Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Mobilní aplikace pro návštěvníky a obyvatele obcí ČR pro platformu Android

Plzeň 2020 Pavel Habžanský

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 6. května 2020

Pavel Habžanský

Abstract

The goal of this thesis if analysis of needs and requirements of citizens and visitors to Czech towns and cities and delivering relevant information to them. The application is developed for Android platform, the thesis describes basics of Android development. During implementation we will be taking into consideration existing solutions and their user's criticisms.

Abstrakt

Cílem této práce je analýza potřeb obyvatel a návštěvníků českých obcí a implementace mobilní aplikace pro šíření relevantních informací mezi obyvatele a návštěvníky českých obcí. Aplikace je implementována pro platformu Android, práce obsahuje popis základů vývoje pro tuto platformu. Při implementaci funkcí jsou brány v potaz existující řešení a jejich kritika z pohledu uživatelů.

Obsah

1 Úvod

S rozšiřujícím se množstvím mobilních zařízení a jejich funkcí se rozšiřují i potřeby jejich uživatelů a možnosti implementace nových aplikací, které uživatelské potřeby řeší. Mezi tyto funkce patří i šíření informací mezi uživatele aplikace a jeho aktivního zapojení do sdílení těchto infrmací.

V první části se práce zabývá analýzou dostupných řešení a jejich hodnocení od uživatelů. Na základě těchto informací je možné určit hlavní funkce těchto aplikací, jejich důležitost a rozdělení uživatelů do dvou skupin - obyvatelé obcí a návštěvníci.

Na základě předešlých analýz je navržena sada důležitých funkcí včetně jejich uživatelského rozhraní a chování.

Práce popisuje základní fungování platformy Android, verze, historii a vlastnosti. Věnuje se především popisu rozšířeným architektonickým postupům, návrhu a vývoji aplikací, které porovnává s ohledem na jejich vlastnosti.

Aplikace je následně realizována pro platformu Android za použití vybraných architektonických postupů popsaných v předešlých kapitolách. Dále je v kapitole popsán účel používaných datových struktur a práce s nimi.

Následuje popis možností a způsobů testování mobilních aplikací. Funkčnost výsledné aplikace byla otestována pomocí jednotkových, instrumentálních a uživatelských testů za pomoci testovacích scénářů.

Práce se nakonec zabývá možnými rozšířeními pro vytvořenou aplikaci.

2 Analýza stávajících řešení

Na trhu je dostupná řada mobilních aplikací pro občany nebo návštěvníky jednotlivých měst. Často se jedná o aplikace, které si nechaly vyvinout jednotlivé obce na vlastní náklady. Pro analýzu klíčových funkcionalit a pochopení potřeb uživatelů bylo vybráno několik aplikací, které byly analyzovány z hlediska jejich funkcionalit a ohodnocení uživatelů. Následující aplikace byly vybrány na základě vyhledání pomocí klíčových slov v Google Play jako „občan“, „hlášení“ nebo názvy českých metropolitních měst, v tomto případě „Plzeň“, „Praha“ a „Brno“.

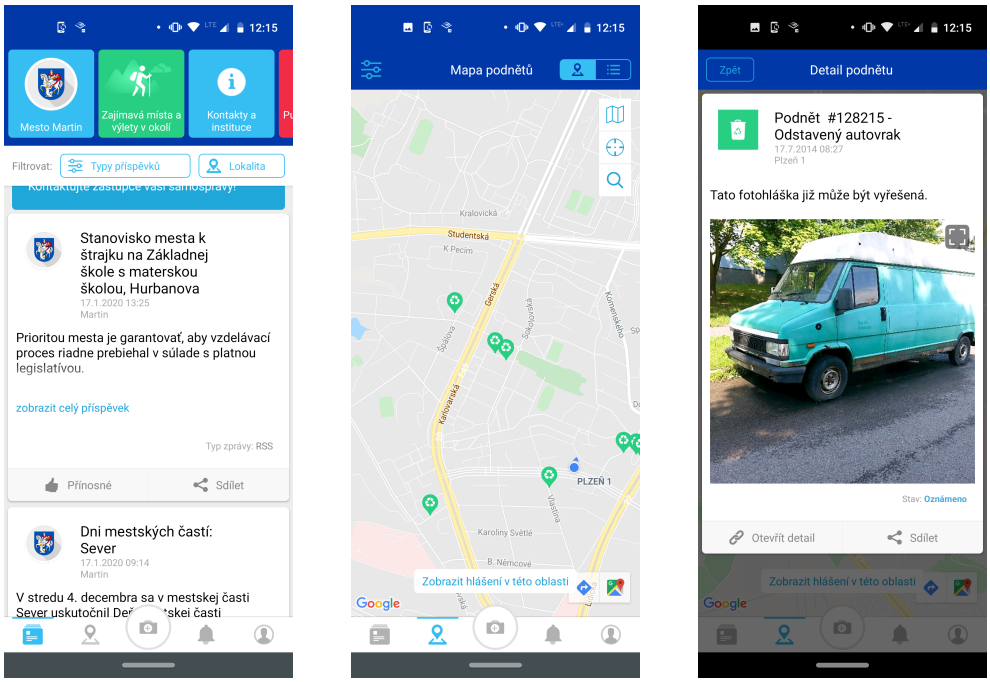
2.1 Mobilní rozhlas

Mobilní rozhlas je z hlediska počtu stažení a hodnocení největší aplikací pro chytrou komunikaci zapojující občany do dění měst a obcí. Je vyvíjena společností Neogenia s.r.o. Aplikace je pravidelně aktualizována, v době psaní této práce získala poslední aktualizaci 26.3. 2020. Má celkové hodnocení 4,3/5 hvězd na základě 641 recenzí a přes 50 tisíc stažení na Google Play. Analýza je prováděna na verzi 3.2.0.

Aplikace umožňuje uživatelům zadávat různorodé podněty (nefungující, osvětlení, výmoly na silnici, černé skládky, pohřešované osoby, ztracená zvířata apod.), o změně stavu podnětů umí aplikace uživatele notifikovat. Tato funkcionalita je dostupná po celé republice nezávisle na lokaci uživatele nebo vybrané obci. Aplikace zobrazuje novinky v obci, nabízí návrhy na výlety, zajímavá místa a kontakty na instituce v obci, nezobrazuje však například kulturní akce nebo zajímavá místa v nejbližším okolí uživatele, pouze v dané obci. Novinky jsou čerpány z RSS zdrojů. Tyto funkcionality však vyžadují aktivní zapojení obce do infrastruktury aplikace Mobilní rozhlas. Obec se musí registrovat na straně Mobilního rozhlasu a skrze jeho rozhraní může přidávat své příspěvky.

Aplikace nabízí přehledné nastavení pro uživatele v rámci filtrování hlášení, které může vidět na mapě nebo zobrazovaý typ mapy.

Aplikace je vhodná jak pro obyvatele měst, tak pro náhodné návštěvníky. Její design není příliš intuitivní a chvíli trvá, než se uživatel v aplikaci zorientuje. Zobrazení hlášení by mohlo být implementováno z technického hlediska lépe, uživatel si musí ručně klikat na dedikovaný text pro aktualizaci hlášení na mapě. Mezi nejčastější stížnosti uživatelů patří pády aplikace, případně chybové hlášení, a časté nevyžádané emaily a SMS zprávy.

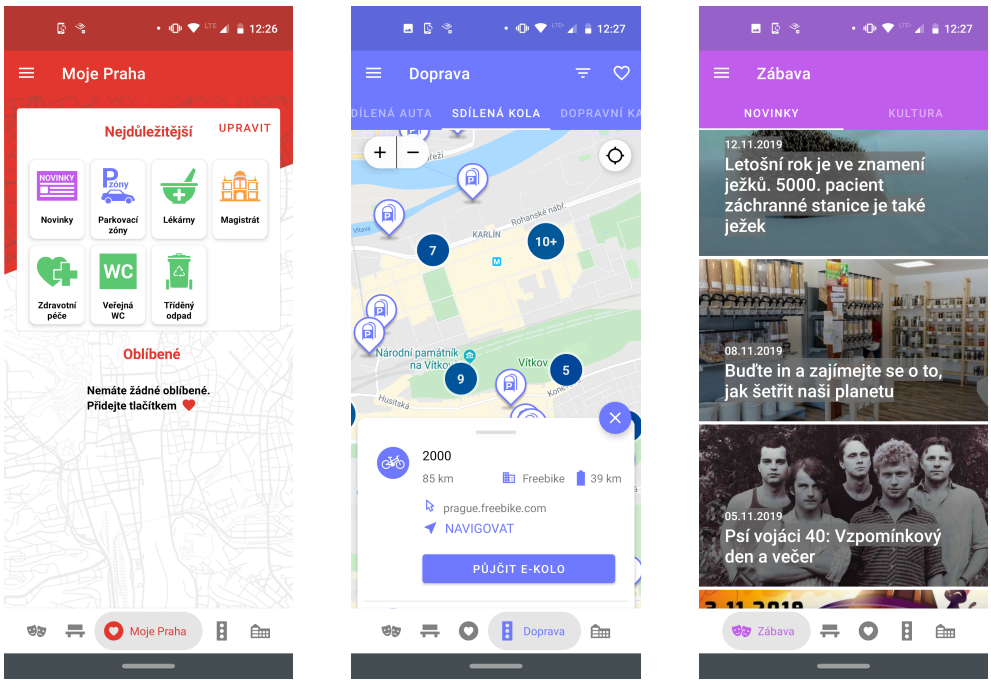


2.2 Moje Praha

Aplikace Moje Praha je vyvinuta společností Operátor ICT a.s., v současné době ve verzi 3.1.2, je pravidelně aktualizována, v době psaní této práce dostala poslední aktualizaci 10.3. 2020. Na Google Play má hodnocení 2,9/5 hvězd na základě 610 recenzí a přes 50 tisíc stažení.

Aplikace Moje Praha cílí na obyvatele a návštěvníky Prahy, zaměřuje se především na sdílení novinek a nabídku služeb dostupných v obci. Aplikace nenabízí vytváření podnětů uživatelem. Sdílení informací dělí mezi kategorie "Novinky"a "Kultura", návštěvník Prahy tak může najít zajímavé aktivity na jednom místě. Pro obyvatele Prahy nabízí aplikace Moje Praha informace o službách jako městské knihovny, parky, zdravotní péče, parkovací zóny a dokonce i přístup k náhledu z dopravních kamer. Velmi dobře technicky vyřešená je mapa, kde se položky shlukují při oddálení mapy. Aplikace je přehledná s minimalistickým designem.

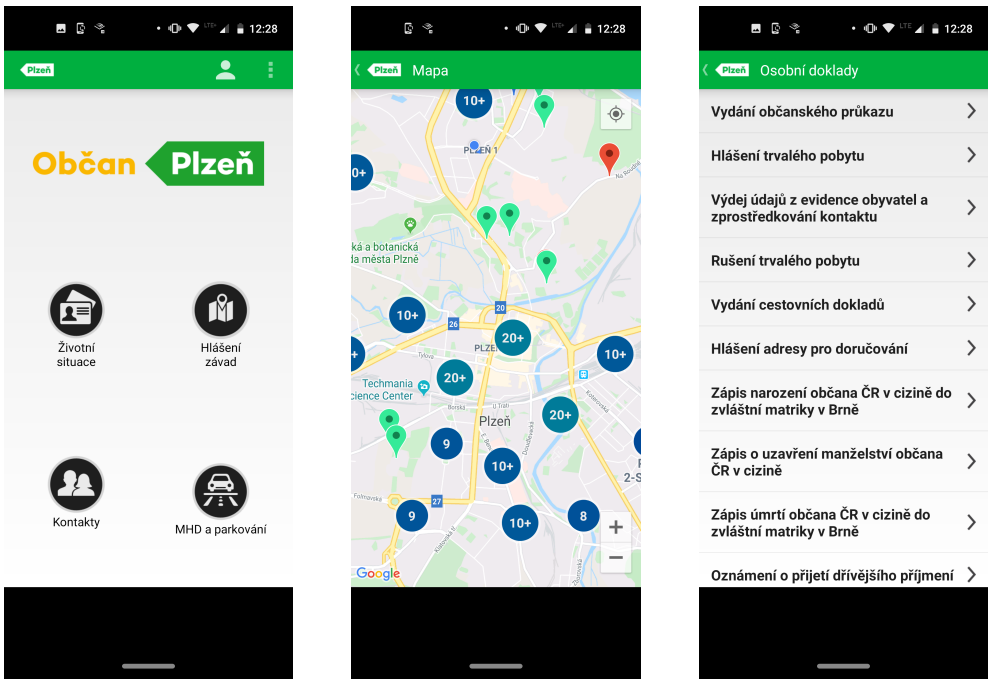
Aplikace má své nedostatky především v oblasti "Novinek", načítání se často zasekávalo a nenačítaly se obrázky, které by měly k novinkám patřit. Stejně na tom byla kategorie "Kultura". Uživatelské nastavení je poměrně omezené, položky na mapě nelze filtrovat podle typu, jen otevřené/zavřené. Výjimkou jsou parkovací zóny. Zřejmě nejčastější stížností uživatelů na Google Play je neaktuálnost informací poskytovaných aplikací, především pak informace vztahující se k parkovacím zónám.



2.3 Plzeň pro občany

Aplikace Plzeň pro občany cílí předně na obyvatele Plzně, zaměřuje se na nabídku služeb a informací, o které budou mít zájem exkluzivně obyvatelé Plzně. Aplikace nabízí jednoduchý systém pro zadávání podnětů a nedostatků v obci uživatelem. Aplikace obsahuje kontakty na plzeňské úřady, photovost, policii, školy a další obecní služby, přehled parkování ve městě, mapku MHD apod. Zajímavostí aplikace je sekce "Životní situace", která slouží jako menší poradna pro časté dotazy obyvatel, například "Jak zrušit trvalý pobyt?", "Kde mi bude vydán řidičský průkaz?"apod.

Oproti ostatním aplikacím má aplikace Plzeň pro občany atypický design, co se týče aplikací pro Android. Aplikace je přehledná, jednoduchá na ovládání, cílovou skupinou jsou však exkluzivně trvalí obyvatelé Plzně, pro návětěvníky Plzně neobsahuje aplikace příls zajímavé funkcionality.



Obrázek 2.1: Ukázky aplikace Plzeň pro občany

2.4 Plzni.to

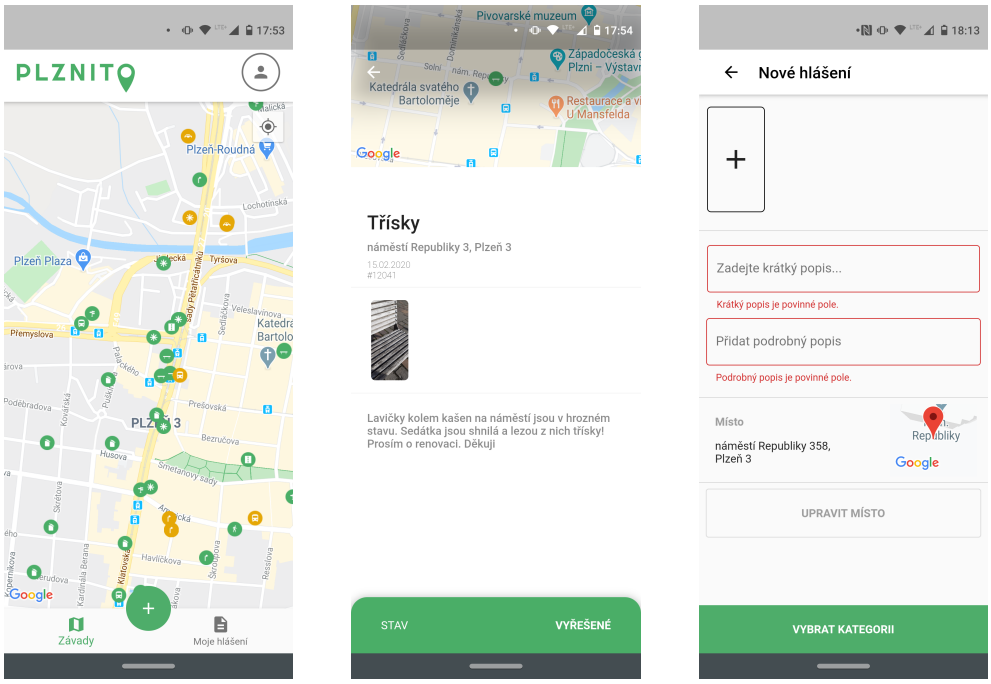
V průběhu práce na této bakalářské práci došlo ke zrušení aplikace Plzeň pro občany. Místo ní je pro Plzeň nyní k dostání aplikace Plzni.to, aplikace tak prošla výraznou změnou ve svém designu i funkcionalitách. Aplikace Plzni.to je vyvíjena Správou informačních technologií města Plzně, současná verze je 3.0.2 a poslední aktualizace byla 21.2. 2020. V současné době je hodnocena 3,9/5 hvězdami na základě 106 recenzí a přes 5 tisíc stažení.

Aplikace nyní dává uživateli možnost zadávat hlášení z různých kategorií, např. veřejné osvětlení, lavičky, černé skládky, zastávky MHD a mnoho dalších. K těmto hlášením uživatel může přidat fotografii. Nutně dané hlášení musí být zadané v oblasti Plzně, mimo oblast Plzně zadat nelze. K aplikaci přibyla možnost registrace a přihlášení, uživatel tak může pohodlně sledovat svá hlášení a jejich stav.

Zároveň z aplikace zmizely všechny informativní položky, které původně obsahovala aplikace Plzeň pro občany, jako Q&A sekce, kontakty na různé úřady a mapy parkování a MHD.

Aplikace je poměrně mladá a má své nedostatky, nelze se například registrovat pomocí Facebook účtu nebo Google účtu, naopak je při registraci uživatel směrován na webovou stránku, což nemusí být pro uživatele na mo-

bilním zařízení příliš pohodlné. Mapa má také své nedostatky, větší množství objektů jí dělá nepřehlednou, o to více chybí funkcionality filtrování na mapě. Bohužel celkový počet funkcionalit je menší než u původní aplikace Plzeň pro občany. Uživatelé si nejvíce stěžují na absenci jejich oblíbených funkcionalit ze staré verze a nelogické pořadí kroků při zadávání hlášení - nejdříve musí být vyplněn popis, pak je teprve vůbec možné vybrat kategorii.



Obrázek 2.2: Ukázky aplikace Plznito

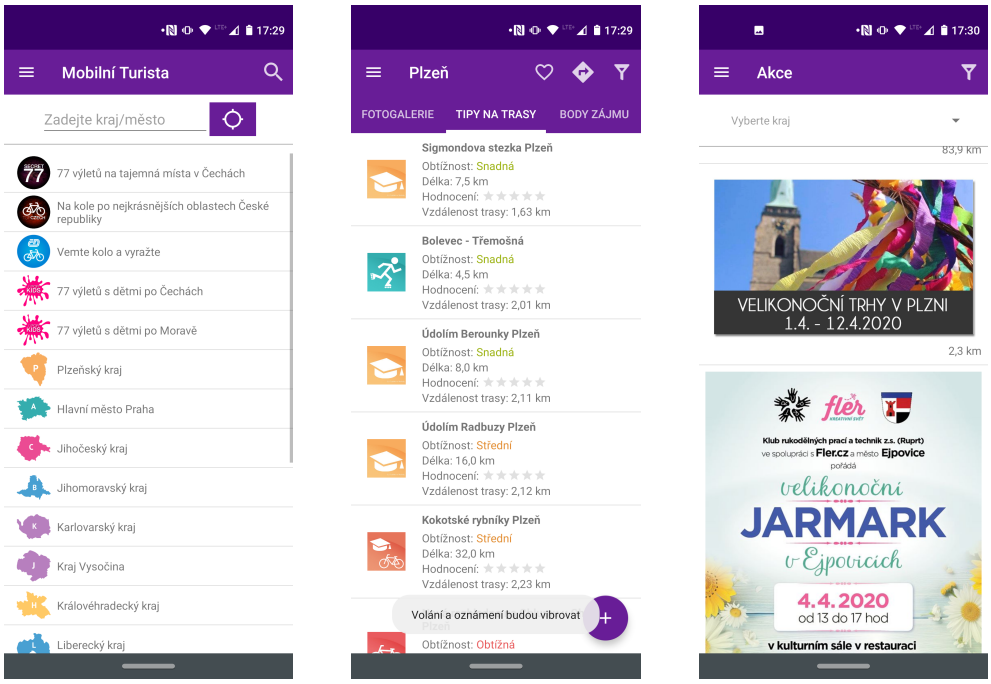
2.5 Mobilní turista

Aplikace Mobilní Turista je vyvíjena společností GOWAY s.r.o., v současné době funguje ve verzi 1.3.8, poslední aktualizace proběha 3.3. 2020, aplikace je tedy stále vyvíjena a rozšiřována. Aplikace je kladně hodnocena 4,1/5 hvězd na Google Play na základě více než tisíce rezencí a přes sto tisíc stažení.

Aplikace cílí exkluzivně na turisty a jejich potřeby na českém území. Obsahuje funkcionality jako kalendář kulturních akcí, seznam cílů pro turisty, trasy, body zájmu apod. Aplikace umí trasovat historii míst navštívených uživatelem. Všechny tyto vyhledávací funkce je možné filtrovat podle konkrétního kraje a okresů. Zajímavou funkcionalitou je také propojení s Google kalendářem mobilního telefonu, do kterého je možné si poznamenávat kul-

turní akce a jednoduše nastavit připomínky.

Hlavním nedostatkem této aplikace v rámci naší analýzy je zaměření specificky na jednu danou věc, kterou jsou kulturní akce a průvodce po zajímavých místech. Existuje však mnoho recenzí na Google Play zmiňující nepřehlednost aplikace, neaktuálnost a omezené množství zobrazovaných informací, především pak tras a bodů zájmu.



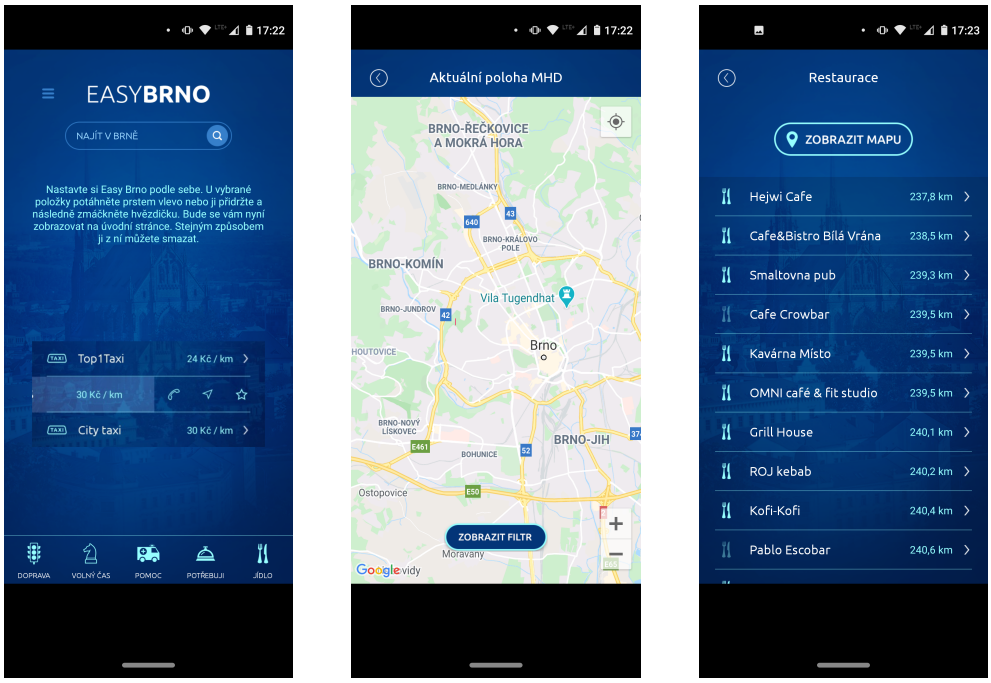
Obrázek 2.3: Ukázky aplikace Mobilní turista

2.6 Easy Brno

Aplikace Easy Brno je zřejmě nejstarší z analyzovaných aplikací.Je vyvinuta společností X Production s.r.o., v současné době je dostupná ve verzi 2.0.3 a poslední aktualizace aplikace proběhla 2.4. 2017. Je hodnocená 3,7/5 hvězdami na základě 432 rencenzí a více než 10 tisíc stažení.

Jak již název napovídá, aplikace cílí na potřeby obyvatel a návštěvníky města Brno. Uživatelům nabízí mnoho funkcí a informací v oblastech dopravy, volného času, jídla nebo se snaží vyhovět dalším potřebám uživatelům, například umožňuje sledování MHD, informace o taxi službách, muzea, divadla, restaurace nebo kontakty na úřady, lékáře a jiné záchranné služby.

Největší nevýhodou této aplikace je její stáří, což je velmi znát na jejím designu, který ani není plně kompatibilní s velikostí moderních zařízení. Může být tedy i zpochybněna aktualita informací, které aplikace nabízí. To je reflektováno i v recenzích na Google Play.



Obrázek 2.4: Ukázky aplikace Easy Brno

2.7 Zhodnocení funkcí

Na základě analýzy zmíněných aplikací byly definovány 4 kategorie funkcí, které dané aplikace zahrnují:

- Informace - informace dodávané aplikací jako novinky v obcích a notifikace
- Hlášení podnětů - hlášení podnětů od užitelů jako upozornění na závady v okolí
- Kultura - informace o kulturních příležitostech a akcích
- Místa - zajímavá místa v okolí jako např. parky, parkoviště, bary, restaurace, knihovny apod.

Výsledek zhodnocení zmíněných aplikací je znázorněn v následující tabulce. Bodování probíhá na škále 1-3, kde 3 je nejvyšší hodnocení. Zhodnocení dané funkcionality je dáno prostudováním uživatelských rezencí na Google Play. Úplná absence funkcionality je značena pomlčkami.

Aplikace	Informace	Hlášení podnětů	Kultura	Místa
Mobilní rozhlas	3	2	–	–
Moje Praha	1	–	1	1
Plzeň pro občany	3	2	–	3
Plzni.to	–	1	–	–
Mobilní turista	1	–	2	1
Easy Brno	1	–	–	1

V rámci *Mobilního rozhlasu* byla většinou uživatelů velmi kladně hodnocena informovanost uživatelů o novinkách v jejich okolí. V rámci hlášení podnětů si někteří uživatelé opět chválili informovanost, někteří si však stěžovali, že hlášení nejsou aktuální nebo jejich zadávní nefunguje příliš dobře. V rámci funkcionalit spojených s kulturou a zajímavostmi v okolí aplikace nic nenabízí. Uživatelé si na uživatelské rozhraní nestěžovali, ani jej nechválili.

V aplikaci *Moje Praha* si uživatelé často stěžovali na neaktuálnost informací, stejný problém měli v případě kulturních událostí a zajímavostí v okolí. Aplikace nenabízí možnost zadávat hlášení, rozhraní aplikace nebylo uživateli nijak zvlášť hodnoceno.

Aplikace *Plzeň po občany* v současné době již není ke stažení a její hodnocení je možné dohledat ve starých recenzích aplikace *Plzni.to* (přibližně v první polovině roku 2019 ještě šlo o aplikaci *Plzeň pro občany*). Uživatelé tehdy kladně hodnotili kvalitu informací, které aplikace poskytovala, navigaci na relevantní místa a její jednoduchost. Stejně tak byla chválena funkcionalita hlášení podnětů, nicméně vyskytovaly se případy, kdy zadávání hlášení nefungovalo správně nebo něčí nahlášený podnět mizel.

Aplikace *Plzni.to* nebyla příliš kladně hodnocená, protože jde o velmi ochuzenou verzi aplikace *Plzeň pro občany* s aktualizovaným UI, na které podle recenzí uživatelé nebyli zvyklí a hodnotili jej zpravila negativně. Uživatelé si stěžovali především na ztrátu možnosti získání kvalitních informací a zhoršení kvality hlášení podnětů.

Aplikace *Mobilní turista* se zaměřuje především na informace ohledně kulturních událostí a trávení volného času. Uživatelé si často stěžovali na neaktuálnost poskytovaných informací a nepřehlednost aplikace.

Aplikace *Easy Brno* byla v recenzích silně kritizována pro své zastaralé UI a špatnou kvalitu poskytovaných informací.

2.8 Důležitost funkcí

Na základě vybraných recenzí uživatelů a hodnocených funkcí jejich recenzí byla sestavena tabulka reprezentující důležitost jednotlivých funkcionalit. Důležitost funkcionality je zvýšena v případě, že si uživatelé stěžovali na její kvalitu, absenci nebo chválili její zahrnutí v aplikaci. Naopak nižší důležitosti funkce dosáhne, pokud se k ní uživatelé nijak nevyjadřovali nebo ji považovali za zbytečnou.

Funkcionalita	Důležitost
Informovanost	3
Hlášení podnětů	2
Kultura	2
Místa	1

Uživatelé si velmi často stěžovali na neaktuálnost informací nabízených v aplikacích, naopak tuto skutečnost oceňovali, když byla dobře naimplementována. Zajímavá místa byla v hodnoceních uživatelů zmiňována zdaleka nejméně, proto má nejmenší důležitost.

2.9 Zhodnocení potřeb uživatelů

Na základě předešlých analýz je možné definovat potřeby uživatelů, respektive skupin obyvatel obcí a turistů nebo návštěvníků obcí.

Obyvatelé se dlouhodobě zdržují v okolí jedné obce, proto vyžadují vyžadují aktuálnost dostupných informací poskytovaných aplikací. Jde tak o novinky z jejich obce a nejbližšího okolí nebo notifikace na konkrétní dění v jejich obci. Dále hlášení podnětů v jejich obci, ke kterým mohou sami přispívat.

Náštěvníkům obcí není okolí dané obce známé, proto spoléhají na aktuálnost dostupných informací z jejich okolí, které je ale nesvazují s danou obcí. To mohou být například zajímavá místa v okolí nebo kulturní akce, které mohou navštívit použitím navigace. Návštěvníci a turisté nemusí být z české republiky, proto je nutné implementovat lokalizaci aplikace do angličtiny. Tato funkce však spadá do nastavení aplikace, které je společné pro obě skupiny uživatelů.

3 Návrh funkcí aplikace

Na základě analýzy uživatelských potřeb byly navrženy funkce popsané v následujících sekcích . Funkce jsou navrženy tak, aby plnily potřeby obou skupin uživatelů.

3.1 Novinky

Informovanost obyvatel obce byla během analýzy vysoce ceněná funkcionalita, tato funkce byla užívána v rámci sdílení novinek v obcích a okolí užívatelů. Kriticky se uživatelé vyjadřovali především k aktuálnosti informací, které dané aplikace nabízely. Kvalita této funkcionalita je tedy dána především kvalitou zdroje, ze kterého jsou data čerpána.

Funkcionalita bude implementována pro obě skupiny uživatelů, kdy bude možné si prohlížet aktuality v obci pomocí detailu obce v aplikaci nebo nastavením obce jako „Moje město“.

Kvalita a užitečnost funkce se odvíjí od kvality zdrojů informací, je tedy zásadní jejich volba. Jako zdroje novinek mohou být použity například Facebook stránky obcí nebo jejich RSS zdroje událostí. Ke vzdálenému přístupu k příspěvkům Facebook stránek je však zapotřebí souhlas ze strany administrátora stránky nebo schválení přistupující aplikace ze strany Facebook. Nicméně existuje možnost využít služby *https://rss.app/rss-feed/create-facebook-rss-feed*, která je schopná generovat RSS z Facebook stránek. Z důvodu aktuálnosti informací bude tato funkcionalita vyvíjena čtením RSS zdrojů obcí.

3.2 Inbox

Pro potřeby aktivní komunikace směrem k obyvatelům obce je navržena funkce inbox, který umožňuje v reálném čase informovat obyvatele města o aktuálních událostech v jejich obci. Příklad takových zpráv může být rušení akcí v obci, odkazy na nové vyhlášky apod.

Nastavením obce jako „Moje město“ se uživatel zároveň přihlásí k odběru push notifikací dedikovaných k danému městu. Uživateli pak budou přicházet notifikace, které bude aplikace persistovat a zobrazovat uživateli v podobě zpráv v sekci „Aktuality“. Přečtené zprávy bude moci uživatel sám mazat.

3.3 Kultura

Implementace funkce pro prohlížení kulturních akcí je klíčová pro uživatele ze skupiny návštěvníků obcí. V rámci analýzy této funkce byly analyzovány služby zaměřující se na seznamy kulturních akcí - www.kudyznudy.cz, www.vyletnik.cz, www.goout.net. Jako zdroj těchto akcí bude použito veřejně dostupné API společnosti GoOut, které o akcích nabízí podrobné informace jako datum a čas konání, místo, stát, ceny vstupenek nebo kategorie. Informace o kulturních akcích jako jejich kategorie a místo konání budou použity pro jejich filtrování. Tato služba byla použita z důvodu jednouché čitelnosti API a možnosti stránkování.

3.4 Mapa

Mapa tvoří základ mnoha ostatních funkcionalit, primárně pak zobrazování nahlášených podnětů a zajímavých míst v okolí. Implementace mapy je podmíněná registrováním vyvíjené aplikace na *console.developers.google.com* a odemknutím balíčku *Maps SDK for Android*. Pro plnou funkci mapy a jejích funkcí je třeba sledovat polohu uživatele, například načtení mapy na jeho aktuální poloze pro jednodušší orientaci. Znalost aktuální polohy uživatele může být využita pro stahování relevantních dat vzhledem k poloze uživatele nebo navigaci.

3.5 Výběr obcí

V aplikaci bude implementován vyhledávač obcí s našeptávačem zobrazující všechny dostupné obce a historií vyhledávání pro jednodušší navigování na dříve navštívené položky. Jelikož nebylo možné nalézt žádný obecný zdroj dat poskytující informace o českých obcích, bude zdroj dat pro správu záznamů o obcí v rámci práce vytvořen formou cloudového úložiště, ke kterému bude mobilní aplikace přistupovat. Nezávislost na cizím zdroji dále zajistí spolehlivou definici protokolu komunikace mezi mobilní aplikací a vzdáleným úložištěm.

3.6 Detail obce

Uživatel si bude moci vybrat obec a prohlédnout si její detail, který bude obsahovat základní informace o obci včetně jejího znaku, popisu a fotogalerie zajímavých míst v obci. Z detailu bude dále vést navigace na webové

stránky obce, zobrazení polohy na mapě, novinky a možnost nastavit obec jako „Moje město“.

V detailu obce bude možnost ji nastavit jako „Moje město“, to znamená, že uživateli budou zobrazovány novinky z obce přímo na titulní obrazovce aplikace.

3.7 Hlášení podnětů

Možnost hlášení podnětů je označována za funkcionalitu se střední důležitostí. Nepřináší příliš velkou přidanou hodnotu pro návštěvníky obcí. Nimcéně obyvatelé obce by měli mít možnost hlásit podněty pro zlepšení situace v jejich okolí. Hlášení podnětu probíhá zadáním titulku, popisu, kategorií hlášení a přidáním ilustračni fotografie. Tato funkcionalita bude tedy oddělená od funkcionalit vztahujících se k návštěvníkům obce, pro které by byla spíše rušivá.

Zadaní hlášení budou viditelná na mapě pro uživatele obce a rozlišené podle kategorií. V detailu hlášení budou zobrazeny základní informace jako nadpis, popis, datum zadání a přidaná fotografie.

Kategorie hlášení podnětů

Uživateli jsou nabídnuty různorodé kategorie, ve kterých bude moci zadávat hlášení závad v obci. Jako kandidáti na tyto kategorie jsou zvoleny

- Poškození veřejného majetku
- Ztráta věci
- Černá skládka
- Jiné

Kategorie podnětů slouží pro snadnou orientaci mezi podněty a možnost implementace jejich filtrování.

3.8 Zajímavá místa

Obdobně jako obyvatele obcí budou zajímat hlášení a stížnosti z dané obce, pak turisty a návštěvníky obce budou zajímat primárně zajímavá místa v jeho okolí, kde může trávit svůj čas, případně najít informace, které mu dále pomohou s orientací.

Uživatele zajímá detail místa, jehož součástí je například název, adresa, informace o otevírací době a fotografie, pokud je dostupná.

Kategorie míst

Uživatelům jsou nabídnuty různorodé kategorie míst, která by mohl navštívit. Ideálními kandidáty na tyto kategorie jsou

- Občerstvení - bar, kavárna
- Příroda - park
- Doprava - zastávky MHD, parkování
- Kultura - muzeum, kostely
- Historické památky - turistické atrakce

Kategorie míst slouží pro snadnou orientaci mezi místy a možnost implementace jejich filtrování.

3.8.1 Navigace

Implementace funkcionality pro navigaci je vhodná pro uživatele ze skupiny turistů a návštěvníků obce, kteří by potenciálně chtěli navštívit nějaké zajímavé místo v obci. Navigační služby fungující uvnitř aplikace jsou zpravidla placené, nicméně zde existuje možnost využít aplikací třetích stran, například Waze nebo aplikaci Google Mapy, které jsou standardně součástí všech verzí OS Android.

3.9 Přizpůsobení aplikace

Z analýzy vybraných aplikací vyplývá, že se některé položky na mapách budou vyskytovat ve větším množství, což má negativní vliv na výkonnost aplikace, výdrž baterie zařízení a uživatelský zážitek. Některé položky navíc nemusí být pro uživatele aplikac relevantní. Z toho důvodu bude implementována funkcionalita pro filtrování položek zobrazovaných na mapě. Zároveň bude implementováno shlukování položek na mapě do shluků v případě jejich hustého výskytu.

Pro návštěvníky obcí bude implementována možnost filtrování kulturních událostí na základě místa jejich konání nebo kategorie.

Jelikož aplikace není cílená na české uživatele, bude implementována i možnost přepínat jazyk aplikace mezi českým a anglickým jazykem. Tato možnost absolutně chyběla u všech analyzovaných aplikací.

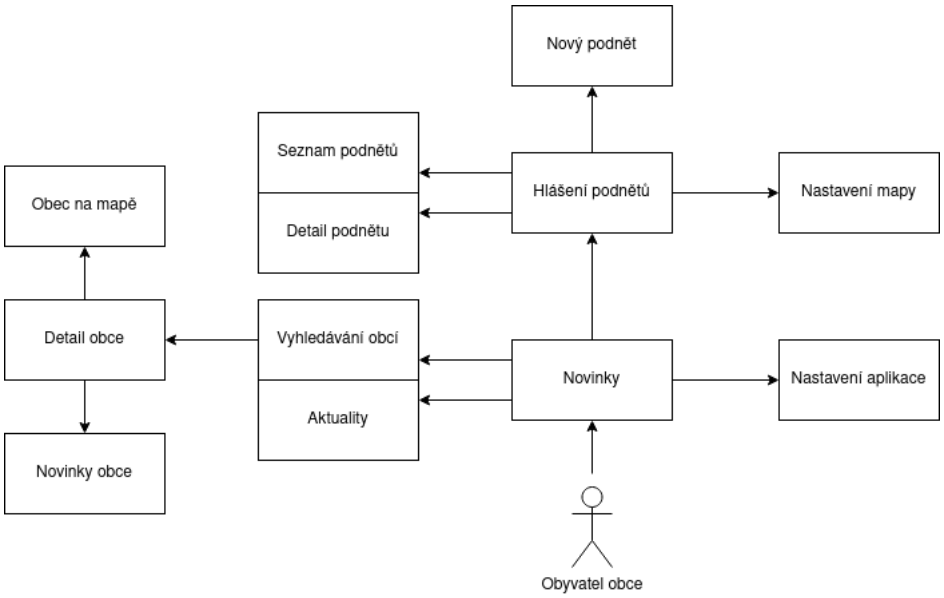
3.10 Návrh navigace aplikací

Na základě vybraných funkcí byl navržen navigační graf s důrazem na dostupnost jednotlivých funkcí aplikace a s ohledem na specifické potřeby definovaných tříd uživatelů.

3.10.1 Navigace pro obyvatele

Navigace obyvatele obce začíná na úvodní obrazovce, kde jsou pro uživatele relevantní aktuální události v obci, ze které informace odebírá. Navigace jej vede na mapu, kde je možné si prohlížet a zadávat vlastní podněty.

Z úvodní obrazovky se dále může dostat na vyhledávání obcí, kde si může obec najít a nastavit jako „Moje město“ a sledovat novinky v dané obci. Pak bude dostávat aktuality z obce, které si bude moci prohlédnout ve svém inboxu.



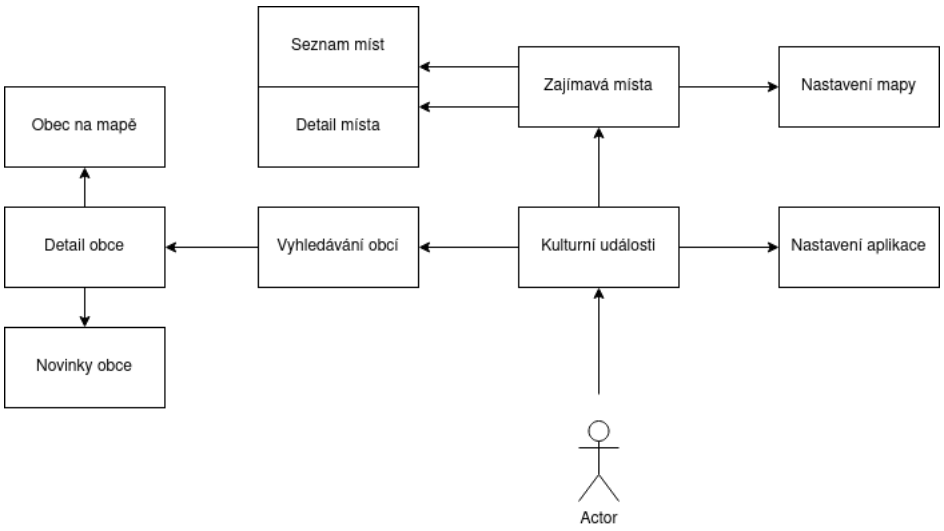
Obrázek 3.1: Graf navigace obyvatele

3.10.2 Navigace pro návštěvníka

Navigace návštěvníka začíná na úvodní obrazovce se seznamem nejnovějších kulturních událostí. Dále se navigací dostává na mapu se zajímavými místy v jeho okolí, ke kterým si může spustit navigaci.

Stejně jako obyvatel obce si může obec návštěvník vyhledat a případně si ji i nastavit jako „Moje město“, nicméně tato funkce není pro obyvatele

stěžejní. Další společnou funkcí je nastavení aplikace a nastavení mapy, kde je pro uživatele možné nastavit přizpůsobení mapy a lokalizaci aplikace.



Obrázek 3.2: Graf navigace turisty

4 Vývoj mobilních aplikací pro platformu Android

Součástí zadání je implementace navržených funkcí jako aplikaci pro platformu Android. Proto bude v následujících sekcích přiblíženo fungování platformy Android a specifika vývoje SW.

4.1 Platforma Android

[3]Android je mobilní operační systém založený na modifikované verzi OS Linux. Původně byl vyvíjen vývojářskou společností stejného jména, Android, Inc. V rámci úmyslu vstoupit na mobilní trh byl Android zakoupen společností Google a ta převzala jeho vývoj.

[3]Google chtěl aby byl Android otevřený a zdarma, proto byla většina Android kódu vydána pod Apache Licencí. To znamená, že kdokoliv chce používat Android si jej sám stáhnout a modifikovat. Tohoto často využívají výrobci mobilních zařízení, kteří si do původního OS Android přidávají vlastní funkcionality pro jejich zařízení. To dělá Android pro výrobce mobilních zařízení velmi atraktivní a je zásadním rozdílem oproti OS iOS od společnosti Apple, který je uzavřený a mobilní zařízení jsou vyráběna exkluzivně společností Apple.

V současné době se OS Android objevuje v široké řadě druhů zařízení, nejčastěji chytré telefony, tablety, čtečky, chytré hodinky nebo internetové televize. V poslední době se rozšiřuje použití OS Android v palubních počítačů automobilů.

4.2 Specifika vývoje

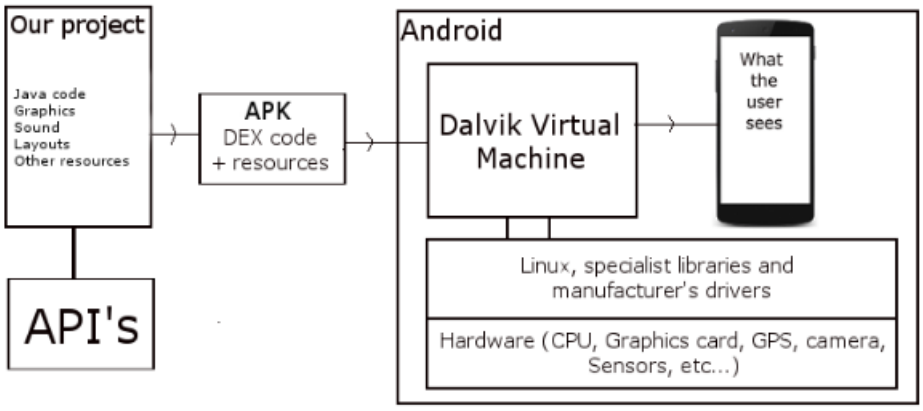
Dominantním jazykem pro vývoj aplikací pro Android je jazyk Java, který je však v současné době rychle nahrazován modernějším jazykem [6] Kotlin vyvinutým společností [8] JetBrains s.r.o. Kotlin je moderní jazyk schopný fungovat na JVM a je překládán do stejného bajtkódu, to zajišťuje zpětnou kompatibilitu s Javou. Tato kompatibilita však není plně oboustranná, Java nemůže vždy volat knihovny Kotlinu, protože některé jeho funkcionality nepodporuje. Kotlin přináší oproti Javě mnoho zajímavých funkcionalit, například null-safety, extension functions, data classes apod.

Nejrozšířenějším vývojovým prostředím pro vývoj Android aplikací je IDE Android Studio, založené na vývojových prostředích od společnosti JetBrains s.r.o. Dále se Android aplikace často vyvíjí i v prostředí IntelliJ IDEA od výše zmíněné společnosti, funkcionality tohoto IDE jsou velmi podobné těm Android Studia.

4.2.1 Sestavení a kompatibilita aplikací

Pro sestavení a ladění vyvíjených aplikací byl dříve používán systém Maven, v současné době je však tento postup zastaralý a je používán systém Gradle. Pro ladění jsou využívána fyzická zařízení nebo virtuální emulátory. Spustitelné verze aplikací jsou zabalena do souborů, které jsou používány pro distribuci aplikacé na systém Android. Při vývoji Android aplikací je potřeba brát v potaz, na které verze Android OS aplikace cílí. Některé funkce ve starších verzích Android SDK mohou být již zastarelé, naopak nové funkce nemusí být kompatibilní pro starší zařízení, které dané funkce nepodporují nebo nemají potřebný HW.

[7]Kód je na Android překládán do DEX (Dalvik Executable) kódu a spouštěn na DVM (Dalvik Virtual Machine), který má propojení na OS Linux. Zkompilovaný Java kód je s dalšími zdroji umístěn do balíčků APK (Android Application Package).



Obrázek 4.1: [7]Postup překladu Android aplikace

4.2.2 Uživatelské rozhraní

Implementace uživatelského rozhraní (dále jen UI) spočívá v jeho definici v XML souborech pomocí grafických komponent již definovaných v Android SDK. Mezi tyto komponenty patří jednotlivá rozložení definující chování a

rozložení podřízených prvků v rozhraní, např. *LinearLayout*, *RelativeLayout* nebo *ConstraintLayout*, a jednotlivé prvky, se kterými interaguje uživatel, např. *Button*, *TextView* nebo *ImageView*. UI je možné definovat i programovým kódem, tento postup je však zbytečně zdlouhavý a nepřehledný.

Jednotlivé UI komponenty mohou být v kódu referencovány několika způsoby. Jedním z nejstarších způsobů je získání reference voláním funkce *View.findViewById(int id)* nad rodičovským elementem hledaného prvku. Tento postup je poměrně spolehlivý pro získání reference, neboť vrátí referenci na hledaný prvek vždy, když jsou on i jeho rodičovský element vykresleny. Nicméně tento způsob je zbytčně zdlouhavý a způsobuje vysoký nárůst v množství kódu.

S příchodem podpory programovacího jazyka Kotlin přišel i nový způsob referencování UI komponent. Použitím pluginu Kotlin Android Extensions a balíčku *kotlinx.android.synthetic* je nyní možné referencovat UI komponenty z kódu přímo identifikátorem, který je deklarován u dané komponenty v XML definici. Tento způsob reference je velmi pohodlný a od počátku podpory jazyka Kotlin pro Android byl velmi oblíbený.

Nejnovějším způsobem reference UI komponentem je v současné době pomocí *DataBinding* knihovny od Googlu. DataBinding knihovna umožňuje vygenerování abstraktního modelu založeného na XML definici UI. Během sestavení programu je tato abstrakce vygenerována a je následně možné referencovat přímo přes vygenerovanou abstrakci. Tento postup přivádí dále možnost provázání UI s daty, UI je pak automaticky překresleno při změně dat a dále tak šetří množství psaného kódu.

4.3 Životní cyklus Android aplikací

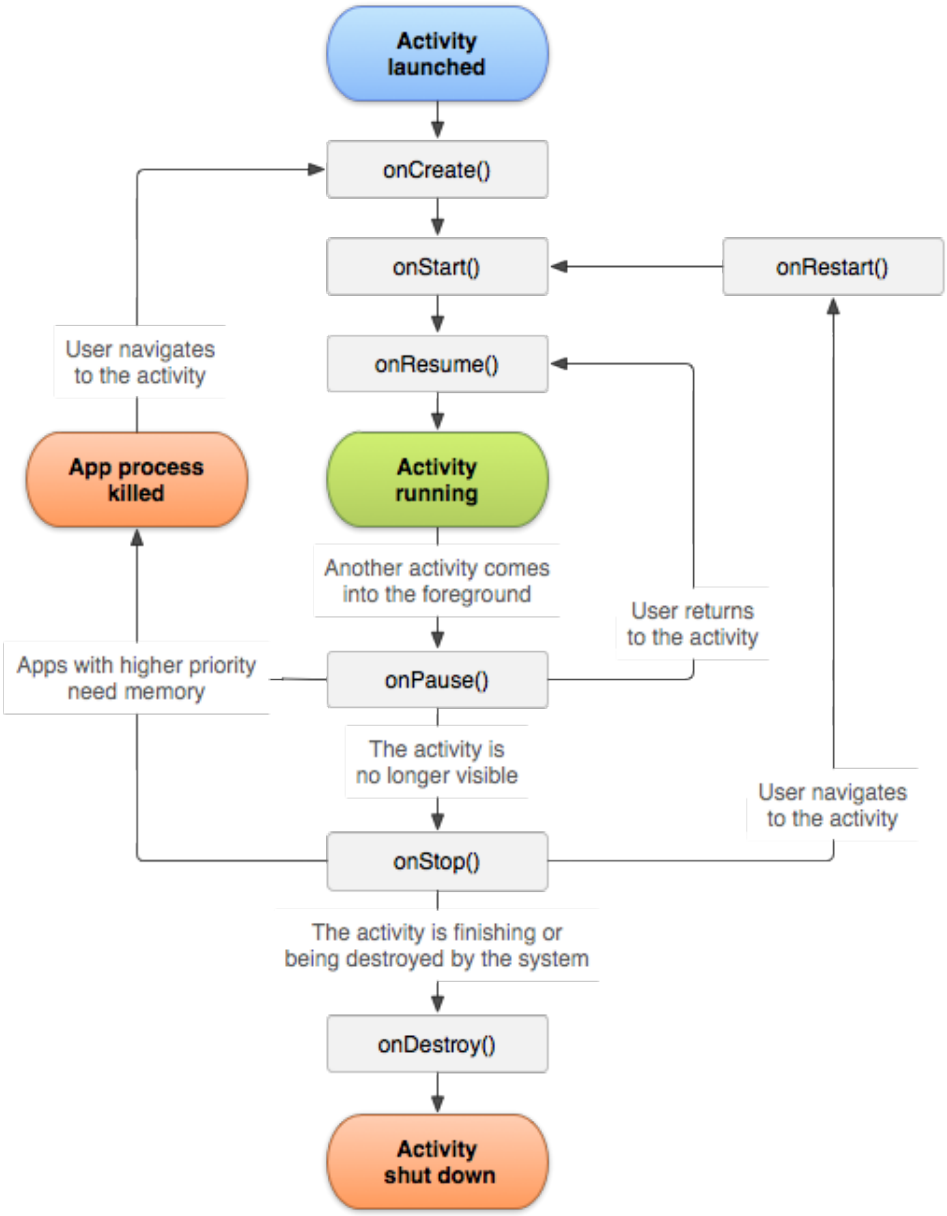
Pro Android aplikace jsou charakteristické životní cykly jejich běhu, na které musí být během vývoje aplikace brán zřetel. Životní cykly Android aplikacé jsou významné především v rámci *Activit* a *Fragmentů*, především jsou ovlivněny fáze tvorby a zániku jejich UI.

4.3.1 Životní cyklus Activit

Activita je základním prvek interakce s uživatelem, stará se o iniciální vykreslování aplikace. Jedna *Activita* reprezentuje jednu obrazovku aplikace, aplikaci je tedy teoreticky možné napsat exkluzivně pomocí *Activit*. Tento postup je však velmi nepraktický a přináší svá úskalí v ohledech navigace a především zpracování dat.

Průběh životního cyklu *Activit* se odvíjí od jeho skutečných činností, související s jeho vytvořením, vykreslením prvků, zničení nebo uvedení aplikace na pozadí systému. Životní cyklus je reprezentovány funkcemi, které systém volá nad *Activitou* a dává tak možnost vývojáři například otevřít připojení ke vzdáleným zdrojům ve vhodnou dobu, naopak je uvolnit nebo i obnovit stav aplikace po jejím znovuotevření. Pro připojení ke zdrojům je například vhodné během volání funkce *onStart()*, naopak jejich uvolnění je vhodné provádět během *onStop()* a *onDestroy()*. Pro uložení stavu aplikace je používána funkce *onPause()*, naopak pro jeho obnovu je používána funkce *onResume()*.

Řízení se tímto životním cyklem je během vývoje absolutně klíčové pro stabilitu aplikace i pro správné fungování implementovaných funkcionalit.



Obrázek 4.2: [1] Graf hlavního životního cyklu Activity

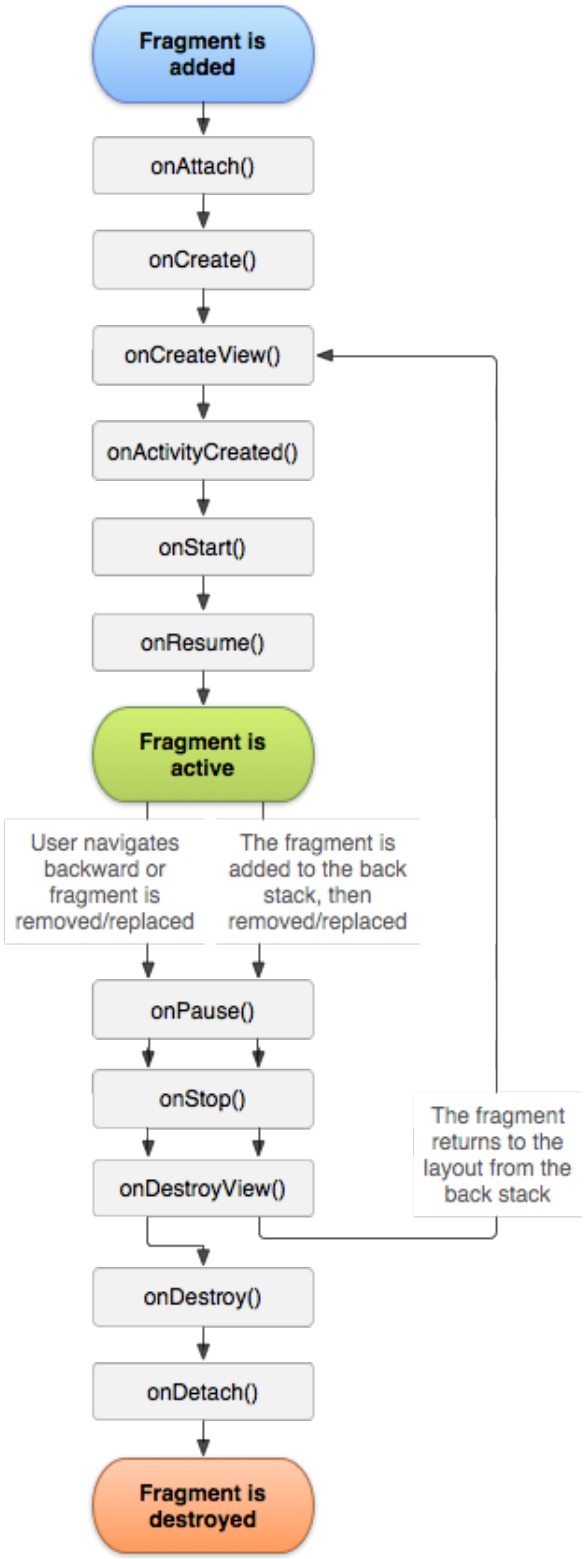
4.3.2 Životní cyklus Fragmentů

Fragmenty jsou objekty reprezentující chování aplikace na úrovni podřazené *Activitám*. Mohou obsahovat vlastní UI, se kterým může uživatel interagovat, ale zároveň žádné UI obsahovat nemusí a mohou být zodpovědná pouze za nějakou logickou činnost. Jejich nejrozšířenější využití je

právě pro reprezentaci obrazovek, často se proto v současné době objevují aplikace využívající pouze jednu *Activitu* a mnoho *Fragmentů*. Tento postup přináší mnoho výhod, především v rámci navigace, pro kterou v současné době existujjí vlastní komponenty (tzv. *NavController*), a předávání dat mezi fragmenty pomocí argumentů.

Životní cyklus *Fragmentu* je velmi podobný životnímu cyklu *Activity*, rozdíl je zde však v hierarchii těchto dvou komponent. Jelikož je *Activita* komponentou nadřezenou *Fragmentu*, její životní cyklus předchází životnímu cyklu *Fragmentu*. *Fragment* může být pádem vytvářen až po vytvoření *Activity*. Naopak při ukončování aplikace předchází životním cyklus *Fragmentu* tomu *Activity*.

Podobně jako v případě *Activity* i životním cyklu *Fragmentu* je vhodná alokace zdrojů v prvních fázích životního cyklu, například v rámci funkce onActivityCreated() nebo onStart(), a pozdější fáze pro jejich uvolňování.



Obrázek 4.3: [5] Graf hlavního životního cyklu Fragmentu

V případě *Activit* i *Fragmentů* životní cyklus silně ovlivňuje přístupnost a vykreslení UI. Například v případě *Fragmentu* je možné referencovat prvky UI už ve funkci *onCreateView()*, vykreslené jsou však až ve funkci *onViewCreated()*, do té doby tedy nebudou validní jejich vlastnosti jako například šířka a výška.

V současné době jsou v Android SDK implementovány komponenty *ViewModel* a *AndroidViewModel*, jejichž implementace následně slouží jako prostředník mezí prezentační a datovou částí aplikace. Tyto komponenty jsou provázány s životním cyklem *Fragmentů* a *Activit* a na rozdíl od *Fragmentů* a *Activity* dokážou udržet svůj stav i při změně konfigurace jako otočení orientace zařízení nebo uvedení aplikace na pozadí. Tyto komponenty je pak vhodné provázat s UI aplikací a uchovat tak data i obsah UI prvků napříč životním cyklem.

4.4 Úložiště dat na zařízeních Android

Persistence dat na Android zařízeních je možné realizovat několika způsoby. Základním postupem je uložení dat v soukromém úložišti dané aplikace, který je alokován systémem pro její fungování. Do tohoto úložiště jsou ukládána data, ke kterým by ostatní aplikace neměly mít přístup. Často tak jde například o konfigurační soubory nebo o výstupy logování aplikace.

Možnost zápisu dat na pevný disk zařízení nebo SD kartu může být využita v případě, kdy data není potřeby nijak zabezpečovat nebo je potřeba s jejich pomocí komunikovat s ostatními aplikacemi.

Drobná data jako textové identifikátory nebo číselné parametry je možné ukládat do tzv. *Preferences*. Jde o soukromé úložiště určené pro aplikaci, do kterých má přístup jen aplikace sama. Hodnoty dat jsou ukládány pod textovými klíči. Tento postup je zpravidla používán pro ukládání jednoduchých konfigurací a předávání některých parametrů napříč aplikací. Tento postup není vhodný pro persistenci velkých dat a objektů.

Nejvíce používaný způsob persistence dat je implementace SQLite SŘBD. Tímto způsobem je možné dlouhodobě persistovat data a manipulovat s nimi pomocí SQL dotazů, případně vytvářet vztahy mezi tabulkami. V současné době je pro tento účel nejvíce používaná abstrakce SQLite Room od Googlu, který je schopen sám vygenerovat databázi, tabulky a přístupy k nim jen pomocí abstrakce, kterou poskytne sám vývojář.

4.5 Současné verze Android

Verze Android mají obsáhlou historii, každá verze přinášela na mobilní zařízení nové funkcionality a podporu pro jejich HW. Android verze jsou známé označováním podle názvů sladkostí, například Android 5.0 Lollipop, Android 6.0 Marshmallow apod. Tato tradice však skončila verzí Android 9 Pie, poslední verze Android 10 se touto tradicí již neřídí.

Mezi nejnovější přidané funkcionality patří například API pro neuronové sítě ve verzi Android 8.1, podpora pro Vulkan API a ovládání gesty ve verzi Android 9, tmavý mod a zkratky aplikací ve verzi Android 10.

4.6 Podíl užívaných verzí

Níže uvedná tabulka reprezentuje distribuci verzí systému Android ma všech druzích zařízení.

Android Name	Android Version	Usage Share
Pie	9	10.4%
Nougat	7.0, 7.1	19.2% ↓
Marshmallow	6.0	16.9% ↓
Lollipop	5.0, 5.1	14.5% ↓
Oreo	8.0, 8.1	28.3% ↑
KitKat	4.4	6.9% ↓
Jelly Bean	4.1.x, 4.2.x, 4.3.x	3.2% ↑
Ice Cream Sandwich	4.0.3, 4.0.4	0.3%
Gingerbread	2.3.3 to 2.3.7	0.3% ↑

Obrázek 4.4: [12] Distribuce verzí Android ke květnu 2019

Z tabulky vyplývá, že vývoj aplikace kompatibilní s verzí alespoň Android 6.0 zpřístupní danou aplikaci pro alespoň 74,8 % mobilních zařízení se systémem Android. V době psaní této práce je toto procento zřejmě vyšší.

4.7 Architektury a filozofie při vývoji pro Android

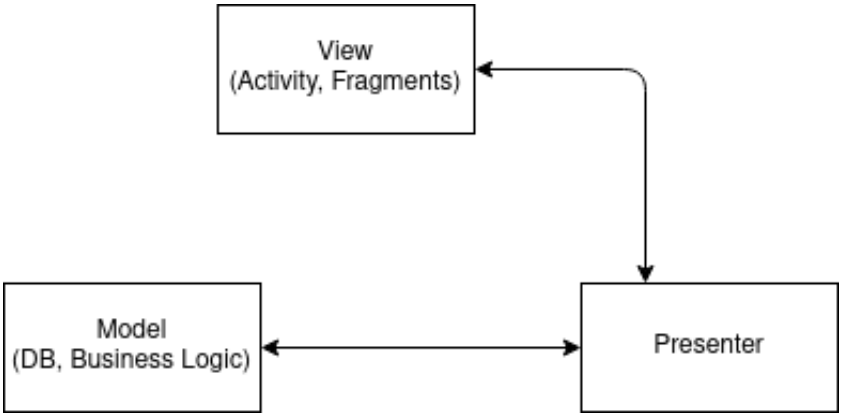
Architektura aplikace rozděluje fungování aplikace na uživatelskou interakci a zpracovávání dat, popisuje fungování aplikace, závislosti a vztahy mezi

jednotlivými komponentami. Architektura definuje rozhraní, pomocí kterých mezi sebou komunikují jednotlivé vrstvy architektury, případně definuje komunikaci s ostatními aplikacemi.

Použitím vhodné architektury se stává aplikace rozšiřitelná, přehlednější, jednodušší pro ladění a testování. Fungování aplikace testujeme především jednotkovým testováním jednotlivých funkcí, v případě prezentačnívh vrstev se používají tzv. UI testy (někdy také automatizované testy). Použitím vhodné architektury je díky izolaci jednotlivých vrstev jednodušší aplikaci testovat.

4.7.1 MVP

MVP je architektura složená ze 3 základních vrstev - *Model*, *View* a *Presenter*, kde *Presenter* slouží jako prostředník mezi prezentační a datovou částí aplikace.



Obrázek 4.5: Schéma MVP architektury

Model

Model je rozhraní zodpovědné za správu dat aplikace, tím může být například přpojení k SQLite databázi (nebo jiné SŘBD), připojení k REST API a jiným zdrojům mimo paměť aplikace. Funkce *Modelu* jsou přístupné *Presenteru* pomocí rozhraní, které *Model* implementuje.

View

Účel *View* (prezentační vrstva) je zobrazovat data v takové podobě, v jaké je dostalo od svého *Presentera* (doménová vrstva). *View* by mělo obsahovat co nejméně funkčního kódu, respektive co nejméně logiky, protože prezentace

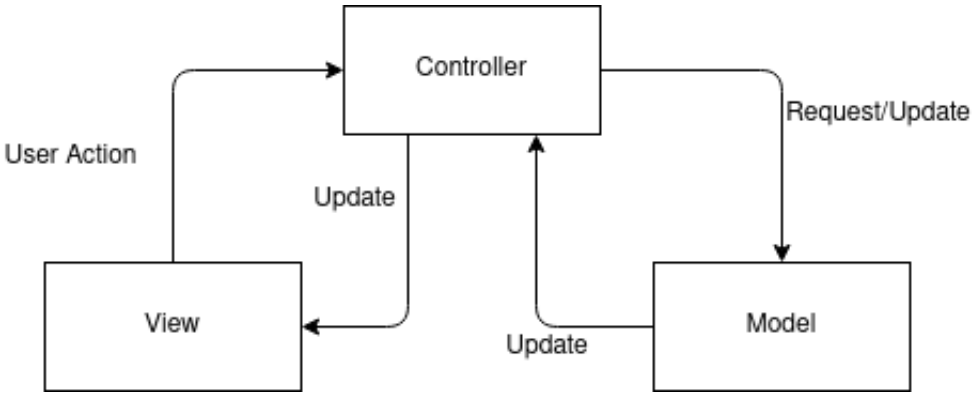
dat a interakce s uživatelem je jeho jediným účelem. *View* má s *Presenterem* vztah 1:1, tedy pro jedno *View* existuje jen jeden *Presenter*.

Presenter

Presenter funguje jako prostředník mezi *View* a *Modelem*, měl by obsahovat tzv. business logiku aplikace (vykonávání funkcí, které řeší konkrétní problémy). Z toho důvodu je ideální jej implementovat pomocí prostředků, které jsou nezávislé na platformě z důvodu použitelnosti pro multiplatformní vývoj. Je zodpovědný za aktualizaci datové vrstvy a předávání dat do *View*.

4.7.2 MVC

MVC je architektura složená ze tří základních částí - *Model*, *View* a *Controller*, kde *Controller*. Oproti MVP architektuře má MVC lepší propojení mezi *Controllerem* a *View*, kdy *Controller* může vědět o *View*.



Obrázek 4.6: Schéma MVC architektury

Model

Oproti MVP architektuře, zde *Model* přebírá zodpovědnost za business logiku aplikace. Mimo ní zároveň slouží pro připojení k databázím, vzdáleným zdrojům a komunikaci po síti. *Controlleru* je *Model* dostupný implementací deklarovaným rozhraní.

View

View je v MVC architektuře zodpovědný pouze za zobrazení dat poskytnutých *Controllerem*. V případě Androidu tak obsahuje pouze Fragmenty, Activity, případně vlastní implementace grafických komponent.

Controller

V MVC architektuře *Controller* zpracovává vstup od uživatele a rozhoduje, které View se bude uživateli zobrazovat. Zároveň nemusí být vztah mezi *Controllerem* a *View* 1:1, jeden *Controller* může obsluhovat více *View*. V případě MVC architektury se často mluví o tzv. *aktivním Controlleru*, který má referenci na *View* pomocí rozhraní, které *View* implementuje. Tento postup není povinností a je zcela na uvážení vývojáře.

Na rozdíl od *Presenteru* v MVP architektuře není *Controller* zodpovědný za business logiku aplikace, tuto zodpovědnost zde přebírá *Model*.

4.7.3 MVVM

MVVM se skládá ze 3 základních vrstev - *Model*, *View* a *ViewModel*. Myšlenka za implementací *ViewModelů* je uchovávání stavů a informací právě zobrazených ve *View*.

Model

Model reprezentuje doménové objekty, reprezentuje skutečná data a informace, se kterými aplikace pracuje pro vykonání svého účelu (*business logic*). U modelu je důležité si uvědomit, že jednotlivé objekty obsahují informace (například telefonní číslo, jméno...), ale ne logiku pro manipulaci s danou informací. Model může obsahovat i databáze nebo připojení ke vzdálenému serveru, ale tyto objekty jsou samy modelem, které pracují s informacemi jiných objektů.

View

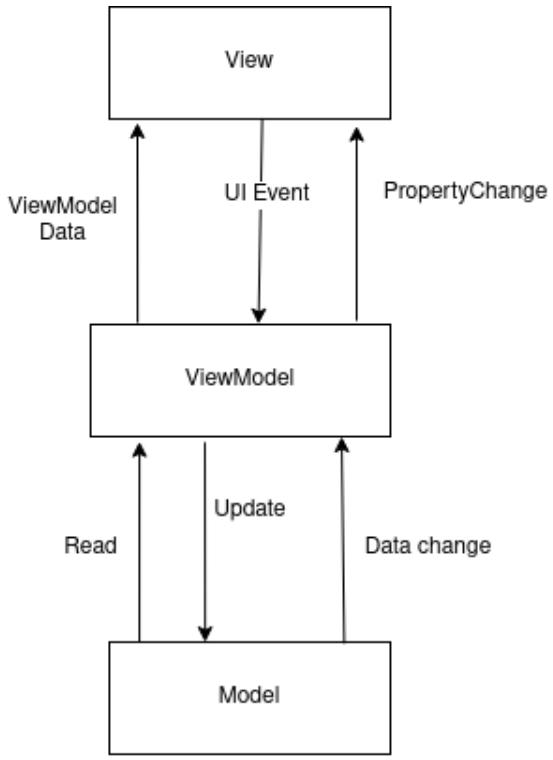
View je místo, kde dochází k prezentaci dat a interakci s uživatelem. Obsahuje minimum funkční logiky, zpravidla formátovací funkce pro výstup textu a jiných dat. Dále zaznamenává vstup uživatele, jako stisk tlačítka, kliknutí na displej apod.

V případě MVVM architektury je View takzvaně *aktivní*, to znamená, že obsahuje chování a zpracování událostí, které vyžadují znalost chování ViewModelu a Modelu. Naopak *pasivní* View nemá žádnou znalost o Modelu a je samo manipulováno přiřazeným Controllerem/Presenterem.

ViewModel

ViewModel slouží jako prostředník mezi View a Modelem tak, aby View nemuselo znát chování Modelu a zároveň aby Model nemusel vědět o změ-

nách stavu ve View, což je v případě aplikací pro Android zásadní kvůli životnímu cyklu Activit, Fragmentů a jejich Views. ViewModel zároveň zařizuje takzvaný *two-way databinding*, to znamená, že uživatelská akce ve View se promítne v datech Modelu a naopak změna dat v Modelu se promítne ve View, například v obsahu textu.



Obrázek 4.7: Schéma MVVM architektury

4.7.4 Clean Architecture

Clean Architecture není konkrétní implementace architektury, jde primárně o koncept, filozofii vývoje software. [10][9] Aplikace založená na Clean Architecture splňuje tzv. SOLID principy:

- Single responsibility principle/Princip jedné zodpovědnosti - třída by měla mít jednu a pouze jednu zodpovědnost
- Open-closed principle/Princip otevřenosti-uzavřenosti - aby bylo systémy jednoduché měnit, musí být navrženy tak, aby se jejich chování měnilo spíše přidáním nového kódu než modifikací existujícího
- Liskov's substitution principle/Liskovův princip nahraditelnosti - každý

objekt musí být nahraditelný instancemi jejich potomků, bez pozměnění korektnosti systému

- Interface segregation principle/Princip segregace rozhraní - mnoho specifičtějších rozhraní je lepší než jedno obecné rozhraní
- Dependency inversion principle/Princip inverze závislosti - komponenty se spoléhají na abstrakci, ne na konkrétní implementaci

Celý návrh se následně formuje do konkrétnější podoby pomocí následujících modulů [11].

Entities

Entity (nebo také doménové objekty) reprezentují nositele informací v aplikaci a obsahují pouze data, nejsou zodpovědné za posílání zpráv v kontextu aplikace nebo poskytování služeb. Výjimkou je logika, kterou aplikuje sama na sebe.

UseCase

Případy užití reprezentují business logiku aplikace, tedy vykonávání logiky, která řeší konkrétní problémy. K poskytovatelům dat přistupují pomocí rozhraní, které poskytovatelé dat implementují, ale případu užití nezáleží na tom, odkud konkrétně data přicházejí (viz princip inverze závislosti).

Data providers

Účel poskytovatelů dat je přistupování dat v různých prostředích, ať už jde o vzdálený server, lokální databázi nebo souborový systém. Poskytovatelé dat implementují rozhraní definované na straně *UseCase*. Tento modul bude zároveň obsahovat entity databázových tabulek v případě použití ORM implementace databáze.

Entrypoints

Vstupní body jsou způsoby interakce s aplikací. Nemusí nutně jít jen o vstupy od uživatele, vstupní body zahrnují i dlouhodobě běžící procesy a komunikaci ze třetí strany, případně komunikaci s RESP API.

Configuration

Konfigurace je zodpovědná za provázaní všech modulů do jednoho celku. Často se stará o logování činností aplikace, DI framework, konfigurace přístupu ke zdrojům dat, apod.

4.7.5 Závěr

Existuje mnoho architektur pro vývoj Android aplikací a aplikací obecně, zde jsou zmíněny historicky nejpopulárnější. Použití vhodné architektury je závislá na uvážení, vkusu a zkušenostech vývojáře, který musí brát v potaz veškeré uživatelské a technické požadvky projektu. Současná implementace Android SDK však svádí k použití MVVM architektury (abstrakce komponenty *ViewModel* a *AndroidViewModel*). Často je tento postup kombinován s dodržováním principů definovaných v *Clean Architecture*.

5 Realizace aplikace

Aplikace byla vyvinuta pomocí moderního programovacího jazyka Kotlin. Kotlin byl zvolen pro své výhody oproti Javě - null-safety, typová inference, čitelnější kód, silnou podporu a propagaci ze strany Googlu, tudíž bude aplikace v budoucích letech lépe udržitelná a rozšiřitelná. Pro sestavení aplikace je použit nástroj Gradle, jelikož jde o standardní postup. Zdrojový kód aplikace byl verzován pomocí verzovacího systému Git.

5.1 Architektura

Pro realizaci aplikace byla zvolena architektura Clean Architecture (Kap. 3.7.4), protože jde v současné době o velmi populární postup implementace Android aplikací. Volba této architektury umožní rozdělení aplikace do několika přehledných modulů, každý plnící konkrétní účel. Jako propojení mezi prezentační a datovou vrstvou budou sloužit *ViewModely*, jejichž abstrakce je součástí Android SDK.

5.1.1 Implementace Clean architecture

Aplikace je založena na myšlenkách Clean architecture a implementována pomocí následujících modulů.

Presentation

Primární zodpovědností *Presentation* modulu je zobrazování dat na výstup displeje zařízení. Toho je dosaženo použitím *Activit* a *Fragmentů*, které dědí ze základní abstrakce *BaseActivity* a *BaseFragment*. *Presentation* modul obsahuje implementaci všech GUI prvků.

S ostatními moduly komunikují z *Presentation* modulu *ViewModely*, které jsou připojeny na *Fragmenty* a dle potřeby i hlavní *Activity* aplikace. Do Fragmentů jsou události změn dat a výsledků operací propagovány notifikací *LiveData* objektů, ke kterým je připojen *Observer*, který tyto události zpracová a předá *Fragmentu*, který je přihášen k odběru těchto dat.

Mimo zobrazení dat obsahuje *Presentaion* modul i konfiguraci aplikace, například inicializaci knihoven pro logování Timber a DI frameworku KOIN.

Domain

Domain modul reprezentuje business logiku aplikace. Je klíčové, aby tento modul nebyl nijak závislý na Android SDK, to ho činí vhodným pro multiplatformní vývoj. Více platforem by tak mohlo používat stejný kód pro vykonávání business logiky aplikace.

O interakci uživatele je *Domain* modul notifikován voláním *UseCase* objektů *ViewModelem*, které vykonávají business logiku aplikace a pokud je třeba, předávají data zpět do *ViewModelu*. *UseCase* objekty implementují předem deklarovanou abstrakci, která vyžaduje znalost o vstupní a výstupních parametrech konkrétního *UseCase*.

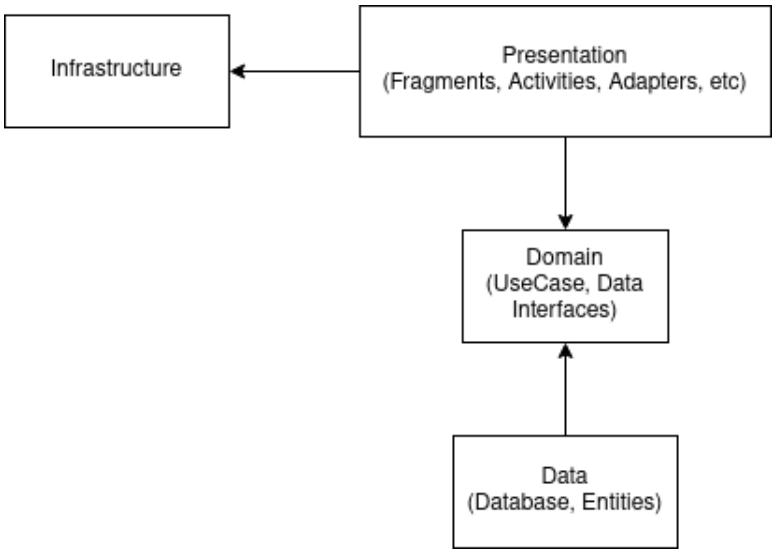
Pro komunikaci s *Data* modulem deklaruje *Domain* modul rozhraní, které *Data* modul následně implementuje. Data modul díky těmto rozhraním dokáže vracet výsledky do *Domain* modulu, aniž by odkrýval svojí implementaci.

Data

Zodpovědností *Data* modulu je komunikace s datovými zdroji třetích stran, například vzdálené servery, REST API, apod, a správa komunikace s databází. Databáze je v tomto případě implementována jako SQLite SŘBD pomocí abstrakce Room API, kterou poskytuje Google a je v současné době moderní implementací DB v Android aplikacích. Struktura této DB je reprezentována pomocí entit představující databázové tabulky. Tyto entity podporují běžné tabulkové SQLite vlastnosti jako primární klíče, cizí klíče, kaskády, relace, apod. Přístup k těmto entitám je realizován pomocí DAO (Data Access Object) objektů, které slouží jako abstrakce deklarující funkce pro přístup do DB pomocí klasických SQLite příkazů [2].

Infrastructure

Infrastructure modul je primárně zodpovědný za činnosti spojené s HW zařízení, na kterém aplikace běží. Nejčastěji jde například o zapisování na disk nebo volání aplikací třetích stran. Částečně je proto zodpovědný za logování do souborů.



Obrázek 5.1: Schéma závislostí modulů

5.2 Zdroje dat

Pro účely této práce bylo zavadeno několik zdrojů dat, některé jsou poskytnuty třetí stranou, jiné byly zřízeny pro účely této práce.

5.2.1 Firebase

Firebase je back-end (dále jen BE) vyvinut společností Google pro vývoj mobilních a webových aplikací. Firebase nabízí mnoho služeb, například Firebase Authentication pro autentikaci uživatelů, Firebase Database pro databázi, Firebase Storage pro vzdálené úložiště a další. V této práci byly použity služby Firebase Database a Firebase Storage.

Firebase Database

Firebase Database nám umožnila vytvořit BE databázi pro mobilní aplikaci bez potřeby přímého zřizování hostingu. Veškerá konfigurace je již navržena na straně Firebase, detaily se dají ze strany uživatele konfigurovat.

Firebase Database nereprezentuje data pomocí tabulek a relací, nýbrž jako JSON objekty ve vztahu *klíč* → *hodnota*. Relace je tak do jisté míry možné reprezentovat na straně mobilní aplikace. Tato vlastnost vyžaduje na straně aplikace deserializaci z JSONu do klasických Java/Kotlin objektů.

Pomocí Firebase Database jsme byly schopni reprezentovat data obcí, kde jako klíč byl zvolen anglický název města malým písmem a JSON objekt

reprezenzující konkrétní obec jako hodnota. Objekt reprezentuje data jako název obce, cesta ke znaku obce, obecné informace a URL RSS kanálu. Reprezentace obce tak může vypadat například takto:

```
{
  "pilsen" : {
    "id" : "pilsen",
    "name" : "Plzen",
    "name_en" : "Pilsen",
    "rssFeed" : "https://www.plzen.eu/",
    "rssUrl" : "rsshandlerext.aspx?exportId=34&dontparse=false",
    "wiki" : {
      "citizens" : 172000,
      "gps" : {
        "lat" : 49.7455665,
        "lng" : 13.3636333
      },
      "headline" : "...",
      "headline_en" : "...",
      "logo" : "plzen_logo.jpg"
    },
    "www" : "https://www.plzen.eu/"
  }
}
```

Samotné obce jsou pak uloženy pod klíčem *cities*.

Právě díky podpoře Firebase Database jsme schopni implementovat takové funkcionality jako detail obce, přesměrování na její web nebo připojení na RSS kanál. Firebase také podporuje živé připojení na uzel v databázi identifikovaným, konkrétním klíčem, aplikace je pak notifikována vždy po změně obsahu některého potomka. Díky tomu jsme byli schopni implementovat našeptávač obcí, který je vždy aktuální nezávisle na tom, jestli jej má uživatel právě otevřen nebo do něj například právě píše.

Obdobným způsobem jsme schopni reprezentovat data vztahující se k hlášením zadaným uživateli. Hlášení jsou reprezentována klíčem, který je unikátně generován pri vložení do Firebase Database, data obsahují informace jako titulek, tyo hlášení, popis, poloha, apod. Reprezentace hlášení pak vypadá takto:

```
{
  "incidents": {
    "INC12345": {
      "description": "Incident 1235",
      "createTime": 123456,
      "type": "PUBLIC_DAMAGE",
      "gps": {
```

```
        "lat": 49.7455665,
        "lng": 13.3636333
      },
      "time": 1234566,
      "title": "Titulek 1",
      "img": "img.img"
    },
    ...
  }
}
```

Firebase Storage

Firebase Storage nám umožnila ukládat soubory do vzdáleného úložiště, je tak jednoduché sdílet data mezi uživateli. Mezi tato data patří především obrázky vztahující se k obcím, zde jde primárně o jejich znaky, a přílohy přiložené k hlášením od uživatelů.

Na jednotlivé soubory ve Firebase Storage se aplikace odkazuje pomocí jednoznačných identifikátorů v datech reprezentujících obce a hlášení. V případě obcí jde o atribut *logo*, pro hlášení pak atribut *img*.

Jednotlivé soubory jsou jednorázově načítány v případě potřeby, tedy při otevření detailu obce nebo otřevření detailu hlášení.

Firebase Cloud Messaging

[4] Firebase Cloud Messaging (dále jen FCM) umožňuje posílání push notifikací na Android zařízení pracující s danou aplikací. FCM rozlišuje dva typy zpráva - *notification messages* (dále jen NM) a *data messages* (dále jen DM).

Notification messages jsou zprávy automaticky zpracovávané FCM SDK, v případě zpracování takové zprávy je uživatelům na Android zařízení zobrazena notifikace. NM mají předem definované klíče a hodnoty v jejich zprávách. Tyto notifikace mohou vypadat například takto:

```
{
  "message":{
    "token":"bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...",
    "notification":{
      "title":"Slavnosti svobody zruseny",
      "body":"Z duvodu koronavirove pandemie jsou plzenske
          slavnosti svobody zruseny"
    }
  }
}
```

Za zpracování DM je zodpovědná klientská aplikace a toto zpracování musí být implementováno vývojářem. DM nemají předem dané klíče pro ukládání hodnot, tyto klíče mohou být definovány vývojářem a může tak definovat protokol mezi klientskou aplikací a FCM. DM notifikace mohou vypadat například takto:

```
{
  "message":{
    "token":"bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...",
    "data": {
        "title": "Slavnosti svobody zruseny",
        "body": "Z duvodu koronavirove pandemie jsou plzenske
           slavnosti svobody zruseny"
     }
  }
}
```

NM i DM mohou být odesílány dvěma způsoby:

- Konkrétnímu zařízení
- Všem zařízením přihlášeným k odběru daného tématu (dále jen *topic*)

[4] Výše uvedené příklady popisují odesílání zpráv na konkrétní zařízení, které je identifikováno hodnotou klíče *token*. V případě odesílání zpráv na daný *topic* se musí klientské aplikace přihlásit k jejich odběru pomocí FCM SDK. Tato funkce FCM byla v této aplikaci využita pro odesílání aktualit vztahujících se k obci, kterou uživatel dříve označil jako „Moje město“. Ceý obsah zprávy používaný v této aplikaci může pak vypadat například takto:

```
{
  "to": "/topics/pilsen",
  "data": {
    "title": "Slavnosti svobody zruseny",
    "body": "Z duvodu koronavirove pandemie jsou plzenske
       slavnosti svobody zruseny",
    "url": "https://www.plzen.eu/"
  }
}
```

Vzhledem k tomu, že obsahem této práce je pouze Android aplikace, musí být tato funkce volána manuálně, například pomocí služby jako *Postman* pro volání *HTTP* požadavků.

5.2.2 Google Maps API

Společnost Google nabízí pro webový a mobilní vývoj API pro práci s Google mapami. Tato služba je v této aplikaci využita a je pro její fungování

klíčová pro zobrazení polohy uživatele, hlášení a zajímavých míst. Tyto objekty jsou na mapě zobrazeny na mapě podle vlastních ikon, reprezentujících zda jde o hlášení nebo zajímavé místo, případně jejich kategorii.

Google Maps nabízí možnost shlukování bodů do shluků v případě oddálení mapy. Tato funkcionalita je v aplikaci využita, její účinnost je velmi znát především při výskytu většího množství položek na mapě, které se během vývoje silně podepisovaly na výkonnosti aplikace.

Pro plné využití funkcionalit Google map je ideální sdílené polohy uživatele, ke které aplikace potřebuje explicitní souhlas uživatele. Mapa a funkcionality na ní závislé jsou však schopné fungovat i bez sdílení polohy.

5.2.3 Google Places API

Google Places API je služba poskytována společností Google, která je schopna na základě zadané polohy, dosahu v metrech, typu míst a dalších parametrů poskytnout data obsahující informace o zajímavých místech v okolí. Kategorie těchto míst je obsáhlá a naše aplikace nevyužívá je všechny, protože některé z nich nejsou pro její účely relevantní. Některá místa mohou patřit do více kategorií.

Získání informací z Google Places probíhá voláním GET požadavku nad dedikovaným URL s parametry. Podoba URL může být například:

```
https://maps.googleapis.com/maps/api/place/nearbysearch/json?
    location==49.767483,13.3703997&radius=5000&type=bar&key=[
    API_KEY]&pagetoken=
```

Data je možné stahovat jako XML nebo JSON, to je specifikováno v URL pomocí identifikátory *json* nebo *xml*. K URL je nutné připojit parametr *key*, který identifikuje volajícího klienta, API klíč je generován v Google konzoli a je vázán na konkrétní typ aplikace, případně přímo na její identifikaci pomocí názvu balíčku. Parametr *pagetoken* slouží pro stránkování na straně Google Places API, díky němu je možné volat URL vždy s novým *pagetoken*, který vrátí další výsledky. Jedno volání je vždy omezené na 20 výsledků. Toto volání je možné použít maximálně třikrát, uživatel tak tedy dostane maximálně 60 výsledků pro dané URL. Tento token má další omezení, které je vázáno k jeho platnosti, je nutné počkat přibližně 2 vteřiny před dalším voláním a použitím nového tokenu. Pokud tato prodleva nebude dodržena, token nebude platný a volání dostane stejný výsledek jako předešlý. Toto omezení je pak vzhledem k čekání na výsledky uživatelsky nepříjemné a není v aplikaci použito. Omezení na 20 výsledků je řešeno voláním endpointu vždy pro konkrétní kategorie místa a použitím menšího okruhu vyhledávání. Použitím

menšího okruhu dostáváme relevantnější výsledky vzhledem k poloze uživatele. Aplikace využívá až 9 kategorií míst, je tedy schopna získat až 180 výsledků za sérii volání.

5.2.4 RSS kanály

RSS (Rich Site Summary) kanály jsou rozšířený způsob jak sdílení zpráv v rámci různých institucí, v našem případě jde o obce. Některé obce nemusí mít vlastní RSS kanál, například Praha nemá vlastní RSS kanál, nicméně existuje mnoho zpravodajských služeb cílících na sdílení zpráv v oblasti Prahy, které RSS kanály používají. Data z RSS kanálu jsou serializovány v XML tabulkách.

RSS kanál je v případě naší aplikace využit pro zobrazování novinek v obcích, tyto novinky jsou následně persistovány v databázi. Zdroj novinek je znovu provoláván jen v případě, že je tabulka novinek prázdná nebo si obnovu uživatel explicitně vyžádá. Informace relevantní pro naší aplikaci jsou především nadpisy, popisy, datum vydání a URL, přes kterou je možné se směrovat přímo na zdroj novinky.

5.2.5 GoOut API

Zpbrazování kulturních akcí bylo implementováno pomocí veřejně dostupného GoOut API *https://goout.net/services/activityfeed/v2/feedpublic*. API přijímá číselný parametr *page* pro stránkování odpovědí ze serveru, číselný parametr *locality* jako identifikátor měst a textový parametr *language* pro specifikaci jazyka odpovědí.

Relevantní data získaná z API jsou persistována v databázi a následně zobrazena uživateli s informacemi jako název, popis, cena nebo lokalita akce. Kulturní akce je možné v nastavení aplikace filtrovat podle lokality.

5.3 Uživatelské rozhraní

Uživatelské rozhraní aplikace bylo implementováno tak, aby se obě skupiny uživatelů dostaly k nejdůležitějším funkcionalitám s použitím několika kliků. Aplikace je tak přehledná a jednoduchá pro navigaci. Pro chybové a varovné hlášky byly implementovány dialogy.

Pro konzistenci aplikace bylo definováno několik stylů, které jsou používány napříč aplikací a dělají ji přehlednou. Pro implementace ikon a grafických elementů byly použity volně dostupné zdroje dostupných na *mate-*

rial.io. Během vývoje uživatelského rozhraní bylo uplatněno několik následujících pravidel

- Použití vektorových obrázků pro správné škálování podle velikosti zařízení
- Definice znovupoužitelných stylů
- Oddělení stylovaích elementů od definice rozhraní

Rozhraní aplikace je navržené tak, aby odpovídalo *portrait* (vertikální) orientaci zařízení, protože *landscape* (horizontální), nepřináší implementovaným funkcionalitám přidanou hodnotu.

5.4 Použité knihovny

V průběhu vývoje bylo použito několik knihoven pro realizaci daných funkcí aplikace.

Timber

Timber je rozšířená knihovna pro logování zpráv pro ladění v Android aplikacích. Podporuje několik různých úrovní logování, které mohou být konfigurovány podle modu aplikace - debug, test, release, apod. `https://github.com/JakeWharton/timber`

KOIN

KOIN je knihovna sloužící pro DI. DI knihovny zjednodušují kód vytvářením instancí objektů dle předem definovaných pravidel. `https://insert-koin.io/`

Retrofit

Retrofit slouží pro zjednodušení síťové HTTP komunikace. Díky této knihovně stačí na straně klienta deklarovat rozhraní a anotacemi určujícími parametry URL, o zbytek volání a zpracování výsledku se postará Retrofit. `https://square.github.io/retrofit/`

Gson

Gson je rozšířená knihovna pro jednoduchou serializaci a deserializaci JSON objektů. Mimo jiné jej také využívá knihovna Retrofit pro deserializaci JSON objektů do instancí tříd používaných v klientské aplikaci. `https://github.com/google/gson`

5.5 Datový model

CityEntity

Tabulkový objekt sloužící pro reprezentaci obce. V aplikaci je využit primárně pro persistenci a reprezentaci rezidenční obce. Data obce jsou stahovány z Firebase pouze při jeho výběru ve vyhledávači.

LastSearchCityEntity

Tabulkový objekt sloužící pro reprezentaci posledních vyhledávání obcí.

IssueEntity

Tabulkový objekt sloužící pro reprezentaci hlášení zadaných uživateli aplikace. Jsou stahovány z Firebase vždy při vstupu na mapu a následně persistovány.

NewsEntity

Tabulkový objekt reprezentující novinky z obce. Uchovává informace jako URL, titulek, popis, datum a informaci o přečtení.

PlaceEntity

Tabulkový objekt reprezentující místo zájmu. Místa zájmu jsou stahována vždy po přístupu na mapu v aplikaci. Obsahují informace jako název místa, adresa, kategorii místa, info o otevření, apod.

PlaceSettingsEntity

Tabulkový objekt reprezentující nastavení viditelnosti kategorií míst zájmu na mapě.

IssueSettingsEntity

Tabulkový objekt reprezentující nastavení viditelnosti kategorií hlášení na mapě.

PhotoEntity

Tabulka reprezentující fotografii k místu zájmu. Vztah mezi *PhotoEntity* a *PlaceEntity* je realizován pomocí cizího klíče *placeId* odkazující na primární klíč v *PlaceEntity*.

CitySettingEntity

Tabulkový objekt pro reprezentaci nastavení filtru kulturních akcí dle měst. Obsahuje tedy číselníkový identifikátor města jako primární klíč a nastavení viditelnosti.

EventEntity

Tabulka pro reprezentaci události jako celku, obsahuje obecné informace jako název, URL hlavního obrázku a odkaz na detail události.

EventImageEntity

Tabulka pro uchování obrázků vztahujících se k dané události. Neobsahuje celé obrázky, jen jejich URL. Na rodičovskou událost se odkazuje pomocí cizího klíče.

LocalityEntity

Číselníková tabulka pro uchování míst konání události.

ScheduleEntity

Tabulka podřazená tabulce *EventEntity*, na kterou se odkazuje pomocí cizího klíče. Jde o informace konkrétního naplánování události včetně času události, ceny lístků, měny nebo lokality. Jedna *EventEntity* může být nadřazená více *ScheduleEntity*.

PushEventEntity

Tabulka pro persistenci notifikací z měst, ke kterým má užuvatel přihlášen odběr.

6 Příprava dat

Některé funkce aplikace závisí na přesné definici dat, která jsou nutná pro její správné fungování a testování. Předpřipravená data musí být především data měst, které si aplikace získává z Firebase Database. Dále je možné připravit data nahlášených podnětů, nicméně není to nutné, tato data jsou vytvářena uživateli aplikace.

6.1 Firebase data měst

Data měst jsou uchována ve Firebase v kořenovém adresáři na adrese , který je dále větven do uzlů *cities* (pro data měst) a *incidents* (pro data hlášení podnětů). Každé město je identifikováno unikátním *id*, jménem *name* a adresou webu města *www*, tyto parametry jsou povinné pro základní fungování funkcí jako vyhledávání měst nebo zobrazení jejich detailu. Objekt města dále obsahuje odkaz na zdroj RSS novinek *rssFeed* (například *https://prazsky.denik.cz/*) a konkrétní odkaz na RSS API nazvaný *rssUrl* (například *rss/z_regionu.html*). Tyto parametry tvoří meta data měst.

Podrobnější data měst jsou uložena pod uzlem *wiki*, jde o počet obyvatel *citizens*, popis měst *headline*, logo města *logo* a nejhlubší uzel *gps* s daty *lat* a *lng* dohromady tvořící zeměpisnou polohu města. Data uložená v atributu *logo* obsahují název obrázku na Firebase Storage, který si aplikace následně stahuje a zobrazí v detailu města. Obrázek je uložen na Firebase Storage je uložen ve složce dedikované danému městu, složka má název klíče města ve Firebase Database.

7 Ověření funkčnosti

V průběhu vývoje mobilních aplikací je implementovaný kód testován pomocí různých druhů testů. Jmenovitě jde o jednotkové testy (unit testy), instrumentální testy, UI testování a testování uživatelem za pomocí uživatelských scénářů. UI testy nebyly v této práci využity, protože neobsahuje komplexní práci s UI elementy (např. animace, transakce), které by vyžadovaly tento přístup.

7.1 Unit testy

Funkčnost business logiky aplikace byla testována jednotkovými testy, především pak mapování objektů mezi jednotlivými vrstvami aplikace a pomocné funkce, například pro formátování výstupů nebo serializace objektů do JSON textu. Pro jednotkové testování je, stejně jako pro ostatní JVM aplikace, používána knihovna JUnit.

Listing 7.1: Jednotkové testy pro formátování výstupu datumu.

```
@Test
fun dateToFormatStringTest() {
  val date = Date(1584819388288)
  val formatted = date.toFormattedString()

  Assert.assertNotNull(formatted)

  Assert.assertEquals("21.03.2020 08:36", formatted)
}

@Test
fun timestampToStringTest() {
  val time = 1584819388288L
  val string = time.timestampToString()

  Assert.assertNotNull(string)

  Assert.assertEquals("21.03.2020 08:36", string)
}
```

7.2 Instrumentální testy

Instrumentální testy jsou používány pro testování funkčností Android API. Zpravidla tak jde o testování přístupu k funkcím, které potřebují svolení od uživatelů, přístup k HW (například kamera) nebo databázové dotazy, právě ty byly v této aplikaci testovány pomocí instrumentálních testů.

Listing 7.2: Instrumentální test nastavení rezidentního města.

```
@Test
fun setResidentialTest() {
  val city = CityInformationDO(
   key = "A",
   id = "ID",
   name = "Pilsen",
   description = "desc",
   ...
  )

  runBlocking { cityRepository.setAsResidential(city) }
  val residential = runBlocking {
   cityRepository.getResidentialCity()
  }

  Assert.assertNotNull(residential)

  residential?.let {
   Assert.assertEquals("A", it.key)
   Assert.assertEquals("ID", it.id)
   Assert.assertEquals("Pilsen", it.name)
  }
}
```

Spuštění všech instrumentálních testů byl předcházen funkcí *setup()* a následován funkcí *tearDown()*, které se starají o přípravu a vyčištění testovaného prostředí.

Listing 7.3: Příklad funkcí setup() a tearDown().

```
@Before
fun setup() {
  val appContext = InstrumentationRegistry.getInstrumentation()
    .targetContext
  database = Room.inMemoryDatabaseBuilder(appContext,
    AppDatabase::class.java).build()
  lastSearchDao = database.lastSearchDao
  cityDao = database.cityDao

  cityRepository = CityRepository(...)
}

@After
fun tearDown() {
  database.clearAllTables()
  database.close()
}
```

7.3 Uživatelské testy

Největší část testování byla prováděna uživatelským testováním pomocí plnění testovacích scénářů. Testovací scénáře jsou založeny na základních uživatelských potřebách, které vychází z analýzy jiných aplikací a uživatelských hodnocení. Testování probíhalo nad předpřipravenými daty dostupných na Firebase.

V průběhu vývoje aplikace probíhalo testování primárně nad daty postavených na základě informací o Plzni. Plzeň byla pro toto testování vhodné například s ohledem na to, že má vlastní RSS kanál, některá města (například Praha) takový kanál nemá, v takových případech je obecní RSS kanál nahrazen RSS kanálem zpravodajské služby operující v daném městě.

```
{
  "pilsen" : {
    "id" : "pilsen",
    "name" : "Plzen",
    "name_en" : "Pilsen",
    "rssFeed" : "https://www.plzen.eu/",
    "rssUrl" : "rsshandlerext.aspx?exportId=34&dontparse=false",
    "wiki" : {
      "citizens" : 172000,
      "gps" : {
        "lat" : 49.7455665,
        "lng" : 13.3636333
      },
      "headline" : "...",
      "headline_en" : "...",
      "logo" : "plzen_logo.jpg"
    },
    "www" : "https://www.plzen.eu/"
  }
}
```

7.4 Firebase Crashlytics

Pro projekt byla aktivována služba Firebase Crashlytics, která je schopná detekovat pády aplikace a z konkrétní chyby vygenerovat podrobné hlášení s informacemi o pádu aplikace. Do informací z hlášení spadají informace jako konkrétní výjimka, která byla za pád aplikace zodpovědná, místo v kódu aplikace, kde k pádu došlo, informace o zařízení, na kterém k pádu došlo apod. Na základě tohoto hlášení může vývojář zpětně dohledat chyby v produkci a vydávat opravné verze aplikace.

8 Další vylepšení aplikace

8.1 Mobilní klient

Po technické stránce existují možnosti dalšího vylepšení v architektuře. V současné implementaci má *presentation* modul závislost na všech ostatních modulech kvůli iniciální konfiguraci aplikace. Zároveň se nabízí možnost rozdělit *data* na dva další moduly, jeden starající se o přístup k databázi, druhý pro komunikaci se vzdáleným serverem. Existují tedy možnosti vylepšení aplikace v rámci rozdělení a závislostí modulů.

Vzhledem k velikosti některých obcí nemusí existovat jednotný RSS zdroj novinek, místo toho na území obce mohou fungovat jiné zpravodajské služby využívající RSS. V případě více takových služeb pro jednu obec je vhodná implementace uživatelského výběru RSS zdrojů. Uživatel by si tak mohl přepínat novinky mezi různými médii. Tato funkce by mohla potenciálně fungovat plošně, ne jen pro jednu obec.

Existují možnosti implementace pro širší využití FCM notifikací, například pro hlášení stavu nahlášeného podnětu nebo přidání komentáře a jeho následných odpovědí u nahlášeného podnětu. Tyto funkcionality by bylo možné vhodně zapracovat ve webovém portálu.

8.2 Webový portál

Jako přirozené vylepšení celého systému se nabízí implementace vlastního back-endu (BE) s webovým portálem, který by měl na starost současné funkce Firebase nebo by sloužil jako rozhraní se stávající Firebase DB, tedy persistence dat sdílených mezi uživateli. Webový portál by mohl vhodně sloužit například pro vkládání vlastních obcí a vytvoření dalších funkcionalit, například kalendáře kulturních akcí v jednotlivých obcích.

9 Závěr

Během práce byla analyzována a rozebrána problematika potřeb návštěvníku a obyvatel obcí ČR. Byly analyzovány dostupné aplikace a byly rozebrány jejich funkce a nedostatky. Během analýzy byly vysvětleny specifika a náležitosti charakteristických pro vývoj aplikací pro platformu Android.

Na základě analýzy potřeb obyvatel a návštěvníků obcí ČR byly vybrány a navržen funkce, které staví na funkcích analyzovaných aplikací, rozšiřují je a vylepšují existující řešení.

Navržené řešení bylo implementováno jako mobilní aplikace pro platformu Android za použití moderních nástrojů a postupů pro vývoj aplikací. Aplikace byla implementována pomocí programovacího jazyka Kotlin, sestavovacího systému Gradle, verzovacího systému Git a knihoven široce používaných pro vývoj Android aplikací.

Řešení bylo otestováno pomocí jednotkových, instrumentálních a uživatelských výsledků. Všechny realizované funkce prošly těmito testy s uspokojivými výsledky. Na základě retrospektivní analýzy vývoje aplikace a jejího používání byly navrženy možné budoucí vylepšení.

Oproti stávajícím řešením přináší tato práce jednoduchý uživatelský design se všemi funkcemi snadno dostupnými. Zároveň tato práce může sloužit jako studie moderních architektur pro vývoj Android aplikací.

Literatura

[1] *Understand the Activity Lifecycle* [online]. Google. Activity Lifecycle. Dostupné z: https://developer.android.com/guide/components/activities/activity-lifecycle.

[2] *Save data in a local database using Room* [online]. Google. Android Room. Dostupné z: https://developer.android.com/training/data-storage/room.

[3] DiMarzio, J. F. *Beginning Android Programming with Android Studio*. John Wiley and Sons, Inc., 4th edition, 2017. ISBN 978-1-118-70559-9.

[4] *About FCM Messages* [online]. Google. Cloud Messaging. Dostupné z: https://firebase.google.com/docs/cloud-messaging/concept-options.

[5] *Fragments* [online]. Google. Fragment Lifecycle. Dostupné z: https://developer.android.com/guide/components/fragments.

[6] Heller, M. *What is Kotlin? The Java alternative explained* [online]. InfoWorld. Kotlin Programming Language. Dostupné z: https://www.infoworld.com/article/3224868/what-is-kotlin-the-java-alternative-explained.html.

[7] Horton, J. *Android Programming for Beginners*. Packt Publishing Ltd., 2015. ISBN 978-1-78588-326-2.

[8] *JetBrains s.r.o.* [online]. JetBrains. JetBrains Company. Dostupné z: https://www.jetbrains.com/.

[9] Martin, R. C. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall Press, 1st edition, 2017. ISBN 0134494164.

[10] *Clean Architecture by Uncle Bob: Summary and review* [online]. Gil Vegliach, 2018. [cit. 2018/09/08]. SOLID Principles. Dostupné z: https://clevercoder.net/2018/09/08/clean-architecture-summary-review/#code-principles.

[11] *Application Structure* [online]. Mattia Battiston, 2016. [cit. 2016/09/08]. Clean Application Structure. Dostupné z: https://github.com/mattia-battiston/clean-architecture-example.

[12] *Most Popular Android Versions In May 2019* [online]. Charanjeet Singh, 2019. [cit. 2019/11/07]. Android version distribution. Dostupné z: https://fossbytes.com/most-popular-android-versions-always-updated/.

Přílohy

Testovací scénáře

Načtení novinek

- **Předpoklad** - Uživatel si dříve nastavil obec jako "Moje město"
- **Akce** - Uživatel přejde na obrazovku "Novinky"
- **Výsledek** - Jsou načteny novinky z obce nastavené jako "Moje město"

Detail novinky

- **Předpoklad** - Na obrazovce "Novinky"jsou načteny novinky
- **Akce** - Uživatel klikne na novinku
- **Výsledek** - Je zobrazen detail novinky s titulkem, datem, popisem a tlačítkem pro otevření novinky ve webovém prohlížeči. Novinky na obrazovce "Novinky"nemá tučený nadpis.

Zobrazení novinky v prohlížeči

- **Předpoklad** - Uživatel je obrazovce "Detail novinky"
- **Akce** - Uživatel klikne v detailu novinky na tlačítko "Otevřít"
- **Výsledek** - Je otevřen webový prohlížeč s novinkou

Vyhledávání měst

- **Předpoklad** - Uživatel je obrazovce "Města"
- **Akce** - Uživatel do vyhledávacího pole začne vyplňovat název měst
- **Výsledek** - Pod vyhledávacím polem se zobrazí našeptávač, aktualizovaný podle právě zadaného textu

Historie vyhledávání měst

- **Předpoklad** - Uživatel dříve vyhledal a kliknul na město v našeptávači
- **Akce** - Uživatel přejde na obrazovku "Města"
- **Výsledek** - Historie vyhledávání obsahuje dříve vyhledané obce

Nastavení obce jako "Moje město"

- **Předpoklad** - Uživatel se nachází na obrazovce "Detail města"
- **Akce** - V kontextovém menu uživatel vybere tlačítko s ikonou domečku
- **Výsledek** - Obec je nastavena jako "Moje město"a v pravém horním znaku města se objeví zlatá hvězdička

Přepínání mapy

- **Předpoklad** - Uživatel se nachází na obrazovce "Mapa"
- **Akce** - Uživatel klikne na ikonu rozbalené mapy v levém horním rohu obrazovky
- **Výsledek** - Typ mapy je přepínán mezi terénní, satelitní a normální

Výběr hlášení na mapě

- **Předpoklad** - Uživatel je na obrazovce "Mapa"
- **Akce** - Uživatel klikne na hlášení na mapě - položky v červeném poli
- **Výsledek** - Uživatel je navigován na detail hlášení

Výběr místa zájmu na mapě

- **Předpoklad** - Uživatel je na obrazovce "Mapa"
- **Akce** - Uživatel klikne na místo zájmu na mapě - položky v zeleném poli
- **Výsledek** - Uživatel je navigován na detail místa zájmu na mapě

Nastavení položek na mapě

- **Předpoklad** - Uživatel je na obrazovce "Nastavení"
- **Akce** - Uživatel změní stavy zaškrtávacích polí v nastavení
- **Výsledek** - Viditelnost položek na mapě se změní podle nastavení uživatele

Seznam pojmů a zkratek

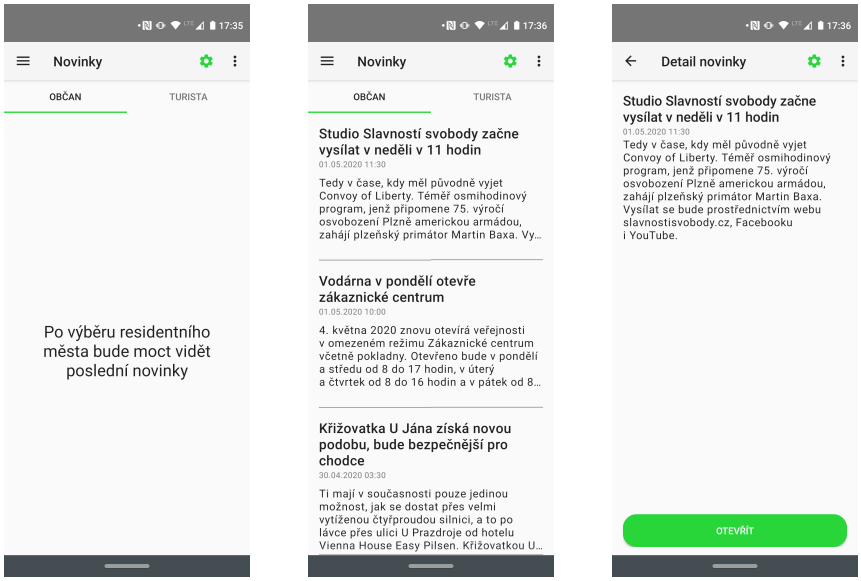
- BE - Back-End, oddělená vrstva aplikace spravující data a přístup k nim
- IDE - Integrated Development Environment, výojové prostředí, zpravidla textový editor
- JVM - Java Virtual Machine, virtuální stroj, umožňící spouštění aplikací napsaných v jazyce Java
- ORM - Objektově Relační Mapování, modelování databázových tabulek pomocí objektů
- REST API - Representational State Transfer, rozhraní komunikující pomocí HTTP volání
- SDK - Software Development Kit, soubor nástrojů umožňující vývoj software pro danou platformu
- SŘBD - Systém Řízení Báze Dat, softwarové vybavení, umožňující práci s databází
- DVM - Dalvik Virtual Machine, běhové prostředí pro aplikace na Android
- DEX - Dalvik Executable, soubory spustitelné v prostředí DVM
- DI - Dependency Injection, vložení závislosti na komponentě do komponenty jiné, zpravidla prováděno vkládáním pomocí konstruktoru objektu

Uživatelská příručka

Novinky

Novinky je možné si prohlížet na úvodní obrazovce aplikace. Novinky na kartě Občan je možné prohlížet v případě, že si uživatel dříve nastavil obec jako rezidentní. Na kartě Turista jsou zobrazovány novinky z obce, která je uživateli nejblíže. Tyto novinky jsou zobrazeny jen v případě, že uživatel povolil přístup k jeho poloze.

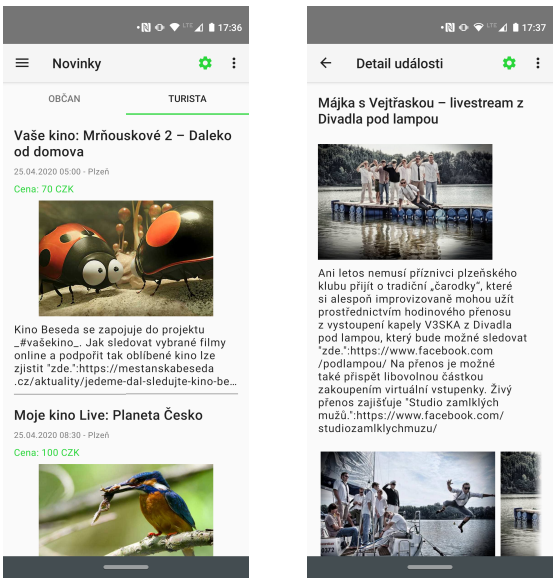
Po kliknutí na novinku je uživatel navigován na její podrobnější detail, ze kterého se může dále navigovat na webové stránky novinky. V případě novinek pro turisty je uživatel navigován na stránky novinky okamžitě.



Obrázek 9.1: Novinky z obce

Kulturní akce

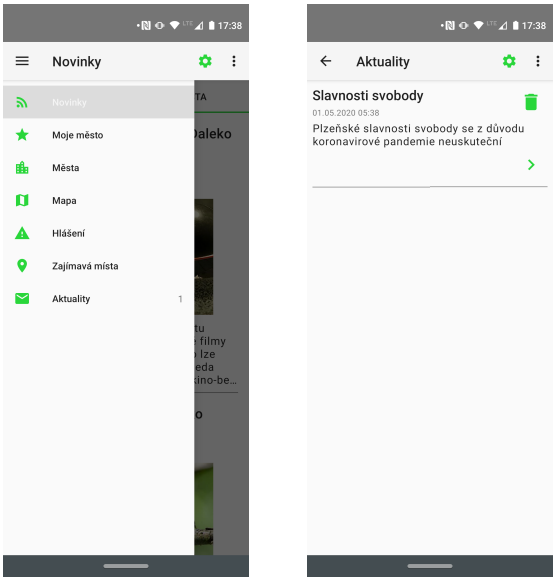
V případě, že má uživatel připojení k internetu, bude mu na úvodní obrazovce na kartě „Turista“ zobrazen seznam kulturních akcí a jejich informace. Detail akce obsahuje podrobnější popis a odkaz na oficiální stránky pro zakoupení vstupenek. Akce je možné filtrovat dle místa konání v nastavení aplikace.



Obrázek 9.2: Kulturní akce

Aktuality

V případě, že uživatel dříve nastavil obec jako „Moje město", je přihlášen k odběru aktualit, které si může později pročíst a mazat dle potřeby. V navigačním menu je možné vidět počet právě uložených aktualit.

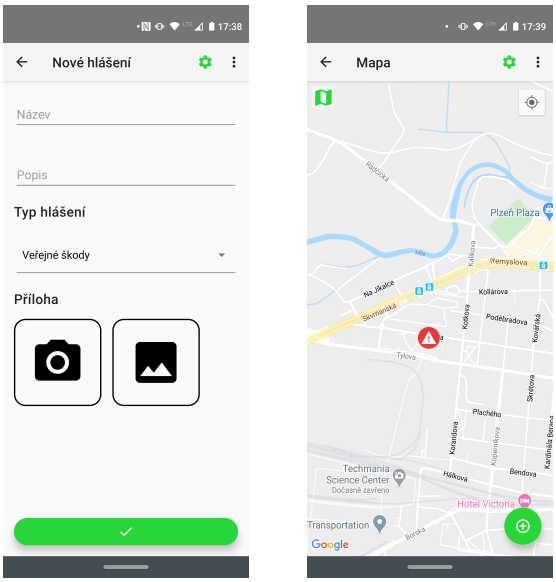


Obrázek 9.3: Aktuality

Založení hlášení podnětu

V případě, že uživatel dříve poskytl souhlas s přístupem k jeho poloze, může na mapě založit nové hlášení kliknutím na ikonu tužky v kontextovém menu. Jiný způsob založení hlášení na mapě je dlouhým kliknutím na místo na mapě. Uživatel je následně navigován do formuláře pro založení hlášení.

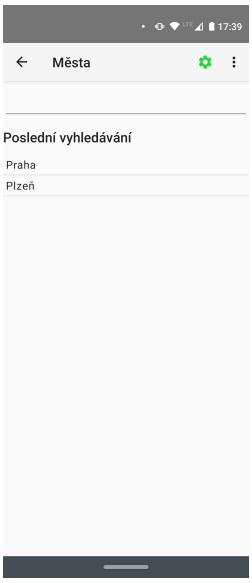
Je nutnost vyplnit hlášení, popis hlášení, kategorii a přidat přílohu. Příloha může být přiložena vyfocením kamerou nebo výběrem z fotogalerie.



Obrázek 9.4: Hlášení podnětů

Vyhledávání obcí

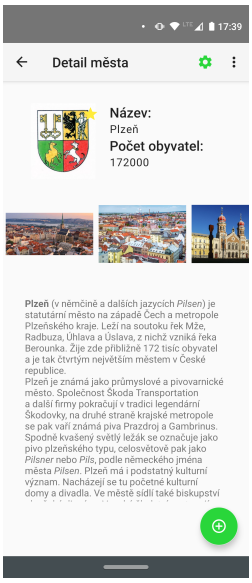
Uživatel může vyhledávat obce na obrazovce Města. Při vstupu do vyhledávacího pole se uživateli bude zobrazovat našeptávač s dostupnými obcemi. Po kliknutí na obec je uživatel navigován na detail obce. Historie vyhledávání je uchovávána a následně zobrazena po vyhledávacím polem.



Obrázek 9.5: Vyhledávání obcí

Detail obce

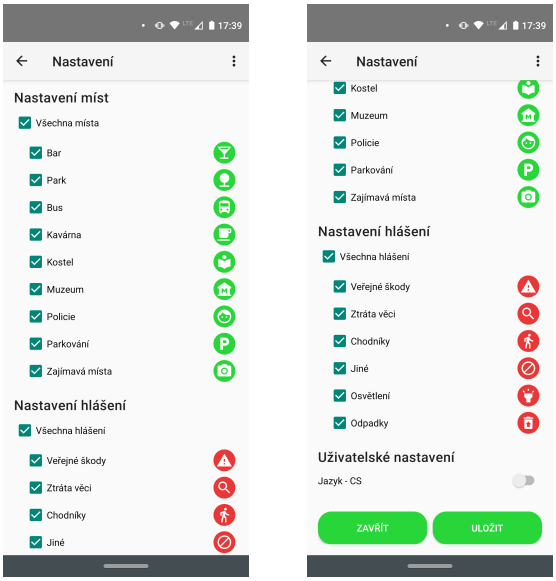
V detailu obce jsou načteny podrobnější informace o obci - počet obyvatel, popis, její znak. Uživatel má možnost v kontextovém menu navigovat se na webové stránky obce, její polohu na mapě, novinky v obci nebo si nastavit obec jako rezidentní. Po nastavení obce jako rezidentní budou uživateli na úvodní obrazovce zobrazovány novinky z dané obce. V případě, že je obec nastavená jako rezidentní, je tato skutečnost indikována zlatou hvězdou v pravém horním rohu znaku města.



Obrázek 9.6: Detail obce

Nastavení

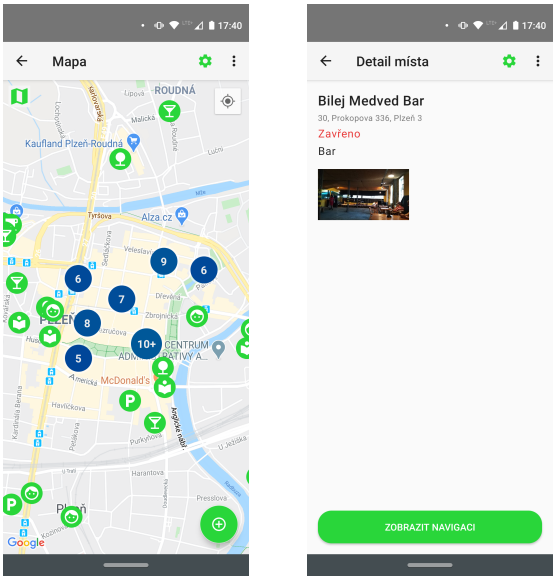
V nastavení aplikace má uživatel možnost změnit nastavení zobrazovaných položek na mapě a přepínat mezi českou a anglickou lokalizací. Změnit nastavení mapových položek je možné pomocí zaškrtávacích polí jednotlivých kategorií, jazyk je možné změnit pomocí posuvníku.



Obrázek 9.7: Nastavení aplikace

Místa zájmu

Místa zájmu jsou na mapě znázorněna v zelených polích s patřičnou ikonou, která je determinována kategorií. Kliknutím na dané pole je možné si zobrazit detail místa. Detail místa obsahuje název, adresu, stav otevřeno/-zavřeno a kategorie, do které dané místo patří.



Obrázek 9.8: Místa zájmu