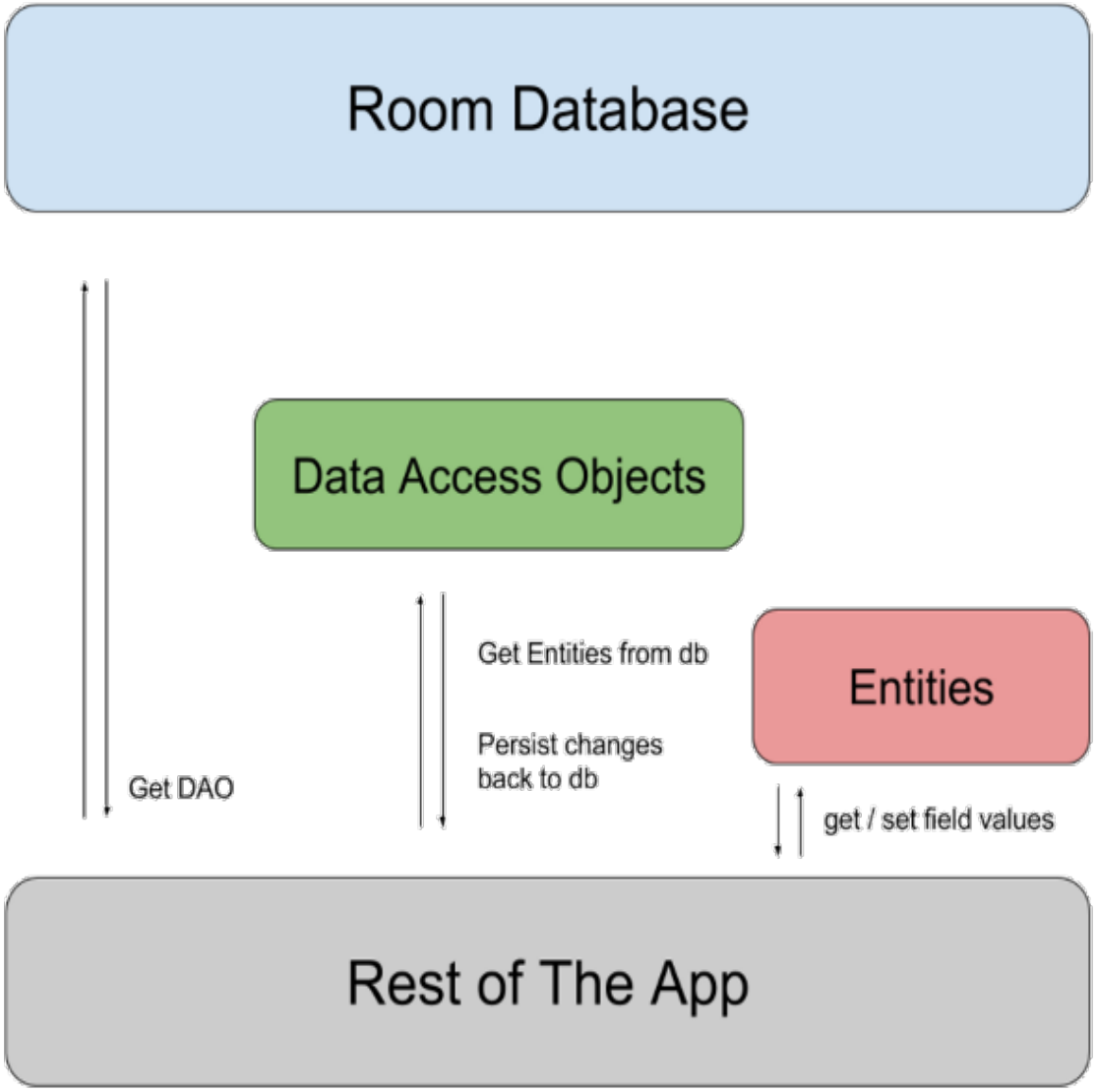
```
private suspend fun <T> getPagedResponse(
        page: Int,
        apiCall: suspend () -> Response<List<T>>
    ): Result<PagedResponse<T>> {
    val response: Response<List<T>>
    try {
        response = apiCall.invoke()
    } catch (t: Throwable) {
        return Result.Error(mapToNetworkError(t))
    }

    val body = response.body()

    if (response.isSuccessful) {
        return if (body != null) {
        val pagedResponse = PagedResponse(
            page = page,
            totalPages =
                response.headers().get("X-WP-TotalPages")?.toInt()!!,
            totalResults =
                response.headers().get("X-WP-Total")?.toInt()!!,
            results = body
        )
        Result.Success(pagedResponse)
        } else {
            Result.Error(HttpException(response))
        }
    } else {
        return Result.Error(mapApiException(response.code()))
    }
}
```

4.1.5 Práce s databází

Pro práci s databází jsem použil knihovnu Room ze sady knihoven Android Jetpack. Tato knihovna výrazně zkracuje počet kódů, stará se o automatické vytvoření databáze a tabulek a o přístup k souboru databází.



Obrázek 4.4. Architektura knihovny Room

Specifikace tabulky se provádí ve třídě odpovídající řádku této tabulky, stačí jen přiřadit anotace @Entity cílové třídě a @PrimaryKey některému z polí této třídy, které může vykonávat roli primárního klíče, například:

```
@Entity(tableName = "channels")
data class CatvusaChannel(
    @PrimaryKey
    val slug: String,
    val name: String,
    val description: String,
)
```

V rámci projektu jsem vytvořil 3 entity – CatvusaVideo, CatvusaChannel a CatvusaAudio, které představují videa, příspěvkové typy a obsah rádiových stanic.

Pro přístup k tabulkám se používá rozhraní realizující pattern Dao (Data Access Object).

V tomto rozhraní programátor definuje signatury funkcí, které používají výše uvedené třídy (podle toho Room zjišťuje příslušnou tabulku), a přidává příslušné anotace. Pro

jednoduché operace vložení, aktualizace a smazání Room poskytuje anotaci @Insert, @Update a @Delete. Pro složitější dotazy a dotazy k výběru dat se používá anotace @Query s příslušným SQL dotazem.

```
@Dao
interface CatvusaVideoDao {

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertAll(videos: List<CatvusaVideo>)

…

    @Query("SELECT * FROM videos WHERE id NOT IN(:ids)")
    fun videosComplement(ids: List<Int>): List<CatvusaVideo>
}
```

4.1.6 Nastavení synchronizaci obsahu

Z důvodu potřeby poskytovat obsah hlavní obrazovce Android OS a globálnímu vyhledávači a vyhovění požadavku zadavatele, aby tento obsah byl vždy aktuální, je nutné regulérně spouštět synchronizaci obsahu webu s vnitřní databází a hlavní obrazovkou Android OS.

Pro tyto účely Android Jetpack nabízí knihovnu WorkManager – API, která usnadňuje plánování spolehlivých asynchronních úloh, u kterých se očekává, že budou spuštěny, i když se aplikace ukončí nebo se zařízení restartuje. WorkManager API je vhodnou a doporučenou náhradou za všechna předchozí rozhraní API pro plánování na pozadí systému Android, včetně FirebaseJobDispatcher, GcmNetworkManager a Job Scheduler. [27]

Kromě toho, že poskytuje jednodušší a konzistentní API, má WorkManager řadu dalších klíčových výhod, včetně: [27]

- Omezení úlohy – možnosti deklarativně definovat podmínky pro spuštění úlohy (například spouštět pouze, když je zařízení připojeno k Wi-Fi nebo když je zařízení v režimu spánku nebo když má dostatek úložného prostoru atd.).
- Robustní plánování – WorkManager umožňuje plánovat práci tak, aby byla spuštěna jednorázově nebo opakovaně pomocí flexibilních oken plánování. Úlohy lze také označit a pojmenovat, což umožní plánovat jedinečné a vyměnitelné úlohy a společně monitorovat nebo rušit skupiny úloh. Naplánovaná úloha je uložena v interně spravované databázi SQLite a WorkManager se stará o to, aby tato práce přetrvávala a byla přeplánována po restartu zařízení. WorkManager navíc dodržuje funkce pro úsporu energie a osvědčené postupy, jako je Doze Mode, takže se o něj programátor nemusí starat.
- Flexibilní zásady opakování – programátor má široké možnosti nastavení chování pro případ selhání úlohy – maximální počet opakování, model výpočtu pauzy před opakováním (např. lineární, exponenciální).
- Řetězení úloh – úlohy se dají spojit dohromady, spouštět v určitém pořadí, postupně a paralelně, používat výstupy prvních úloh jako vstupy pro následující atd.

Vytvořil jsem 2 typy úloh – WpVideoSynchonizer (pro synchronizaci videí a kanálu) a WpAudioSynchronizer (pro synchronizaci obsahu rádiových stanic).

Synchronizace videí se spouští každou hodinu, protože uživatel je vidí nejen v aplikaci, ale i na hlavní obrazovce OS a zadavatel požádal o doručení změn včas (některá videa

jsou aktuální jenom v krátkém časovém okamžiku, například videa o svátcích). Pravidla opakování jsou exponenciální, ale s malou počáteční hodnotou (5 minut).

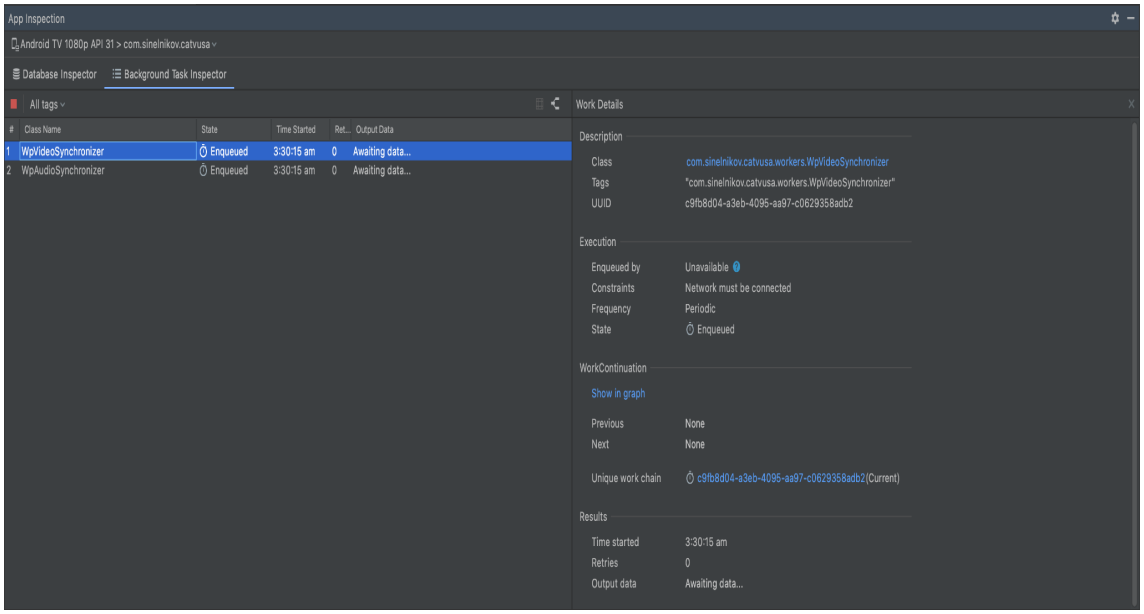
Synchronizace audia není tak důležitá, proto se dělá denně. Pravidla opakování jsou lineární, s počáteční hodnotou 1 hodina.

Obě úlohy vyžadují internetové připojení. Plánují se za spouštění aplikace, ale díky tomu že jsou nastaveny jako unikátní, vždy existuje maximálně jedna instance úlohy. ExistingPeriodicWorkPolicy.KEEP zaručuje, že při spuštění aplikace nová úloha nebude přidána, pokus jedna už existuje. Další možností je ExistingPeriodicWorkPolicy.REPLACE – zrušení staré naplánované úlohy a plánování znovu.

```
// Syncs the videos hourly
WorkManager.getInstance(baseContext).enqueueUniquePeriodicWork(
    "syncing-videos",
    ExistingPeriodicWorkPolicy.KEEP,
    PeriodicWorkRequestBuilder<WpVideoSynchronizer>(1, TimeUnit.HOURS)
        .setConstraints(constraintsBuilder.build())
        .setBackoffCriteria(BackoffPolicy.EXPONENTIAL, 5, TimeUnit.MINUTES)
        .build()
)
// Syncs the radio station daily
WorkManager.getInstance(baseContext).enqueueUniquePeriodicWork(
    "syncing-audios",
    ExistingPeriodicWorkPolicy.KEEP,
    PeriodicWorkRequestBuilder<WpAudioSynchronizer>(1, TimeUnit.DAYS)
        .setConstraints(constraintsBuilder.build())
        .setBackoffCriteria(BackoffPolicy.LINEAR, 1, TimeUnit.HOURS)
        .build()
)
```

Kromě plánování při spuštění aplikace se jedna instance úlohy spouští hned po instalaci. Uživatel uvidí obsah na hlavní obrazovce ještě před spuštěním aplikace, a když ji spustí, nebude zbytečně čekat. Mohlo by to být jediným místem plánování úloh, ale protože ne vždy se aplikace instaluje přes Google Play (mnoho zařízení postrádá tento marketplace), vytvořil jsem hlavní plánování při spuštění.

Android Studio nabízí nástroje pro monitorování a ladění těchto úloh – Background Task Inspector. [28] Vypisuje seznam zahajovaných, běžících a ukončených úloh, jejich konfiguraci, stav, vstupní a výstupní data, čas spuštění. V případě řetězce úloh generuje graf.



Obrázek 4.5. Background Task Inspector

4.1.7 Synchronizace obsahu

Samotná synchronizace videí probíhá následujícím způsobem:

1. Stahování obsahu ze serveru. Zaprvé se stahuje se seznam příspěvkových typů, potom se paralelně spouští požadavky pro každý příspěvkový typ. Čeká se na dokončení všech požadavků a výsledek se vrátí do následujícího kroku algoritmu:

```
private suspend fun fetchFreshVideoData()
    : Map<CatvusaChannel, List<CatvusaVideo>> {

    val postTypes = when (
        val postTypesResult = apiService.getPostTypes()
    ) {
        is Result.Error -> throw postTypesResult.error
        is Result.Success -> postTypesResult.data.values
    }

    return supervisorScope {
        postTypes.filter { it.taxonomies.contains("android_tv") }
            .map { async { fetchPostTypeVideos(it) } }
            .mapNotNull {
                try {
                    it.await()
                } catch (t: Throwable) {
                    null
                }
            }.toMap()
    }
}
```

Paralelní běh se vytváří za pomoci koprogramů – mechanismu jazyka Kotlin pro korporativní multitasking a paralelní běh. Prostor supervisorScope zaručuje, že selhání jednoho koprogramu nebude příčinou selhání rodičovského koprogramu a nezruší vykonávání ostatních (takže když se nepodaří dostat videa jednoho příspěvkového typu,

nebude to mít vliv na stahování videa ostatních příspěvkových typů a jejich následující použití).
2. Z databáze se dostávají videa a kanály, které nejsou v seznamu vrácených ze serveru. Tato videa a kanály se odstraňují z hlavní obrazovky OS a z databáze.
3. Přidání/obnovení videa a kanálů. V případě databáze je vykonáván příkaz INSERT se strategií vyřízení konfliktu REPLACE, přičemž pozice uložená v databázi přehrávače se nemění díky předdefinovaným funkcím equals a hashCode. API pro práci s hlavní obrazovkou OS nemá příkaz upsert, proto byl implementován ručně ve třídě TvLauncherHelper.

   Synchronizace obsahu rádiových stanic se vytváří mnohem jednodušeji – nemusíme mazat položky odstraněné z hlavní obrazovky jako v případě videí, proto můžeme smazat všechny položky z databáze a stáhnout novou verzi ze serveru. Při stahování ze serveru se nejprve staguje seznam žánrů podle slugu (classical, folk, cultural) a dodávají se identifikátory těchto žánrů. Potom se stahuje seznam hudebníků těchto žánrů. Nakonec se stahují metadata audionahrávek příslušných k těmto hudebníkům a mapování těchto údajů do objektů CatvusaAudio. Ty jsou poté ukládány v databázi.

```
return audios.map { audio ->
    val musician = musicians.find { it.id == audio.parent }!!
    val genre = genres.find { it.id == musician.genreIds.first() }!!
    CatvusaAudio(
        audio.id,
        audio.title?.rendered ?: "",
        musician.title.rendered,
        musician.embedded?.featuredMedia?.firstOrNull()?.sourceUrl,
        audio.sourceUrl,
        genre.slug
    )
}
```

4.1.8 Navigace

Pro navigaci jsem použil další knihovnu ze sady Android Jetpack – Navigation Component.

Navigation Component poskytuje řadu výhod včetně následujících:

- Zpracování přechodu mezi fragmenty.
- Správné zpracování akcí Nahoru a Zpět ve výchozím nastavení.
- Poskytování standardizovaných zdrojů pro animace a přechody.
- Implementace a řízení deep links.
- Implementace uživatelského rozhraní pro sadu navigačních šablon, například spodní navigaci nebo vyjíždějící menu.
- Safe Args – plugin pro systém sestavování Gradle, který poskytuje bezpečnost typu při navigaci a předávání dat mezi destinacemi.
- Podpora ViewModel – ViewModel lze zahrnout do navigačního grafu a sdílet jím data související s uživatelským rozhraním mezi destinacemi grafu. [29]

Navigation Component se skládá ze třech klíčových částí, které jsou popsány níže:

- Navigační graf: Zdroj XML, který obsahuje všechny informace související s navigací na jednom centralizovaném místě. To zahrnuje všechny jednotlivé oblasti obsahu v aplikaci nazývané destinace a také možné cesty, kterými se může uživatel aplikací vydat.

- NavHost: Prázdný kontejner, který zobrazuje destinace z navigačního grafu. Navigation Component obsahuje výchozí implementaci NavHost, NavHostFragment, která zobrazuje navigační destinace (fragmenty).
- NavController: Objekt, který spravuje navigaci aplikací v rámci NavHost. NavController organizuje výměnu cílového obsahu v NavHost, když se uživatelé pohybují v aplikaci. [29]

Destinace v navigačním grafu může být představena prvkem activity, fragment, dialog nebo určitou vlastní třídou implementovanou v projektu. [30] Tento projekt používá jenom prvky fragment. Každý prvek destinace obsahuje následující atributy:

- id – používá se pro odkazování na této destinaci.
- name – jméno třídy, která představuje tuto destinaci.
- label – uživatelsky přijatelné jméno destinace, používá se ve standardních navigačních elementech Androidu.

Uvnitř prvků popisujících destinaci XML lze definovat argumenty – vstupní data pro destinace. Zahrnují jméno proměnné a představující třídu. Plugin Safe Args pro Navigation Component vygeneruje kód pro cesty, které vedou do této destinace s použitím těchto argumentů, a takovým způsobem zaručí typo-bezpečnou navigaci. Alternativou využití tohoto pluginu je použití třídy Bundle (funguje jako asociativní pole) a explicitního přetypování, což je prostorem pro chyby.

Cesta mezi destinacemi může být představována prvkem action nebo deeplink.

Prvek action představuje cestu, kterou programátor přímo používá v kódu. Obsahuje následující atributy:

- id – používá se pro odkazování na tuto cestu.
- destination – identifikátor cílové destinace.
- popUpTo – mění chování navigačního zásobníku. Přijímá identifikátor destinace a při vyvolání action z navigačního zásobníku jsou smazány všechny destinace specifikované v popUpTo.
- popUpToInclusive – přijímá „true“ nebo „false“ a definuje, jestli destinace specifikovaná v popUpTo musí také být smazána.

```
<fragment
    android:id="@+id/video_details_fragment"
    android:name="com.sinelnikov.catvusa.fragments.VideoDetailsFragment"
    android:label="Video Details">
    <action
        android:id="@+id/action_video_details_fragment_to_video_details_fragment"
        app:destination="@id/video_details_fragment"
        app:popUpTo="@id/browse_fragment"
        app:popUpToInclusive="false" />
    <action
        android:id="@+id/action_video_details_fragment_to_video_player_fragment"
        app:destination="@id/video_player_fragment" />
    <action
        android:id="@+id/action_video_details_fragment_to_entry_fragment"
        app:destination="@id/entry_fragment" />
    <argument
        android:name="video"
        app:argType="com.sinelnikov.catvusa.model.pojo.CatvusaVideo" />
</fragment>
```

Příkladem použití popUpTo je action_video_details_fragment_to_video_details_fragment. Když se uživatel nachází na obrazovce detailu videa, přechází mezi videi ze seznamu „Related videos“ a stiskne tlačítko „zpět“, bude přesměrován hned na hlavní obrazovku, místo toho, aby byl přesměrován na detail předchozího videa.

Navigační akce se vykonávají za pomoci metody navigate objektu NavigationController:

```
findNavController().navigate(
VideoDetailsFragmentDirections.actionVideoDetailsFragmentToEntryFragment()
)
```

Dalším navigačním prvkem je deepLink. Deep links jsou identifikátory URI jakéhokoli schématu, které uživatele zavede přímo do konkrétní části aplikace. [31] Obyčejně se používají s jinými aplikacemi, v případě této aplikace s globálním vyhledávačem nebo s hlavní obrazovkou operačního systému (kam se nahrávají videa a kanály).

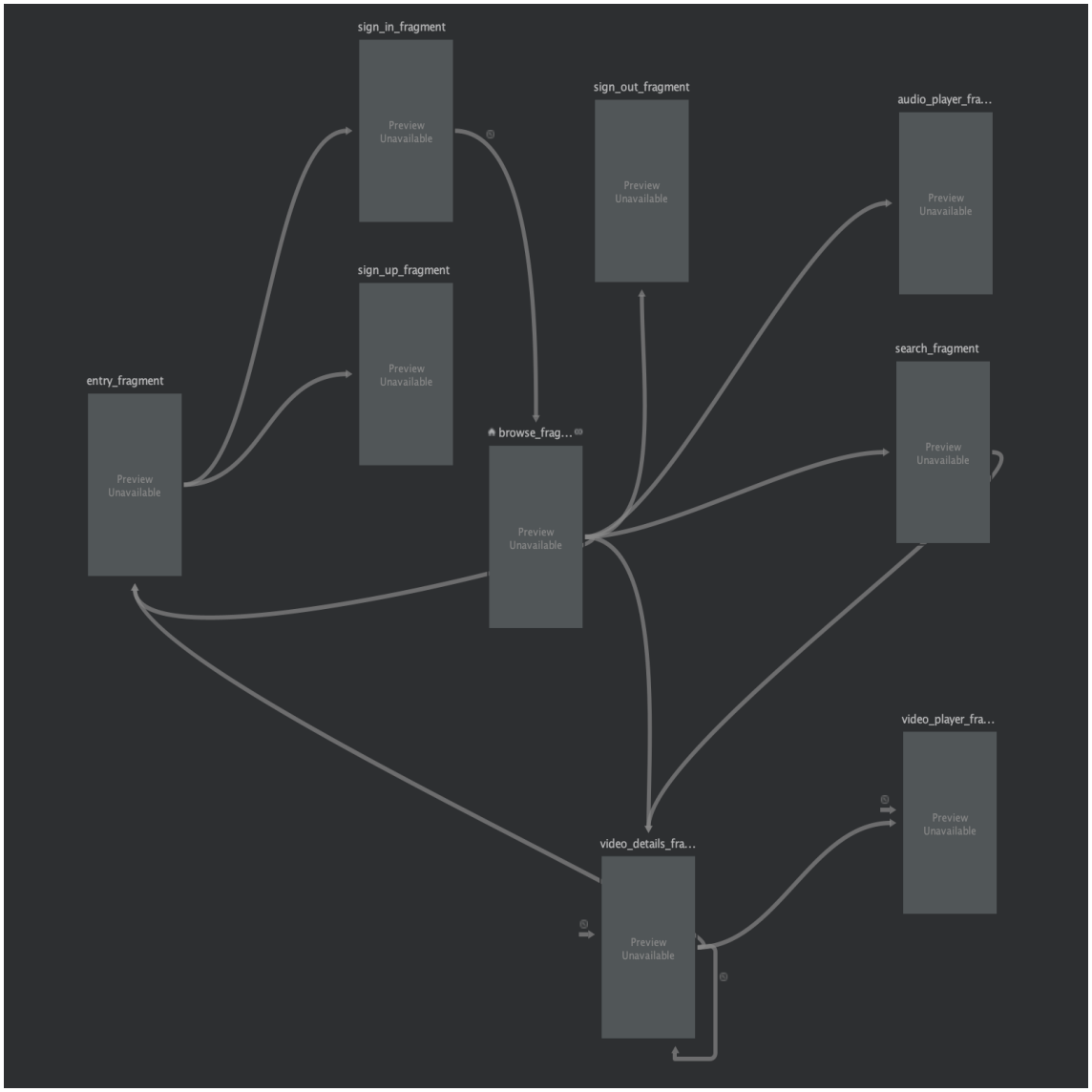
```
<fragment
    android:id="@+id/browse_fragment"
    android:name="com.sinelnikov.catvusa.fragments.BrowseFragment"
    android:label="Media Browser">

…

<argument
    android:name="channelSlug"
    app:argType="string"
    android:defaultValue="__propagated__" />
<deepLink
    android:id="@+id/homeScreenChannelDeepLink"
    app:uri="catvusa://com.sinelnikov.catvusa/channel/{channelSlug}" />
</fragment>
```

V navigačním grafu do XML prvku deepLink se předává URI, který podporuje placeholder pro argumenty. Navigační framework automaticky zachytí odkazy specifikovaného formátu, vyčlení argumenty a otevře příslušnou destinaci s předanými argumenty. Výše uvedený deep link se používá při kliknutí na ikonu kanálu na hlavní obrazovce OS a otevírá hlavní obrazovku aplikace na příslušném kanálu.

Navigační graf celé aplikace vypadá následovně:

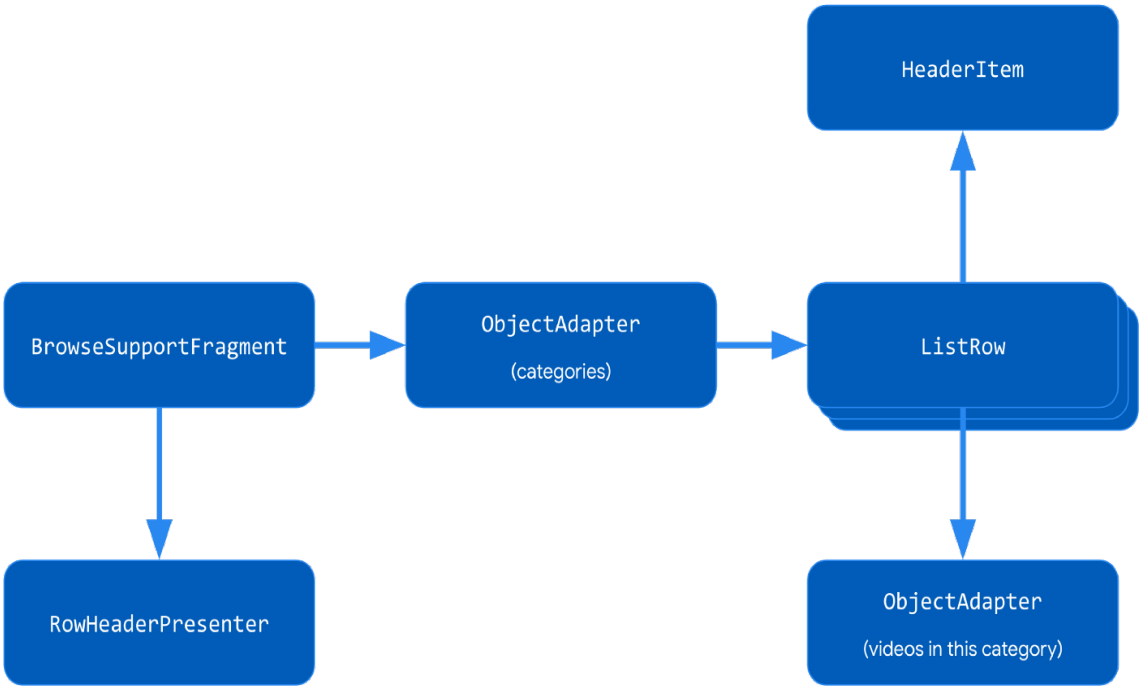


Obrázek 4.6. Navigační graf aplikace

4.2 Implementace jednotlivých částí

4.2.1 Hlavní obrazovka

Implementace uživatelského rozhraní hlavní obrazovky se zakládá na androidx.leanback.app.BrowseSupportFragment. Tato třída z knihovny Leanback poskytuje základní strukturu pro zjednodušenou implementaci seznamu kategorií s videi. Skládá se ze dvou fragmentů – RowsSupportFragment pro hlavní obsah a HeadersSupportFragment pro menu – a zahrnuje jejich integraci (změna řádku v jednom fragmentu mění řádek ve druhém). Programátor musí nastavit adaptér s daty.



Obrázek 4.7. Schémá závislosti tříd BrowseSupportFragment

Každý prvek v adaptéru představuje řádek a musí dědit třídu Row. Použil jsem standardní prezentér ListRowPresenter, který vytváří UI pro prvky ListRow. ListRow se skládá z HeaderItem (jehož součástí jsou identifikátor a jméno) a dalšího adaptéru, který reprezentuje seznam prvku řádku.

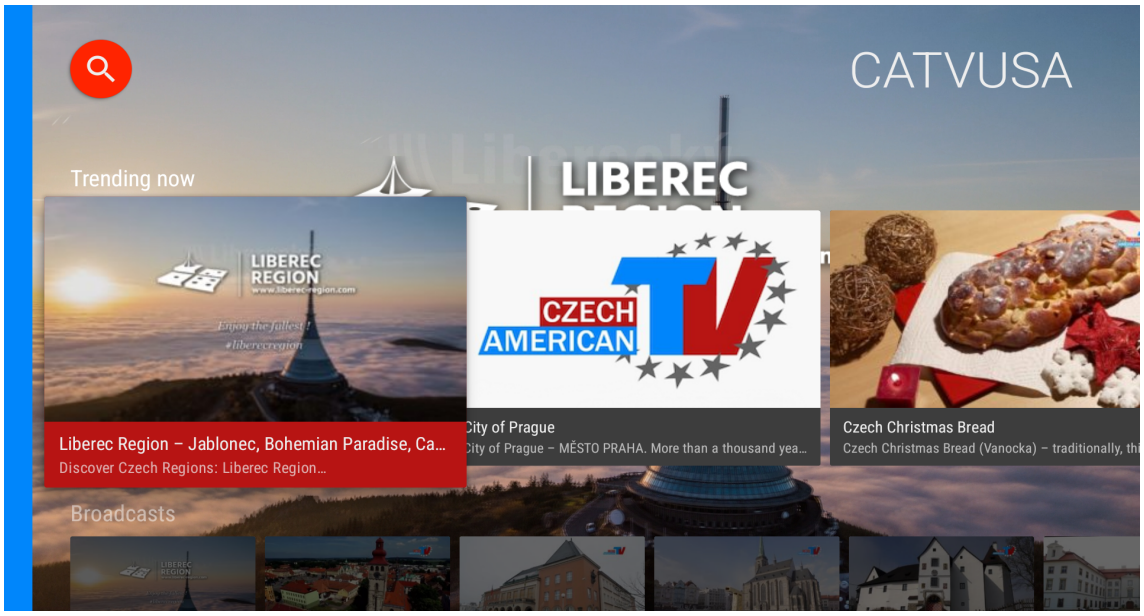
Obrázek 4.8. Hlávní obrazovka aplikace

Pro reprezentaci seznamu videí jednoho kanálu jsem zvolil PagingDataAdapter, který načítá data postupně podle toho, jak uživatel prochází seznamem, ze zdroje a podle konfigurací, které programátor předá do konstruktoru. Takový zdroj odnedávna poskytuje ORM Room, stačí ho použít jako výstupní hodnotu v DAO:

```
@Query("SELECT * FROM videos WHERE channelSlug = :channelSlug
```

```
        ORDER BY createdAt DESC")
        fun videosByChannelPagingSource(channelSlug: String):
            PagingSource<Int, CatvusaVideo>
```

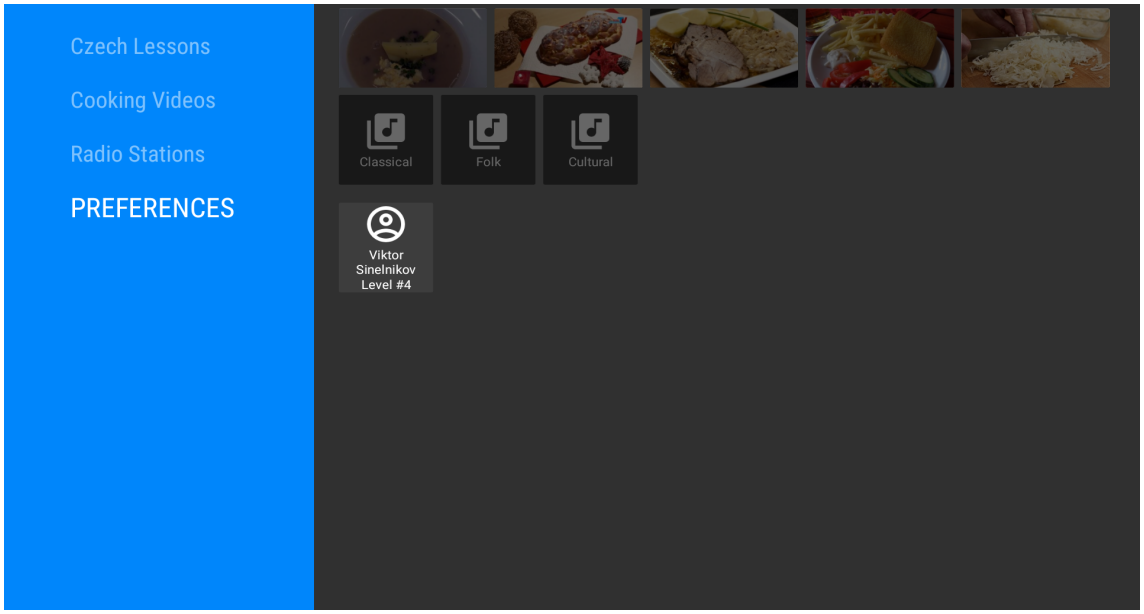
Pro reprezentaci vzhledu jednotlivých videí jsem napsal VideoCardPresenter, ve kterém jsem použil ImageCardView, nastavil sledování stavu zvolení a příslušnou změnu barvy pozadí textu. Argumentem konstruktoru tohoto prezentéru je jen výčtový typ VideoCardType (REGULAR nebo BIG). Rozměry se zadávají přímo v tomto výčtovém typu (v Kotlin na rozdíl od Javy výčtové typy mohou mít vlastnosti a metody).



Obrázek 4.9. Řádek propagovaných videí na hlavní obrazovce aplikace

Pro čtverce reprezentující rádiové stanice a stav přihlášení jsem vytvořil třídu GridItem a prezentér GridItemPresenter.

GridItem obsahuje typ (výčtový typ, používá se pro identifikaci při zpracování kliknutí), nápis a identifikátor ikony.



Obrázek 4.10. Řádek prvků GridItem

Za pozadí odpovídá třída androidx.leanback.app.BackgroundManager. Zaručuje spojitost pozadí mezi různými navigačními destinacemi (Activity a Fragment). Při každé změně zvoleného videa pozadí je volána funkce updateBackground, ve které se za pomoci knihovny pro stahování a kešování obrázku Glide načítá hlavní obrázek videa v nejvyšším rozlišení a předává se do BackgroundManager.

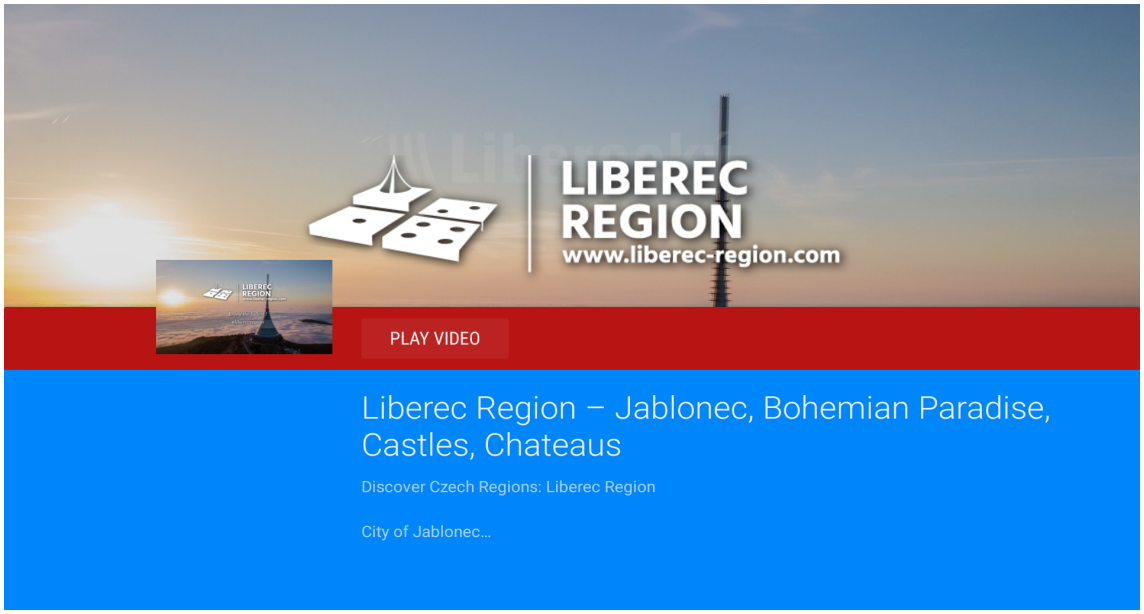
Jak lze vidět z navigačního grafu, z této stránky je možné přejít na obrazovku detailu videa (po kliknutí na kartičku s videem), rádia (po kliknutí na kartičku s rádiem), vyhledávání (po kliknutí na ikonu lupy) a přihlášení/odhlášení.

4.2.2 Obrazovka detailu videa

Implementace uživatelského rozhraní této obrazovky se zakládá na třídě androidx.leanback.app.DetailsSupportFragment. Stejně jako BrowseSupportFragment potřebuje nastavení a naplnění adaptéru, ale mění chování, když se v adaptéru nachází androidx.leanback.widget.FullWidthDetailsOverviewRowPresenter – poskytuje mu celou výšku obrazovky a spravuje uživatelský vstup – procházení textu a akci.

Zobrazení vytvořené pomocí FullWidthDetailsOverviewRowPresenter se skládá ze tří částí: zobrazení loga vlevo, zobrazení seznamu akcí nahoře a přizpůsobitelné zobrazení podrobného popisu vpravo. [32]

Samotné video se předává jako argument navigaci za pomoci pluginu Safe Args.



Obrázek 4.11. Obrazovka detailu videa

Při otevření obrazovky se načítá vyžadovaná úroveň podpory pro toto video, porovnává se s úrovní podpory uživatele a podle výsledku se přidává akční tlačítko „Play Video“, které buď otevře video, nebo ukáže hlášení o nedostačující úrovni členství.

Dalším elementem na této obrazovce je seznam souvisejících videí. Portál CATVUSA ještě nemá tuto funkcionalitu, ale je v plánu, proto jsme se spolu se zadavatelem rozhodli zpřístupnit deset náhodných videi ze stejného kanálu.

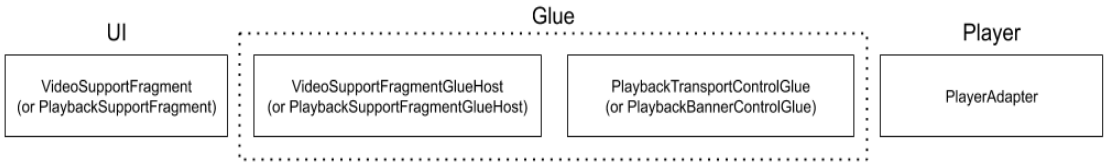
Obrázek 4.12. Řadek souvisejících videí

4.2.3 Obrazovka video přehrávače

Knihovna Leanback odděluje uživatelské rozhraní s ovládacími prvky přenosu od přehrávače, který přehrává video. Toho je dosaženo pomocí dvou komponent: PlaybackSupportFragment pro zobrazení ovládacích prvků přenosu (a volitelně videa) a PlayerAdapter pro zapouzdření přehrávače médií. [33]

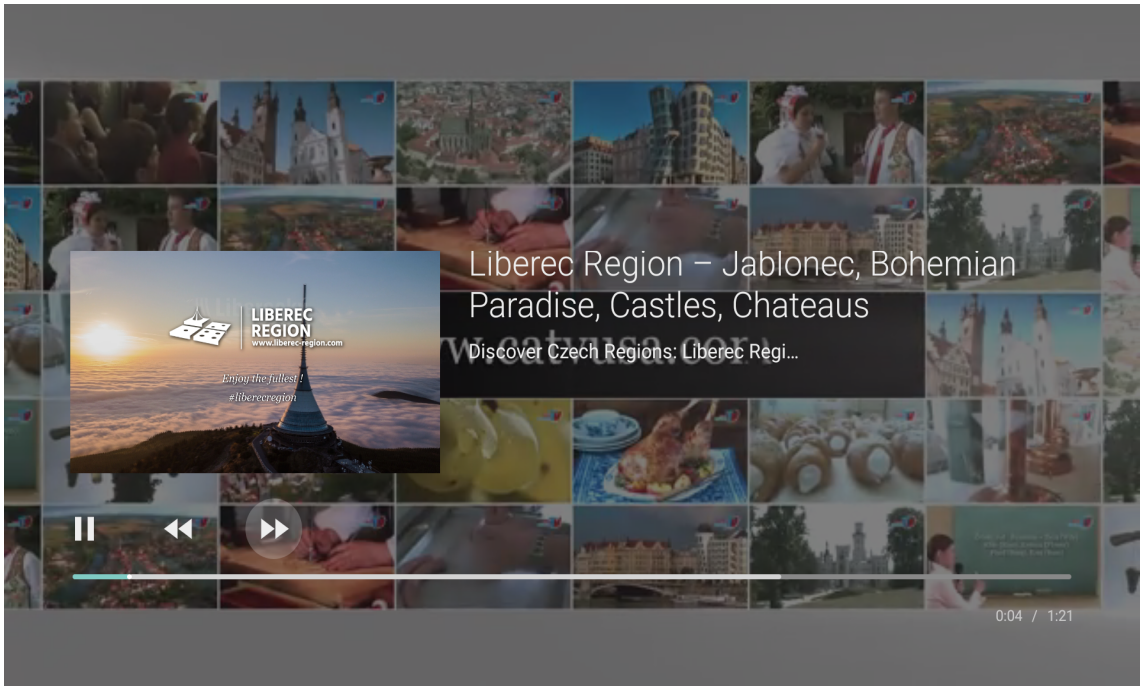
V případě přehrávání videa je doporučena základní třída VideoSupportFragment, která je rozšířením třídy PlaybackSupportFragment a poskytuje plochu pro zobrazení videa a může být integrována s různými přehrávači. [34]

Spojkou mezi fragmentem a adaptérem je Glue. Zpracovává uživatelský vstup od akcí definovaných v adaptérech a vyvolává příslušné funkce přehrávače.



Obrázek 4.13. Konceptualní schéma Glue

Pro implementaci možnosti rychlého převíjení jsem musel vytvořit vlastní Glue, zděděné od PlaybackTransportControlGlue. Předdefinoval jsem funkci onCreatePrimaryActions (kde jsem přidal příslušná tlačítka do adaptéru), onActionClicked (kde jsem definoval reakce na kliknutí těchto tlačítek), přidal jsem funkci skipForward a skipBackward, které mění pozice přehrávače a jsou používány nejen výše uvedenou funkcí onActionClicked, ale i zpracovatelem dálkového ovladače. Také jsem přidal funkce setMetadata pro snadnější převod CatvusaVideo na seznam MediaItem', které přijímá přehrávač Exoplayer a rozhraní MediaSession určené pro Android.



Obrázek 4.14. Obrazovka přehrávání videa

MediaSession vystavuje informaci o přehrávání operačnímu systému a jiným aplikacím a povoluje řízení přehráváním zvenku. Exoplayer poskytuje pomocnou třídu pro zpracování těchto akcí – MediaSessionConnector.

Je velmi důležité, aby uživatel pokračoval ve sledování videa tam, kde naposled skončil. Tato informace musí být zapsána do databáze. Zápis při ukončení prohlédnutí videa není dostatečný, protože televize se může neočekávaně vypnout (například při výpadku elektrické energie), a proto jsem vytvořil úlohu, která se spustí každých 5 sekund, udělá zápis do databáze a obnovuje řádek „Watch Next“ na hlavní obrazovce operačního systému.

4.2.4 Obrazovka přehrávače rádiových stanic

Obrazovka radiové stanice je podobná obrazovce přehrávače videí, ale jako základ se místo VideoSupportFragment používá jeho rodičovská třída PlaybackSupportFragment. Audionahrávky se dostávají z databáze podle žánru v náhodném pořadí a přidávají se do fronty přehrávání. Obrázek na pozadí byl přidán ručně za pomoci předefinování metody fragmentu onCreateView.

Obrázek 4.15. Obrazovka přehrávání audia

4.2.5 Obrazovka vyhledávání

Základem této obrazovky je třída SearchSupportFragment, která poskytuje pole pro zadávání textu a spravuje rozpoznávání řeči. Použití tohoto fragmentu vyžaduje implementaci rozhraní SearchSupportFragment.SearchResultProvider, které reaguje na uživatelský vstup. Zahrnuje metody getResultsAdapter (specifikace adaptéru, uživatelského rozhraní seznamu), onQueryTextChange (určuje, pro který vstup se volá onQueryTextSubmit) a onQueryTextSubmit (odpovídá za vyhledávání a naplnění adaptéru). Vyhledávání se určuje podle vlastnosti CatvusaVideo.searchableTitle, která se vytvaruje v konstruktoru z vlastnosti title odstraňováním interpunkčních symbolů za pomoci regulárního výrazu.

```
fun searchableText(text: String) =
text.replace(Regex("[^A-Za-z0-9 ]"), "").lowercase(Locale.getDefault())
```

Vyhledávání v databázi se provádí za pomoci příkazu LIKE, který vyhledává hledaný text v jakémkoliv místě názvu.

```
@Query("SELECT * FROM videos WHERE searchableTitle
    LIKE '\%' || :title || '\%'")
suspend fun getVideosByTitle(title: String): List<CatvusaVideo>
```

Obrázek 4.16. Obrazovka vyhledávání v aplikaci

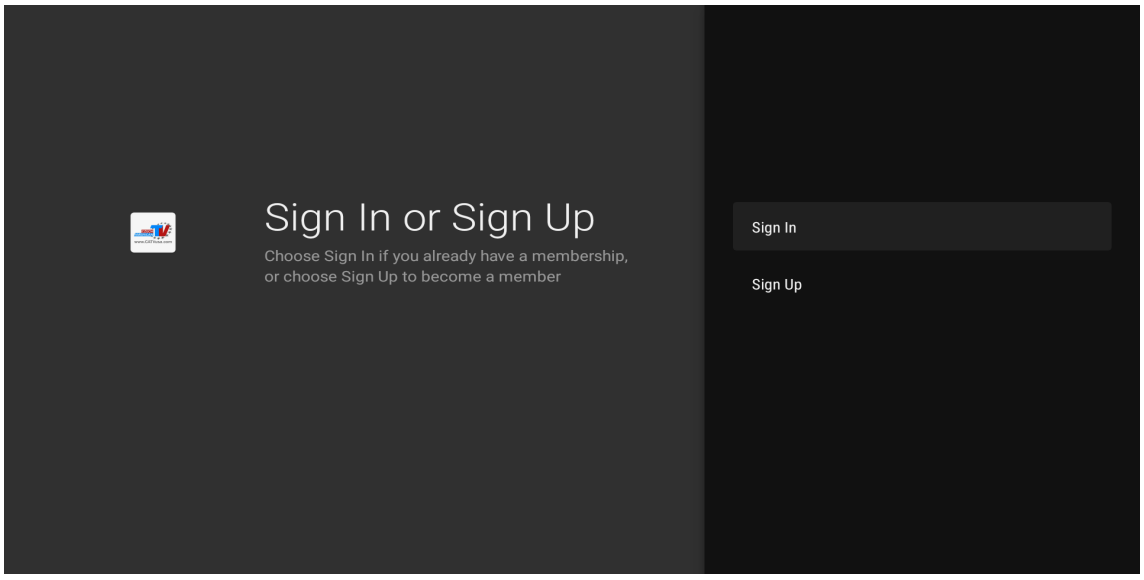
4.2.6 Obrazovky přihlašování, registrace, a odhlašování

Obrazovky přihlašování, registrace a odhlašování jsou vytvořeny na základě GuidedStepSupportFragment.

GuidedStepSupportFragment se používá k vedení uživatele rozhodnutím nebo řadou rozhodnutí. Skládá se z naváděcího pohledu vlevo a pohledu vpravo obsahujících seznam možných akcí. [35]

Předdefinoval jsem funkce onCreateGuidance (kde se definuje informace zobrazená vlevo), onCreateActions (kam se přidávají akce), onGuidedActionClicked (zpracování reakcí definovaných v onCreateActions).

Nepřihlášený uživatel je přesměrován na obrazovku „Sign In or Sign Up“ (EntryFragment), která má 2 akce – přihlásit se a zaregistrovat se. S2Member Pro Remote API nepodporuje registraci, proto při zvolení akce „Sign Up“ bude otevřen webový prohlížeč na stránce registrace.



Obrázek 4.17. Obrazovka přihlašování

Při zvolení akce „Sign In“ bude otevřena následující obrazovka – SignInFragment. Tato obrazovka obsahuje 2 editovatelné akce (pole pro vstup) – username a password a jednu obyčejnou – continue. Při zvolení akce continue jsou využita data z prvních dvou akcí a posílají se na server pro ověření platnosti přihlašovacích údajů. Jsou-li údaje správné, uživatel je přesměrován na hlavní obrazovku, v opačném případě se ukáže chybové hlášení.

Přihlášený uživatel bude přesměrován na obrazovku odhlašování, která obsahuje 2 akce – potvrdit a zrušit.

4.2.7 Globální vyhledávač

Android TV používá vyhledávací rozhraní Android k získávání dat obsahu z nainstalovaných aplikací a poskytování výsledků vyhledávání uživateli. [36]

Aplikace musí poskytnout datová pole operačnímu systému, ze kterých se generují navrhované výsledky vyhledávání, když uživatel zadává znaky do vyhledávacího dialogu. [36]

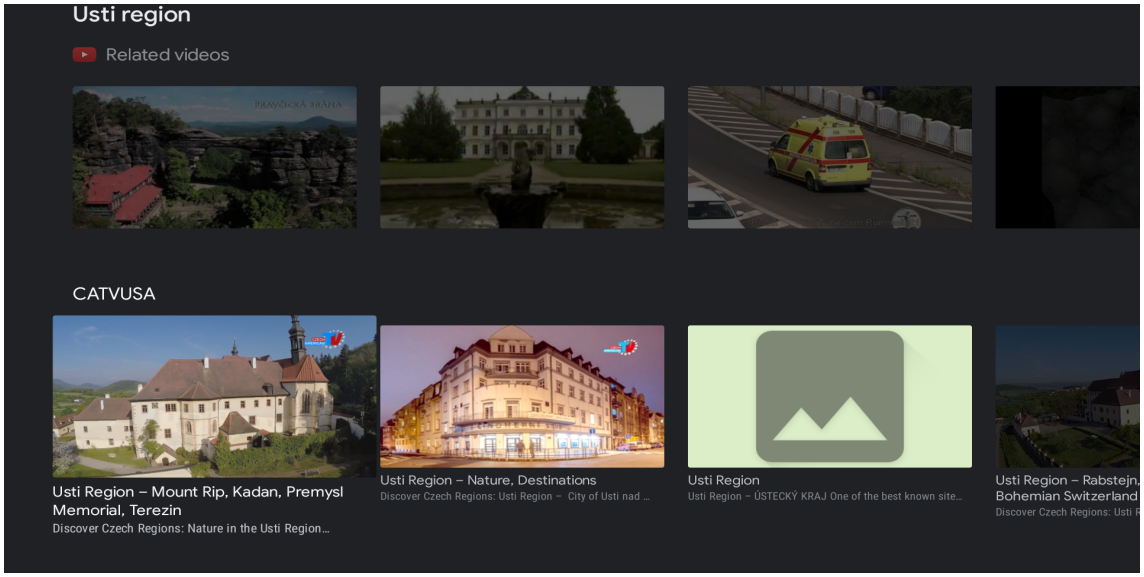
To se dělá za pomoci implementace ContentProvider (poskytovatele obsahu), který poskytuje návrhy spolu s konfiguračním souborem searchable.xml, jenž popisuje poskytovatele obsahu a další důležité informace pro Android TV. [36]

Předdefinoval jsem funkci ContentProvider.query, která vrátí Cursor (reprezentace seznamu databázových řádků) ke sloupcům uvedeným ve třídě SearchManager. Vytvořil jsem mapování svých databázových sloupců na sloupce SearchManager přímo v SQL-dotazu:

```
@Query(
    "SELECT " +
        "id as \${BaseColumns._ID}, " +
        "id as \${SearchManager.SUGGEST_COLUMN_INTENT_DATA_ID}, " +
        "title as \${SearchManager.SUGGEST_COLUMN_TEXT_1}, " +
        "excerpt as \${SearchManager.SUGGEST_COLUMN_TEXT_2}, " +
        "imageUrl as \${SearchManager.SUGGEST_COLUMN_RESULT_CARD_IMAGE}"
    + " FROM videos WHERE searchableTitle LIKE '\%' || :title || '\%'"
)
fun contentProviderQuery(title: String): Cursor?
```

Konfigurační soubor searchable.xml definuje deep link, který je poslán do aplikace, když uživatel vybere navrhovaný výsledek hledání, formát dotazu předaný do ContentProvider.query, název aplikace v globálním vyhledávači.

Hlasový asistent Google Assistant podle kontextu může odhadnout, že jde o vyhledávání, a předat hledaná slova do ContentProvider.



Obrázek 4.18. Seznam videí nalezených přes globální vyhledávač

Závěr

Nejprve byl stanoven cíl práce a požadavky na aplikaci. Ty byly sepsány v úzké spolupráci se zadavatelem a na základě funkčních požadavků byl sestaven UML diagram. V následující kapitole byl probrán systém řízení obsahu webového portálu zadavatele, popsán způsob předání dat ze serveru do klientské aplikace – REST API. Dále pak byl popsán operační systém Android TV a zásady návrhu aplikace pro tento operační systém. Ve třetí kapitole byl zanalyzován datový model webového portálu, popsány odpovídající API metody pro získání dat a navrženo uživatelské rozhraní aplikace. V poslední, čtvrté kapitole, byla popsána implementace aplikace. Hlavní cíl diplomové práce – implementace aplikace – byl splněn. Finální verze byla představena zadavateli, otestována, schválena, napojena na produkční API a je připravena pro nahrání do Google Play. Nebyly implementovány jednotkové a integrační testy, ale použita architektura spolu vkládáním závislostí povolujících jejich implementaci bez refaktoringu zdrojového kódu.

Aplikace má velký potenciál pro budoucí rozvoj. Zaprvé může být provedena integrace s pluginem Radio Manager, aby správce mohl určovat pořadí přehrání audionahrávek ve webové verzi rádia (to bude vyžadovat vytvoření API pluginu Radio Manager). Dále je možné vytvoření vlastního API místo použití zabudovaného do Wordpressu, a to za účelem snížení počtu požadavků a současně vedoucí ke snížení zatížení serveru a času využití dat. Přinejmenším by požadovaná úroveň podpory mohla být zavedena jako metadata příspěvku, co by eliminovalo nutnost využít další příslušné API na obrazovce detailu videa.

Velkým zlepšením by bylo zbavení se potřeby synchronizace celé databáze vynucené zabezpečením fungování globálního vyhledávače, zavedením synchronizace obsahu ze serveru se službami Google (což bude vyžadovat značné úsilí ze strany webmasterů).

Další možností zlepšení je zavedení příslušné analytiky pro sledování uživatelských akcí, popularity jednotlivých videí a kanálů.

Vzhledem k intenzivní spolupráci ČVUT a CATVUSA mohou být potřebné změny na straně serveru provedeny v rámci bakalářských a diplomových prací dalších studentů a vyvinutá aplikace se může dále rozvíjet.

Literatura

[1] *Design principles — Designing for TV* [online]. [cit. 2021-09-06]. Dostupné z: https://tv.withgoogle.com/design-principles/designing-for-tv.html

[2] DRDLÍČEK, Michael. *Aplikace pro mobily a chytré televize pro neziskovou TV*. Praha, 2017. Diplomová práce. České vysoké učení technické v Praze.

[3] LENSKÝ, David. *Tvorba mobilní iOS aplikace pro neziskovou TV*. Praha, 2017. Bakalářská práce. České vysoké učení technické v Praze.

[4] *Wordpress Developer Resources: Post Types* [online]. [cit. 2022-01-04]. Dostupné z: https://wordpress.org/support/article/post-types/

[5] WILLIAMS, Brad, David DAMSTRA a Hal STERN. *Professional WordPress: Design and Development*. 3rd ed. Indianapolis, IN: Wrox Press, 2015, s. 128. ISBN 978-1-118-98724-7.

[6] WILLIAMS, Brad, David DAMSTRA a Hal STERN. *Professional WordPress: Design and Development*. 3rd ed. Indianapolis, IN: Wrox Press, 2015, s. 140. ISBN 978-1-118-98724-7.

[7] *Slovník IT pojmů: Application programming interface* [online]. [cit. 2022-01-04]. Dostupné z: https://www.becorp.cz/slovnik/application-programming-interface/

[8] *Zdroják: REST - architektura pro webové API* [online]. [cit. 2022-01-04]. Dostupné z: https://zdrojak.cz/clanky/rest-architektura-pro-webove-api/

[9] *REST API Tutorial: Using HTTP Methods for RESTful Services* [online]. [cit. 2022-01-04]. Dostupné z: https://www.restapitutorial.com/lessons/httpmethods.html

[10] *Android Developers: TV Apps checklist* [online]. [cit. 2022-01-04]. Dostupné z: https://developer.android.com/training/tv/publishing/checklist

[11] *SUPSKV: Czech-American TV začala spolupracovat s naší školou* [online]. [cit. 2022-01-04]. Dostupné z: https://supskv.cz/index.php/spoluprace-skoly-s-czech-american-tv/

[12] *Online hovor z Johnem Honnerem, 21.09.2020*. Praha.

[13] *WP REST API - Filter parameter for posts endpoints* [online]. [cit. 2022-01-04]. Dostupné z: https://github.com/WP-API/rest-filter

[14] *Wordpress Developer Resources: REST API Handbook - Pagination* [online]. [cit. 2022-01-04]. Dostupné z: https://developer.wordpress.org/rest-api/using-the-rest-api/pagination/

[15] *ISO 8601-1:2019: Representation of dates and times*.

[16] *Pro API for Remote Operations* [online]. [cit. 2022-01-04]. Dostupné z: https://s2member.com/kb-article/pro-api-for-remote-operations/

[17] *Package s2Member: API_Functions - is_post_protected_by_s2member* [online]. [cit. 2022-01-04]. Dostupné z: https://www.s2member.com/codex/stable/s2member/api_functions/functions/#src_doc_is_post_protected_by_s2member()

[18] *Android TV: Browse View* [online]. [cit. 2022-01-04]. Dostupné z: https://tv.withgoogle.com/tv-apps/browse-view.html

[19] *Android TV: TV Apps - In-app Search* [online]. [cit. 2022-01-04]. Dostupné z: https://tv.withgoogle.com/tv-apps/in-app-search.html

[20] *Android TV: Patterns - Dialog* [online]. [cit. 2022-01-04]. Dostupné z: https://tv.withgoogle.com/patterns/dialog.html

[21] *Android Developers: Add build dependencies* [online]. [cit. 2022-01-04]. Dostupné z: https://developer.android.com/studio/build/dependencies

[22] *Android Developers: Configure the app module* [online]. [cit. 2022-01-04]. Dostupné z: https://developer.android.com/studio/build/configure-app-module

[23] *Android Developers: Configure build variants* [online]. [cit. 2022-01-04]. Dostupné z: https://developer.android.com/studio/build/build-variants

[24] *Android TV: best practices for engaging apps* [online]. [cit. 2022-01-04]. Dostupné z: https://medium.com/androiddevelopers/android-tv-best-practices-for-engaging-apps-acd0219ff395

[25] TREBILCOX-RUIZ, Paul. *Android TV Apps Development*. Berkeley, CA: Apress, 2016, s. 32. ISBN 978-1-4842-1784-9.

[26] *Demystifying RowsSupportFragment and BrowseSupportFragment* [online]. [cit. 2022-01-04]. Dostupné z: https://medium.com/swlh/demystifying-rowssupportfragment-and-browsesupportfragment-7985414b9d29

[27] *Android Developers: Schedule tasks with WorkManager* [online]. [cit. 2022-01-04]. Dostupné z: https://developer.android.com/topic/libraries/architecture/workmanager

[28] *Android Developers: Debug your WorkManager Workers with Background Task Inspector* [online]. [cit. 2022-01-04]. Dostupné z: https://developer.android.com/studio/inspect/task

[29] *Android Developers: Navigation* [online]. [cit. 2022-01-04]. Dostupné z: https://developer.android.com/guide/navigation

[30] *Android Developers: Get started with the Navigation component* [online]. [cit. 2022-01-04]. Dostupné z: https://developer.android.com/guide/navigation/navigation-getting-started

[31] *Android Developers: Handling Android App Links* [online]. [cit. 2022-01-04]. Dostupné z: https://developer.android.com/training/app-links

[32] TREBILCOX-RUIZ, Paul. *Android TV Apps Development*. Berkeley, CA: Apress, 2016, s. 37. ISBN 978-1-4842-1784-9.

[33] *Android Developers: Use Leanback transport controls* [online]. [cit. 2022-01-04]. Dostupné z: https://developer.android.com/training/tv/playback/transport-controls

[34] TREBILCOX-RUIZ, Paul. *Android TV Apps Development*. Berkeley, CA: Apress, 2016, s. 43. ISBN 978-1-4842-1784-9.

[35] *Android Developers: GuidedStepSupportFragment* [online]. [cit. 2022-01-04]. Dostupné z: https://developer.android.com/reference/androidx/leanback/app/GuidedStepSupportFra

[36] *Android Developers: Make TV apps searchable* [online]. [cit. 2022-01-04]. Dostupné z: https://developer.android.com/training/tv/discovery/searchable