ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta riadenia
a informatiky

Semestrálna práca z predmetu

vývoj aplikácií pre mobilné zariadenia

POPSHELF

Vypracoval: Dávid Bolko

Študijná skupina: 5ZYR21

Akademický rok: 2024/2025

V Žiline dňa 8.6.2025

Obsah

Zoznam obrázkov

Úvod

V aktuálnej dobe existuje nespočetné množstvo zábavného obsahu alebo náučného obsahu, či už v digitálnej forme, akými sú videohry, filmy alebo vo fyzickej forme, ako sú napríklad knihy. Užívateľ takéhoto obsahu, ktorý toho sleduje, číta alebo hrá veľa súčasne môže naraziť na problém, že niekedy nevie čo čítal naposledy, potrebuje si plánovať čo bude čítať, sledovať alebo hrať neskôr, alebo úplne zabudne čo už videl, dohral alebo prečítal. Existuje niekoľko riešení, ktoré však po väčšinou riešia iba jednu kategóriu obsahu napr. knihy, alebo hry.

Cieľom tejto aplikácie je vytvoriť prostredie a nástroj, ktorým si používatelia budú môcť evidovať väčšinu foriem zábavného obsahu a rozdeliť si tituly do kategórii či už dielo „dokončili“ resp. prečítali, dohrali, dopozerali alebo to len plánujú. V podstate ide o nástroj, kde si používatelia môžu vytvárať takzvanú „digitálnu poličku zábavného obsahu“.

Nápad na aplikáciu vznikol z vlastnej potreby mať jedno miesto, kde si môžem všetok zábavný obsah evidovať, aby som vedel čo sa mi ako páčilo a hlavne aby som mal prehlaď čo ešte plánujem hrať alebo si pozrieť.

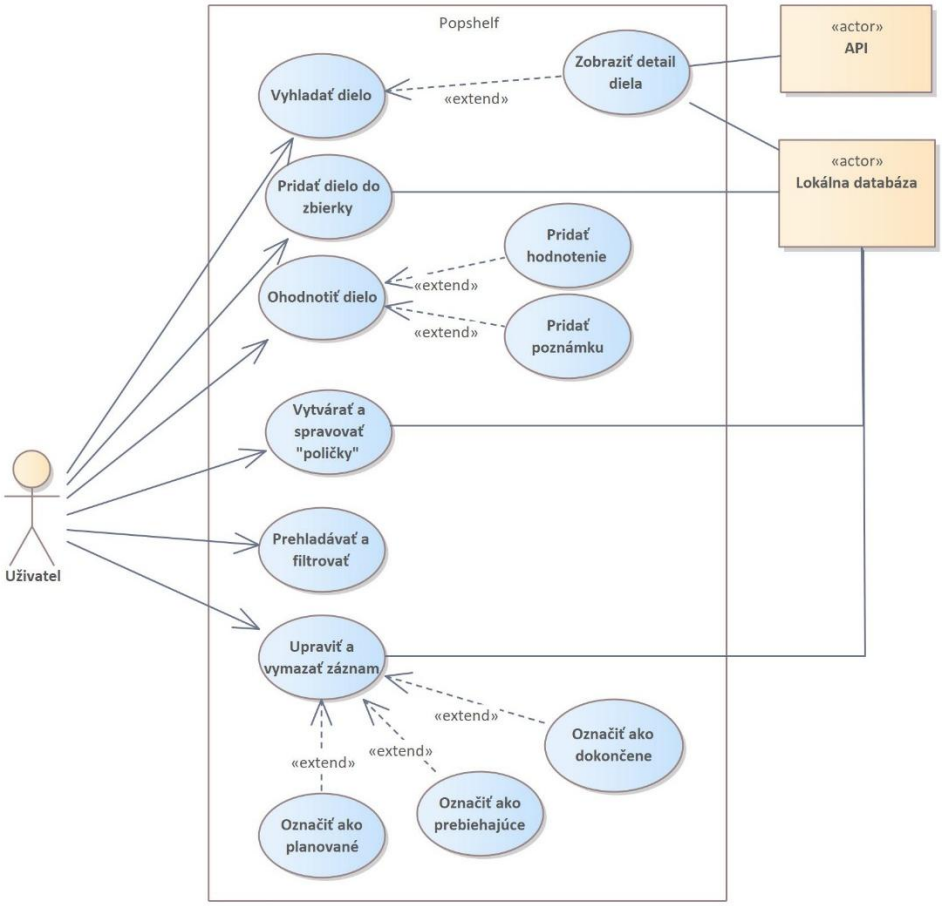
Návrh riešenia problému

Krátka analýza

Aplikácia a jej implementácia je z pohľadu ovládania veľmi jednoduchá, obsahuje málo funkcií, ktoré sú ale veľmi dôležité pre fungovanie celej aplikácie. Aplikácia je navrhovaná s ohľadom na jednoduchosť používania. Dôležite bolo aby užívateľ vedieť nájsť dielo, ktoré chce a uložil si ho tak ako chce. Medzi implementované prípady použitá patria funkcie:

- Vyhľadať dielo
- Pridať dielo do zbierky/poličky
- Ohodnotiť dielo
- Vytvárať a spravovať poličky
- Upraviť alebo zmazať záznam

Na obrázku č.1 je možné vidieť aj diagram prípadov použitia implementovanej aplikácie Popshelf. Diagram v podstate opisuje už spomínane funkcie a taktiež ako jednotlivé funkcie interagujú s vonkajšími časťami.

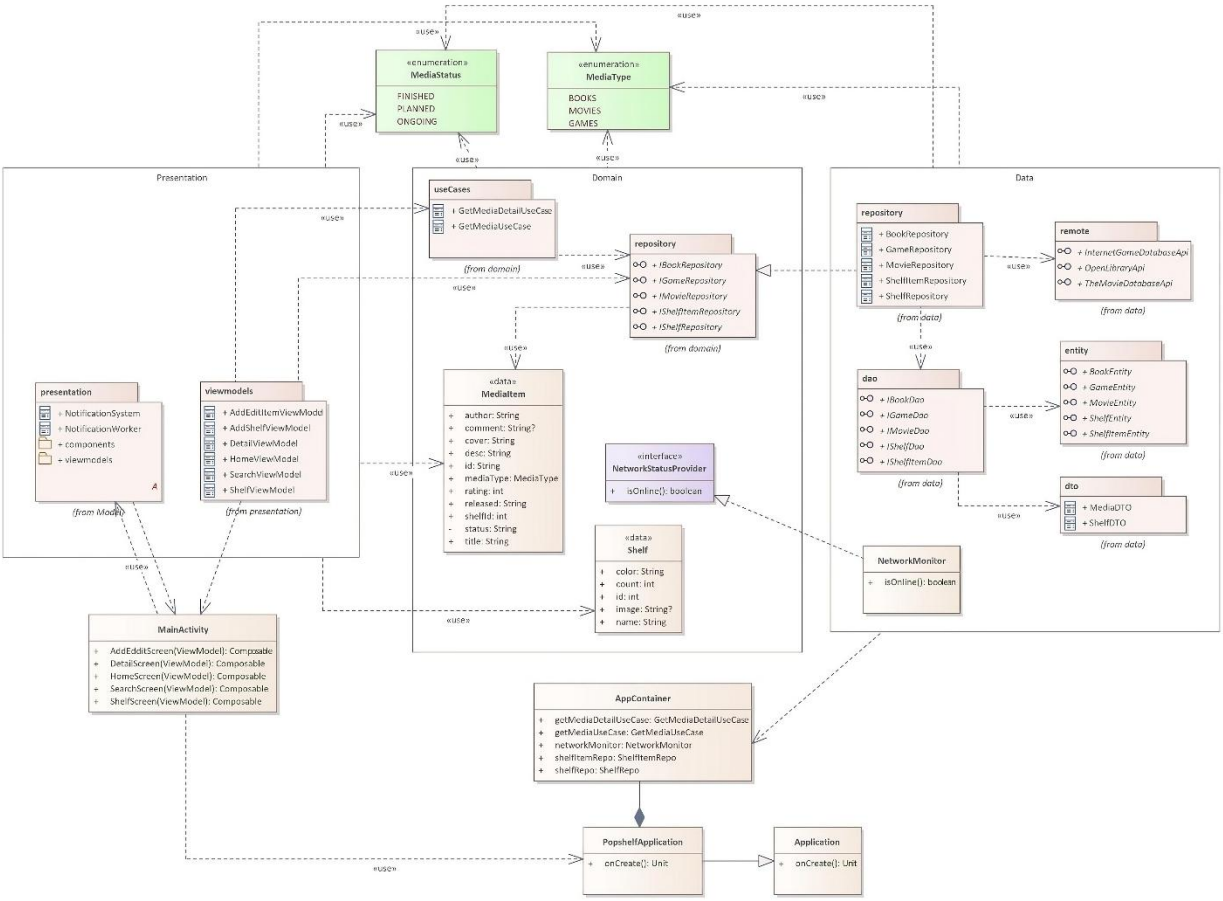


1 Diagram prípadov použitia

Návrh riešenia

Aplikácia a jej časti boli spísane a bol navrhnutý diagram tried, ktory môžeme vidieť na obrazku č. 2. Diagram tried je zámerne spracovaný v zjednodušenej/abstrakčnej podobne, kedže plne funkčná androidová aplikácia postavená na princípoch Clean architektúry, ktorú budeme používať v rámci tohto projektu obsahuje pomerne veľké množstvo tried a rozhraní, ktoré by len zhoršili jeho čitatelnosť.

Zároveň je potrebné uviesť že jazyk Kotlin poskytuje top-level deklarácie (napríklad top-level dekláracie funkcií), čo nie je typické pre objektovo orientované programovanie. Táto skutočnosť mierne komplikuje tvorbu diagramu aktivit, takže boli spravené kompromisy, aby diagram stále obsahoval všetko dôležité a dával čitateľovi logický zmysel.



2. Diagram tried - minifikovaný

Diagram aktivít, taktiež zobrazuje jednotlivé vrstvy aplikácie tak, aby aplikácia spĺňala princípy clean architektúry podľa odporúčaní Google dokumentácie. Aplikácia sa skladá z troch vrstiev a to je:

- Prezentačná (Presentation), vrstva ktorá zabezpečuje prezentovanie dát a ich zobrazenie na obrazovke používateľa a taktiež mení svoj obsah na základe interakcie užívateľa. Obsahuje napríklad obrazovky, komponenty alebo ViewModely, ktoré uchovávajú a sledujú stav obrazovky. [1]
- Doménová (Domain) vrstva sa stará tvorenie biznis logiky, ktorá sa využíva v prezentačnej vrstve konkrétne vo ViewModeloch. Uchováva napríklad dátové modely, ako v našom prípade MediaItem, nejaké useCase triedy ale hlavne rozhrania pre repozitáre. Tieto rozhrania sú neni v Google dokumentácií opísane ale v rámci riadnej Clean architektúry boli v rôznych zdrojoch nespočetne krát spomenuté ide totiž o dodržanie princípu inverzie závislosti DIP (Dependecny

inversion principle). Tento princíp v skratke hovorí o tom že moduly vyššej vrstvy by nemali závisieť od modulov nižšej vrstvy. [1, 2, 3]

- Dátová vrstva (Data) obsahuje biznis logiku, ktorá hovorí ako aplikácia ukladá, získava, ale mení dáta je zostavená z repozitárov, ktoré berú dáta z viacero dátových zdrojov, či už lokálnych alebo vzdialených. Repozitáre v tejto vrstve sú implementáciou rozhraní z dátovej vrstvy podľa princípu DIP. [1, 4]

Ako sme spomenuli diagram bol zjednodušený aby sme vedeli zobraziť každú informáciu o návrhu stručne a v rámci možnosti správne, aj keď sa to nie úplne dalo. Napríklad obrazovky aplikácie (screens) budú v aplikácií definované vo svojich individuálnych súboroch ako top-level composable funkcie. V návrhu diagramu tried sú, ale zobrazené ako metódy triedy MainActivity. MainActivity však tieto metódy len volá a spúšťa pomocou navigácie prostredníctvom NavigationGraph.

Na zhrnutie diagramu sa dá povedať že Aplikácia a jej „Entry point“ je vlastná trieda PopshelfApplication. Tá vytvorí hlavnú aktivitu, ktorá zase vytvorí podľa nás definovaný NavigationGraph pomocou NavigationHost a zobrazí HomeScreen na obrazovke. Každá obrazovka má svoj vlastný ViewModel v rámci prezentačnej vrstvy. Každý ViewModel ma prístup ku určenému repozitáru alebo UseCase funkcií z doménovej vrstvy. ViewModel, však vie len o rozhraniach týchto repozitárov nie o konkrétnych implementáciách. Repozitár dátovej vrstvy získavajú, triedia, posielajú, modifikujú a ukladajú dáta podľa potreby. Tato štruktúra spĺňa, už spomenuté princípy Clean architektúry a dovoľuje aby aplikácia bola rozšíriteľná, oddeľuje zodpovednosti medzi používateľským rozhraním, biznis logikou a dátami.

Popis Implementácie

Zložitosť aplikácie

1. Všeobecné požiadavky:

Aplikácia využíva správnu implementáciu ViewModel tried, ktoré zabezpečujú že každá obrazovka uchováva svoj stav pri každej re-kompozícií vyvolanou zmenou stavu či napríklad otočením displeja z vertikálnej do horizontálnej polohy.

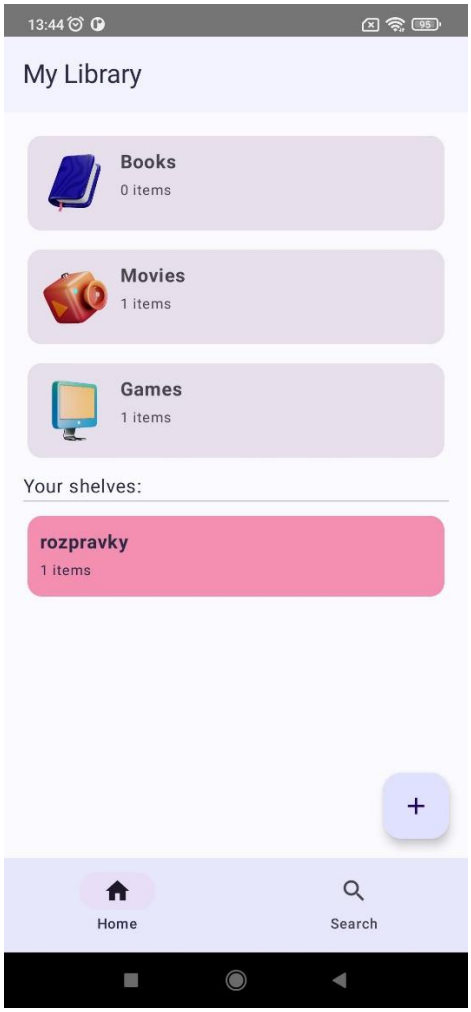
Všetky zdroje ako obrázky, text, multimédia a podobne sú implementované ako resource prostredníctvom resource managera.

2. Obrazovky

Aplikácia Popshelf pre jej plnú funkčnosť obsahuje 6 obrazoviek, ktoré môžeme vidieť na nasledujúcich obrázkoch

HomeScreen

Na nasledujúcom obrázku č.3 môžeme vidieť ako vypadá implementovaný HomeScreen, obrazovka zobrazuje všetky „poličky“ či systémové alebo aj používateľský vytvorené. V pravom dolnom rohu môžeme vidieť tlačidlo na pridanie novej poličky.



3. HomeScreen

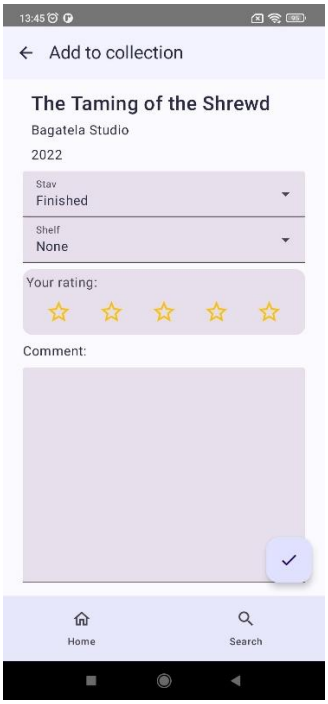
DetailScreen

Na nasledujúcom obrázku č.4 môžeme vidieť ako vypadá implementovaný DetailScreen, obrazovka zobrazuje detail herného, knižného či televízneho diela. V pravom dolnom rohu môžeme vidieť tlačidlo na editáciu záznamu ak je záznam z poličky.

4. DetailScreen

AddEditScreen

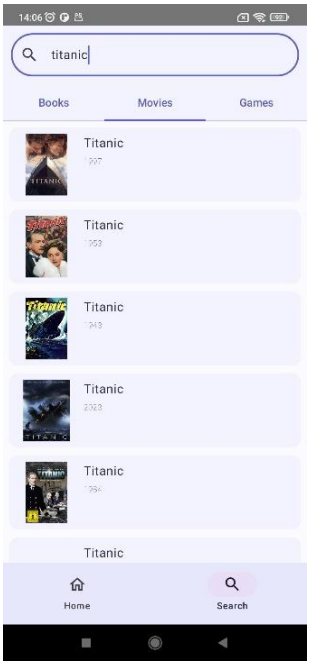
Na nasledujúcom obrázku č.5 môžeme vidieť ako vypadá implementovaný AddEditScreen, obrazovka dovoľuje užívateľovi pridať alebo editovať záznam v poličke. V pravom dolnom rohu môžeme vidieť tlačidlo na potvrdenie.



5. AddEditScreen

SearchScreen

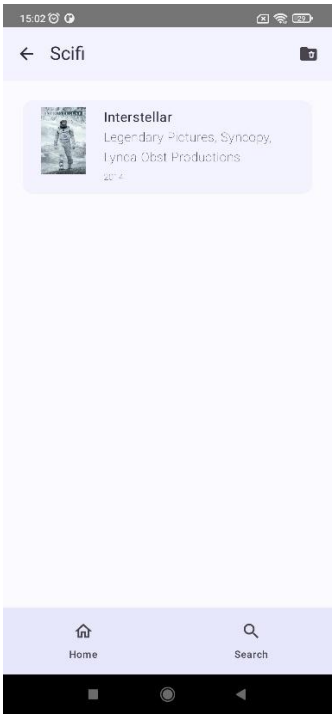
Na nasledujúcom obrázku č.6 môžeme vidieť ako vypadá implementovaný AddEditScreen, obrazovka dovoľuje užívateľovi vyhľadávať diela a filtrovať podľa typu (knihy, filmy, hry)



6. SearchScreen

ShelfScreen

Na nasledujúcom obrázku č.7 môžeme vidieť ako vypadá implementovaný ShelfScreen, obrazovka zobrazuje diela v rámci jeden „poličky“. V pravom hornom rohu sa dá vymazať cela „polička“ aj s obsahom.



7. ShelfScreen

3. Využitie AndroidX komponentov:

Aplikácia Popshelf používa množstvo komponentov medzi ktoré patria aj:

- LifeCycles – využívané najmä ViewModel triedami, už základná abstraktná trieda ViewModel od ktorej dedí každý **ViewModel** našej aplikácie je považovaný za komponent LifeCycle, tak isto sa využíva komponent **savedStateHandle**, ktorý dovoľuje aplikácii získavať parametre priamo z navigácie. Taktiež sa používa **viewModelFactory** a jej **initializer** lambda funkcia alebo **viewModelScope**, čo je vlastne coroutineScope, ktorý žije a zaniká v rámci ViewModel inštancie. No a na záver sa používa aj lambda metóda **dropUnlessResumed**, ktorá podľa LifeCycleOwnera zisťuje či je stav obrazovky aspoň v stave Resumed, zabezpečuje nám že interakcia užívateľa nie je možne mimo obrazovky. (Stáva že obrazovky už vidno nie je ale ešte jej existencia nezanikla)
- Navigation – veľmi dôkladne využívane v rámci aplikácie Popshelf. Boli použité komponenty ako **NavController**, ktorý ovláda navigáciu aplikácie, objekt je veľmi často posielaný obrazovkám ako parameter aby sa pomocou interakcie užívateľa dalo prepínať obrazovky a navigovať sa v aplikácií. **NavHost** definovaný v MainActivity triede zabezpečuje a definuje celú navigáciu aplikácie, volá composable metódy (obrazovky) podľa toho aký je stav navigácie aká obrazovka sa ma zobraziť. [5]
- Room – Je knižnica v rámci Jetpack Compose, ktorá ponúka uľahčenie práce s databázovým systémov v rámci Android aplikácie, je to abstrakčná vrstva postavená nad SQLite databázou. [6]. V aplikácií Popshelf sa používa Room na uchovávanie uloženie mediálneho obsahu či už su to knižky, hry alebo filmy a TV programy a ich užívateľské hodnotenie a podobne Tak isto sa využíva na ukladanie „poličiek“, kde si užívateľ ukladá a triedi svoj mediálny obsah.
- ViewModel – komponent, ktorý dokáže udržiavať a uchovávať stav biznis logiky alebo obrazovky. Dokáže stav zobrazovať v používateľskom rozhraní a uchovávať biznis logiku. V skratke drží svoj stav aby obrazovka nemusela získavať dáta znova pri každej navigácií či iných zmenách. [7] V aplikácií Popshelf sa snaží ViewModel používať všade kde to je možné aby bola zachovaná plna funkčnosť a jednoduchosť obrazoviek. Dovoľujú nám napríklad aby obrazovka zachovala svoj stav a dáta napríklad pri otočení displeja z vertikálnej do horizontálnej polohy. Aplikácia Popshelf sa skladá 6 ViewModel tried pre 5 obrazoviek a jeden komponent:
  - AddEditViewModel
  - AddShelfViewModel
  - DetailViewModel
  - HomeViewModel
  - SearchViewModel
  - ShelfViewModel
- WorkManager – je odporúčaný spôsob akým je možné plánovať úlohy, ktoré aplikácia môže vykonať. WorkManager dokáže vykonávať tieto úlohy aj keď je aplikácia vypnutá alebo sa mobilné zariadenia reštartovalo. Popshelf používa WorkManager prostredníctvom triedy NotificationSystem, ktorá po vytvorení jej inštancie urobí dve veci, zaregistruje Notifikačný kanál a následne nastaví že sa spustí úloha o určitý počet hodín a bude sa opakovať. Práca je definovaná triedou NotificationWorker ktorá dedí z abstraktnej triedy CoroutineWorker, táto trieda obsahuje metódu doWork, do ktorej bola implementovaná logika akou sa notifikácie vytvárajú. WorkManager zariadi sa že táto periodický vykonávaná úloha uloží do úložiska a zariadenie si bude pamätať stav a kedy sa ma práca znova vykonať. [8]

5. Notifikácie

Notifikácie sú implementované pomocou Worker a WorkManagera a vykonávajú sa každých 8 hodín pričom je náhodná šanca aká notifikácia sa používateľovi zobrazí. Aplikácia obsahuje notifikácie ktoré:

- Ponúknu užívateľovi „tip na večer“ z jeho osobnej knižnice (k čomu sa ešte nedostal alebo aby pokračoval)
- Spýtajú sa užívateľa či už niečo z osobnej knižnice dokončil a aby ohodnotil dielo.

6. Použitie externého frameworku / knižnice

V aplikácií Popshelf sme použili aj externé knižnice, akými je napríklad **Retrofit**, ktorý uľahčuje sieťovú komunikáciu a komunikáciu so zdieľanými dátovými zdrojmi, nakoľko v aplikácií používame až tri vzdialené dátové zdroje (3rd party APIs - OpenLibrary, TMDB, IGDB), ktoré posielajú v odpovediach celkom zložito štruktúrované JSON objekty, Retrofit nám umožňuje si vytvoriť jednoduché dátové triedy a jednoduché rozhrania kde sú definované cesty z ktorých dostaneme odpoveď s dátami vo forme JSON objektu, tento JSON objekt sa na mapujú pomocou Retrofitu a GSONConverterFactory do našich vytvorených dátových tried, tie následne meníme na aplikáciou typované objekty.

Knižnica Coil nám pomáha v asynchrónnom načítavaní obrázkov alebo aj animovaných obrázkov vo formáte s príponou .gif v obrazovkách, taktiež nám umožňuje načítavať obrázky pomocou sieťovej komunikácie cez URL adresu bez nutnosti ukladať obrázky v resources. Coil sa využíva na skrášlenie používateľského rozhrania a hlavne správne implementovaného zobrazovania obrázkov ku každému dielu. Knižnica robí „pod kapotou“ veľmi veľa dôležitých úkonov akými sú memory a disk caching, zmenšovanie obrázkov aby boli efektívnejšie pre aplikáciu a sieť a podobne. [9]

7. Použitie sieťovej komunikácie

Aplikácia Popshelf výrazne závisí od sieťovej komunikácie, keďže dáta nezískava z lokálnych zdrojov, ale z viacerých externých API služieb. V rámci aplikácie pristupujeme k trom webovým rozhraniam OpenLibrary, TMDB a IGDB, ktoré poskytujú rozsiahle informácie o knihách, hrach či filmoch.

Tieto API služby vracajú dáta vo formáte JSON, ktoré sa naďalej spracúvajú v Repozitároch dátovej vrstvy aplikácie. Dáta sú získavane prostredníctvom HTTPS požiadaviek, ktoré sú deklarované v jednotlivých rozhraniach. Celé získavanie dát, parsovanie JSON objektov a prácu s API je vykonávané prostredníctvom knižnice Retrofit.

Použitá programátorská technika

1. Práca s Git serverom (max. 10 bodov)

Celý projekt a práca je uchovávaná a spravovaná systémov správy verzií Git na platforme GitHub. Zobrazuje konkrétne zmeny v kóde aplikácie pri každom nahratí (commit a push). Pričom práca na projekte podľa GitHub histórie začala 2. apríla úvodným commitom a bola sústavne udržiavaná do 8.6.2025, kedy je aplikácia obhajovaná.

Git obsahuje tri vetvy, jedna vetva slúžila na implementovanie clean architektúry a následne vetva dev slúžila na implementovanie nových funkcii, ktoré nasledovne boli zlúčene do hlavnej vetvy master.

2. Návrh aplikácie, architektúra (max. 15 bodov)

Hlavným cieľom pri návrhu a implementovaní aplikácie Popshelf bolo cieľom zabezpečiť, aby boli splnene princípy oficiálneho odporúčaného Google postupu a clean architektúry. Toto zabezpečuje že aplikácia je rozdelená do vrstiev, a medzi vrstvami existujú závislosť ktorá ide z vonkajšej vrstvy do vnútornej a nikde nie naopak. Toto zabezpečuje dobrú škálovateľnosť aplikácie, udržiava teľnosť kódu a čitateľnosť.

3. Dodržiavanie zásad kódovania (coding standards).

Pri implementácii sme volili názvy súborov identifikátorov tak aby to bolo v súlade s Google odporučeniami a princípmi clean architektúry a taktiež aby jasne definovali a logicky dávalo zmysel na čo jednotlivé súčasti slúžia.

Taktiež sme sa snažili aby v kóde nenastal nejaký nežiadúci prípad či pád aplikácie a taktiež sme kód doplnili o dokumentačné komentáre nad každým verejným typom alebo štandardne komentáre slúžiace na pochopenie a vysvetlenie určitej časti kódu. Kód sme sa snažili udržiavať čistý bez duplikátov, všade kde sa dalo.

Záver

Našou úlohou bolo navrhnúť a implementovať mobilnú aplikáciu Popshelf na platformu Android. Aplikácia Popshelf ma slúžiť ako digitálne úložisko poličiek a mediálneho obsahu, pomocou ktorého si užívatelia môžu zaznamenávať aký obsah už videli a zaznačiť si hodnotenie a napísať krátky komentár, alebo si označiť obsah ktorý ich ešte len čaká alebo ho aktuálne konzumujú. Vývoj aplikácie nám dal veľa nových skúseností do vývoja mobilných aplikácií a porozumenie ako aplikácie v pozadí fungujú a čo všetko musia vykonávať aby správne fungovali.

Aplikáciu sme implementovali lokálne, pretože hodnotenie obsahu nebol zámer aplikácie, a platforiem na hodnotenie filmov, hier či kníh už existuje veľa, zámerom aplikácie bolo skôr ponúknuť používateľom fakt jedno riešenie kde si vedia zaznamenávať obsah, aplikácia sa da považovať za taký pripomienkoval aby používateľ nezabudol čo už videl, hral alebo čítal alebo čo len chystá čítať, hrať či pozerať. V minulosti sme si chceli zaznamenávať hry, ktoré si ešte chcem zahrať a riešili sme to rôznymi poznámky v mobile či počítači a nebolo to ono, keďže sme nenašli univerzálne riešenie pre každú typ mediálneho obsahu, rozhodli sme sa že by mohla vzniknúť aplikácia, ktorá združí všetky tieto typy dokopy a ponúkne funkcie ako Popshelf ponúka.

Aplikácia je plne funkčne na používanie, načítava dáta o rôznom mediálnom obsahu z viacerých databázových zdrojov a je implementovaná lokálne, dokonca funguje aj v off-line režime, vyhľadávanie je v off-line režime obmedzené len o obsah, ktorý už je zálohovaný a bol v minulosti vyhľadávaný. Aplikácia však obsahuje aj nedokonalosti, akými sú zväčša nestihnutie implementovania funkcií, akými je napríklad widget miniaplikácia, synchronizácia údajov v aplikácií medzi mobilmi či pokročilé filtrovanie obsahu podľa autora, kategórií a podobne. Tieto nedokonalosti sú spôsobene časovou náročnosťou programovania pre platformu Android a dodržanie čistej architektúry (Clean Architecture).

Zoznam zdrojov

[1] Google, „Guide to app architecture,“ 2025. [Online]. Available: https://developer.android.com/topic/architecture.

[2] A. Picón, „Demystifying Clean Architecture,“ 2022. [Online]. Available: https://devpicon.medium.com/demystifying-clean-architecture-1cf744a3692e. [Cit. 2025].

[3] M. Mkhoian, „App Architecture: Domain layer,“ 2024. [Online]. Available: https://proandroiddev.com/app-architecture-domain-layer-b9f6aa839e33. [Cit. 2025].

[4] M. Mkhoian, „App Architecture: Data layer,“ 2024. [Online]. Available: https://www.droidcon.com/2024/05/16/app-architecture-data-layer/. [Cit. 2025].

[5] Google, „Navigation,“ Google, 2025. [Online]. Available: https://developer.android.com/guide/navigation. [Cit. 2025].

[6] Google, „Save data in a local database using Room,“ Google, 2025. [Online]. Available: https://developer.android.com/training/data-storage/room#setup. [Cit. 2025].

[7] Google, „ViewModel overview,“ Google, 2025. [Online]. Available: https://developer.android.com/topic/libraries/architecture/viewmodel. [Cit. 2025].

[8] Google, „Schedule tasks with WorkManager,“ Google, 2025. [Online]. Available: https://developer.android.com/topic/libraries/architecture/workmanager. [Cit. 2025].

[9] Coil Contributors , „Overview,“ 2025. [Online]. Available: https://coil-kt.github.io/coil/. [Cit. 2025].

[10] Thorben, „SOLID Design Principles Explained: Dependency Inversion Principle with Code Examples,“ Stackify, 2023. [Online]. Available: https://stackify.com/dependency-inversion-principle/. [Cit. 2025].