Pedagogická fakulta
Faculty of Education

Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

Jihočeská univerzita v Českých Budějovicích

Pedagogická fakulta

Katedra informatiky

Tvorba Open Source aplikací pro OS Android v jazyce Kotlin

Open Source applications development for OS Android in Kotlin

Bakalářská práce

Vypracoval: Jakub Vojáček

Vedoucí práce: PaedDr. Petr Pexa, Ph.D.

České Budějovice 2021

Zadání bakalářské práce

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH

Pedagogická fakulta

Akademický rok: 2019/2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jakub VOJÁČEK**
Osobní číslo:	**P18486**
Studijní program:	**B7507 Specializace v pedagogice**
Studijní obor:	**Informační technologie a e-learning**
Téma práce:	**Tvorba Open Source aplikací pro OS Android v jazyce Kotlin.**
Zadávající katedra:	**Katedra informatiky**

Zásady pro vypracování

Cílem bakalářské práce bude rozbor tvorby Open Source aplikací pro mobilní zařízení s OS Android v programovacím jazyce Kotlin. Kotlin je staticky typovaný objektově orientovaný programovací jazyk běžící nad Java Virtual Machine, s možností kompilace do JavaScriptu. Je navržen pro interoperabilitu s knihovnami Javy, na některých knihovnách jádra dokonce závisí, ovšem Google v roce 2017 označil Kotlin jako oficiální jazyk pro Android vývoj místo původně používaného jazyka Java. V bakalářské práci bude podrobně zpracován samotný vývoj aplikace v jazyce Kotlin, výstupem práce bude sada praktických příkladů, menších aplikací, a hlavní, větší, aplikace Katedra informatiky PF JU. Aplikace bude otestována na dostupných platformách (telefon, tablet) i na vícero verzích OS Android, součástí práce bude i publikování hotové aplikace na Google Play.

Rozsah pracovní zprávy:	**40**
Rozsah grafických prací:	**CD ROM**
Forma zpracování bakalářské práce:	**tištěná**

Seznam doporučené literatury:

1. Kotlin. ITnetwork.cz – Ajťácká sociální síť a materiálová základna pro C#, Java, PHP, HTML, CSS, JavaScript a další. [online]. [cit. 26.03.2020]. Dostupné z: https://www.itnetwork.cz/kotlin
2. Android Developers. Develop Android apps with Kotlin [online]. [cit. 26.03.2020]. Dostupné z: https://developer.android.com/kotlin
3. Kotlin Programming Language. Kotlin Programming Language [online][cit. 26.03.2020]. Dostupné z: https://kotlinlang.org
4. Google Codelabs. Google Codelabs [online][cit. 26.03.2020]. Dostupné z: https://codelabs.developers.google.com
5. Developing Android Apps with Kotlin | Udacity. Learn the Latest Tech Skills; Advance Your Career | Udacity [online].[cit. 26.03.2020]. Dostupné z: https://www.udacity.com/course/developing-android-apps-with-kotlin– ud9012

Vedoucí bakalářské práce: **PaedDr. Petr Pexa, Ph.D.**
Katedra informatiky

Datum zadání bakalářské práce: **5. dubna 2020**
Termín odevzdání bakalářské práce: **30. dubna 2021**

L.S.

doc. RNDr. Helena Koldová, Ph.D.
děkanka

doc. PaedDr. Jiří Vaníček, Ph.D.
vedoucí katedry

V Českých Budějovicích dne 5. dubna 2020

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne .....

.....

Jakub Vojáček

Anotace

Cílem bakalářské práce bude rozbor tvorby Open Source aplikací pro mobilní zařízení s OS Android v programovacím jazyce Kotlin. Kotlin je staticky typovaný programovací jazyk běžící nad Java Virtual Machine, s možností kompilace do JavaScriptu. Je navržen pro interoperabilitu s knihovnami Javy, na některých knihovnách jádra dokonce závisí, ovšem Google v roce 2017 označil Kotlin jako oficiální jazyk pro Android vývoj místo původně používaného jazyka Java. V bakalářské práci bude podrobně zpracován samotný vývoj aplikace v jazyce Kotlin, výstupem práce bude sada praktických příkladů, menších aplikací, a hlavní, větší, aplikace Katedra informatiky PF JU. Aplikace bude otestována na dostupných platformách (telefon, tablet) i na vícero verzích OS Android, součástí práce bude i publikování hotové aplikace na Google Play.

Klíčová slova

Kotlin, OS Android, Vývoj aplikace, Open Source aplikace

Abstract

The aim of my bachelor's thesis will be to analyze the creation of Open Source applications for mobile devices with Android OS in the programming language Kotlin. Kotlin is a statically typed programming language running on top of a Java Virtual Machine, with the option of compiling into JavaScript. It is designed for interoperability with Java libraries, it even depends on some kernel libraries. In 2017 Google designated Kotlin as the official language for Android development instead of the originally used Java language. The bachelor's thesis will elaborate the development of the application in the language of Kotlin, the output of the work will be a set of practical examples, smaller applications, and the main, larger, application Department of Informatics PF JU. The application will be tested on available platforms (phone, tablet) and on several versions of the Android OS, part of the work will be publishing the finished application on Google Play.

Keywords

Kotlin, OS Android, Application development, Open Source application

Poděkování

Rád bych poděkoval vedoucímu mé bakalářské práce panu PaedDr. Petru Pexovi, Ph.D. za odborné vedení práce, rady, ochotu při vedení mé práce.

Děkuji.

Obsah

1 Úvod

Operační systém Android je nejrozšířenějším mobilním operačním systémem, což přináší i svá úskalí. Vývoj mobilních aplikací je pro tuto platformu o to komplikovanější, neboť vývojář musí brát v potaz různorodost specifikací jednotlivých mobilních telefonů, na rozdíl od konkurenčního iOS firmy Apple, který žije pouze v telefonech s nakousnutým jablkem na zádech, tudíž je hardware i software standardizován. Naopak výhodou vývoje aplikací pro OS Android je množství uživatelů, na které může vývojář svou aplikaci zacílit. Zkrátka vše má svůj rub i líc.

1.1 Cíle práce

Cílem mé bakalářské práce bude seznámit čtenáře s programováním aplikací pro OS Android v jazyce Kotlin. V teoretické části bych chtěl čtenáři představit základní konstrukce jazyka Kotlin, přiblížit základy tvorby Android aplikace tj. základní konstrukce, widgety, layouty, ukázat užitečné knihovny pro tvorbu aplikací a porovnat Kotlin s Javou. V praktické části využiji teoretické znalosti ve dvou menších ukázkových aplikacích a v jedné vetší aplikaci. Chtěl bych, aby čtenář po přečtení mé práce, byl schopný naprogramovat svou vlastní jednoduchou aplikaci pro OS Android.

1.2 Východiska práce

Od roku 2017 se stal Kotlin oficiálním jazykem pro tvorbu aplikací pro OS Android a od roku 2019 byl jmenován preferovaným jazykem. Vedoucí vývoje Kotlinu, Andrey Breslav, označil tento programovací jazyk jako průmyslově spolehlivý objektově orientovaný jazyk, lepší než Java, se kterou je plně interoperabilní. Tato vlastnost značně zjednodušuje vývojářům přechod z Javy na Kotlin. Syntaxe Kotlinu je výrazně zjednodušená, odstraňuje nejčastěji opakované části kódu např. psaní středníků, modifikátor viditelnosti „public“, defi-

nování datových typu a naopak přidává nové možnosti práce např. s datovými kolekcemi (forEachIndexed, any, filter, ...).

1.3 Metody práce

V úvodu teoretické části popíši vývoj OS Android od verze 5.0, základní vlastnosti Kotlinu či jeho základní konstrukce. Dále čtenáři představím zkušenosti, pocity, dojmy z práce v programovacím jazyce kotlin z pohledu studenta JU, který programoval pouze v Javě nebo C#. Představené poznatky demonstruji na aplikacích, které budou součástí bakalářské práce.

2 OS Android

Android je operační systém firmy Google založený na jádře Linux Kernel a je vedený jako Open Source software[1].[1] Android je nejrozšířenější mobilní operační systém, který v roce 2017 obsluhoval neuvěřitelných 85% telefonů.[2] Dle výsledků StatCounter je k srpnu roku 2020 procento zastoupení OS Android v mobilních telefonech o 11% nižší.[1]

Tento operační systém se nevyskytuje pouze na mobilních telefonech, ale také na tabletech, televizích, set-top boxech či hodinkách. Samozřejmě na každé platformě se Android vyskytuje v různých modifikacích.[2]

Různé modifikace Androidu nenalezneme pouze na odlišných platformách, ale i v rámci jedné, např. na mobilních telefonech. Většina výrobců mobilních telefonů s operačním systémem Android přidává do OS svou nadstavbu. Výrobce přidá do své verze OS Android svůj vzhled, nové funkce, nové chování či nová vylepšení, která by uživateli měla poskytnout lepší zážitek z používání jejich mobilního telefonu. Některé nadstavby jsou natolik výrazné a zásadně mění operační systém, že je výrobce již neoznačuje jako Android, ale svým názvem. Příkladem může být OxigenOS od čínského výrobce OnePlus.[1]

Na trhu můžeme najít i výrobce, kteří do systému nepřidávají žádnou nadstavbou nebo pouze v minimálním množství. Mezi mobilní výrobce s minimální nebo žádnou nadstavbou se řadí např. mobilní telefony od Motoroly či telefony přímo od společnosti Google.[1]

2.1 Vývoj OS Android

Prapočátek Androidu se datuje do dob, kdy pojem chytrý telefon lidské ucho ještě neslyšelo a do oznámení, dnešního největšího konkurenta, prvního Apple iPhone s iOS zbývaly roky. Android Inc byl založen v Kalifornii Richem Minerem, Nickem Searsem, Chrisem Whitem a Andym Rubinem v říjnu 2003. Prvotním plánem OS Android bylo vylepšit operační systém digitálních foto-

[1]Software s otevřeným zdrojovým kódem

aparátů. Ale už v té době byl trh s digitálními fotoaparáty nepříliš úspěšný a proto se OS Android přeorientoval trh s mobilními telefony s myšlenkou vyrábět „mnohem chytřejší mobilní zařízení, které budou lépe reagovat na uživatelovu polohu, záliby a preference.“[3]

V roce 2005 vstoupila do hry společnost Google a koupila původní společnost Android. Google přišel s rozhodnutím, použít v operačním systému Linuxové jádro, což dovolilo nabízet Android výrobcům mobilních telefonů zcela zdarma. Operační systém jako takový zcela zdarma je, ovšem výrobci platí společnosti Google za jejich služby, které jsou obsaženy v operačním systému včetně Google aplikací.[3]

O 14 let později, v roce 2019, se o důležitosti Google služeb v OS Android přesvědčuje na vlastní kůži čínský Huawei, který se objevil z důvodu bezpečnostních rizik na černé listině Spojených států. Tato situace dokazuje, že OS Android je zdarma, může ho využívat kdokoliv, ale bez Google služeb je víceméně nepoužitelný pro drtivou většinu uživatelů mobilních telefonů, tedy alespoň prozatím. Díky absenci Google služeb chybí v nových mobilních telefonech Huawei a jejich dceřiných společností mj. Google play vč. všech Google aplikací - Gmail, YouTube, Chrome, ...[4] Absence aplikací by byla to nejmenší, v těchto telefonech chybí i důležitá bezpečnostní vrstva Google.[5]

Ve třetím čtvrtletí roku 2020 hlásí Huawei pokles prodejů v západní Evropě téměř o 60%, tím klesl i jeho podíl na trhu s mobilními telefony z 20% na 8,8%. Nutno podotknout, že tato čísla jsou z dat ze západní Evropy, ale lze předpokládat, že ve zbytku Evropy bude situace podobná.[6]

Mobilní telefony s operačním systémem Android spatřily světlo světa o rok později než Apple iPhone, tedy v roce 2008. Jednalo se o mobilní telefon HTC Dream. Od té doby prošel operační systém velkou proměnou, z nichž nejdůležitější je pro většinu vývojářů Android 5.0.[3]

2.1.1 Android 5

Android 5, Lollipop, který vyšel na konci roku 2014, se stal zatím největší aktualizací operačního systému Android. Přinesl stovky nových API[2] pro vývojáře a rozšířil tak svou působnost z mobilních telefonů, tabletů nebo chytrého příslušenství (např. chytré hodinky) do televizí či aut.[7]

Nejviditelnější novinkou, kterou tato verze Androidu přináší je Material design. Jedná se o nový způsob, jak v aplikacích jednoduše vytvářet UI[3]. Přináší 3D view - lze nastavit osu z a tím docílit „zvednutí“ prvku nad ostatní a dynamicky zobrazit stíny, a to dokonce i při pohybu prvku.[7]

Přidávané prvky jsou již automaticky animované a měly by na uživatele působit přirozeně.[8] Animace jsou plynulé díky novému procesnímu vláknu zvaného RenderThread. Material design přidává přechody mezi jednotlivými aktivitami včetně možnosti sdílení vizuálních prvků napříč aktivitami.[7]

Pro vývojáře aplikací se nejdůležitější novinkou stalo přidání widgetu RecyclerView.[8]

Tento update samozřejmě přinesl spoustu dalších novinek a vylepšení, např. vylepšení výkonu, úprava chování notifikací či multitaskingu . . . [7]

2.1.2 Android 5.1

Aktualizace Androidu na verzi 5.1 byla jedna z těch menších. Pro uživatele telefonů s touto verzí operačního systému Android byla přidána softwarová podpora dvou SIM karet. Z pohledu vývojářů byly nahrazeny HTTP třídy třídou URLConnection.[9]

2.1.3 Android 6

Na konci roku 2015 byl vydán Android 6, Marshmallow, který nově přináší možnost spravovat oprávnění aplikace přímo za jejího běhu. Touto změnou

[2] Application Programming Interface - funkce, třídy či protokoly, které poskytuje vývojář programu/knihovny pro práci vývojáři, který chce jeho produkt využívat

[3] User Interface - vizuální návrh prvků, celků, animací a interakcí

by měl mít uživatel větší kontrolu nad používanou aplikací tím, že její práva přijímá, nebo odmítá až za jejího běhu, nikoliv před. Vývojáři by se uživatele měli ptát na oprávnění až ve chvíli, kdy jsou zapotřebí ke správnému fungování aplikace.[10]

Další novinkou je softwarová podpora čtečky otisku prstu, kterou uživatel může používat k odemčení telefonu, k přístupu do zamčených aplikací...[11]

2.1.4 Android 7

Ve verzi Android 7, Nougat, byla představena nová žádaná novinka v multitaskingu a to zobrazení dvou aplikací najednou tzn. rozdělení obrazovky na dvě okna a v každém z oken zobrazení jiné aplikace. Bylo opět předěláno zobrazování notifikací, byla přidána možnost zablokování konkrétního telefonního čísla.[12]

2.1.5 Android 7.1

Kromě nových emoji nebo posílání obrázků přímo z klávesnice byla přidána hlavně možnost tzv. „zkratek“. Ve chvíli, kdy uživatel dlouze klikne na aplikaci, v případě, že vývojář tuto možnost přidá do aplikace, se zobrazí menu se zkratkami, které uživatele přesunou přímo do konkrétní části aplikace a uživatel ji již nebude muset zdlouhavě „proklikávat“, aby se na danou obrazovku dostal.[13]

2.1.6 Android 8

V roce 2017 přišla aktualizace na Android 8, Oreo, která přináší novinku, která se v konkurenčním iOS představila až o 3 roky později, a to obraz v obraze. Tato funkce umožňuje nezastavení přehrávaného videa v případě, že uživatel odejde z aplikace, ale pokračuje v přehrávání v menším okně zatímco uživatel může zařízení využívat jinak.[14]

Další novinkou je přidání Autofill frameworku, který umožňuje uživateli

automaticky doplňovat údaje do formulářů, jako jsou často vyplňované jméno, příjmení, uživatelské jméno, číslo kreditní karty.[14]

Přidání textových fontů do XML přineslo, že již není zapotřebí přenášet fonty v assets, nýbrž jsou kompilovány přímo v „R“[4] a jsou automaticky dostupné jako nový resource - fonts. Tato nová funkcionalita je zpětně kompatibilní a to až s Androidem 4.0.[14]

2.1.7 Android 8.1

Android 8.1 přináší nový lehčí druh Androidu tzv. Android Go. Jedná se o Android, který je optimalizovaný pro méně výkonná, až slabá zařízení, která jsou používána lidmi po celém světě. Android Go lze používat na zařízeních, která mají 1GB paměti RAM či méně. Google Play pro tyto zařízení doporučuje primárně aplikace, které jsou optimalizované vývojáři na výše uvedené nízkonákladové telefony, ale dostupné jsou samozřejmě i ty neoptimalizované.[15]

2.1.8 Android 9

Android 9, Pie, přináší podporu Wi-Fi protokolu zvaného *Wi-Fi Round-Trip-Time* (dále jen RTT), který umožňuje navigaci ve vnitřních prostorech. Zařízení měří vzdálenosti k nejbližším Wi-Fi přístupovým bodům schopných RTT a tím se orientuje v prostoru.[16] První funkční interierovou navigací v České republice byla aplikace Škodův palác od společnosti Eternal.[17]

V roce 2018 se poprvé na telefonech objevují tzv. výřezy v displeji pro přední fotoaparát nebo sluchátko hovoru a s nimi přichází i podpora OS Android.[16]

2.1.9 Android 10

Android 10 přidává možnost „chytré odpovědi“ přímo z notifikace. Při zobrazení notifikace systém zanalyzuje obsah zprávy a snaží se navrhnout možnosti

[4] R - nabídka zdrojů, která se zobrazí po napsání „R.“ do kódu v Android Studiu

odpovědi. V případě zjištění adresy ve zprávě se objeví možnost její zobrazení na mapě. Vývojáři mají možnost přidat své možnosti „odpovědi“ do notifikace jejich aplikací.[18]

V této aktualizaci si uživatel může na telefonu nastavit tmavý režim, který pomáhá šetřit baterii zařízení. Tmavý režim může být zapnutý pořád, pouze při zapnutém režimu nízké spotřeby baterie nebo podle denní doby tzn. přes den bude telefon ve světlém režimu a přes noc v tmavém.[18]

Nově si uživatel může vypnout zobrazení softwarového navigačního baru (obvykle tlačítko zpět, domů a zobrazení nedávno spuštěných aplikací) a místo něj ovládat telefon gesty.[18]

2.1.10 Android 11

Android 11 přináší obrovské množství změn a nařízení ke zlepšení uživatelova soukromí. Jedním z nich je například automatické obnovování oprávnění pro aplikaci. V případě, že uživatel nějakou dobu, uváděná doba je několik měsíců, aplikaci nepoužívá, obnoví se veškerá oprávnění pro aplikaci a aplikace si je bude muset po svém znovuspuštění vyžádat znovu.[19]

Další změnou v oprávněních pro aplikaci je tzv. dočasné povolení. Uživatel může aplikaci povolit oprávnění pouze dočasně, např. pouze pro jednou a aplikace si jej bude muset vyžádat pokaždé, když ho bude potřebovat.[19]

2.2 Základní části

2.2.1 Manifest

Každá Android aplikace musí mít svůj Manifest v kořenové složce projektu a musí být přesně pojmenovaný AndroidManifest.xml. V manifestu jsou definovány nezbytné informace o aplikace pro operační systém Android, Google Play ale i informace pro sestavení projektu.[20]

Mimo jiné jsou zde definovány jednotlivé komponenty aplikace tj. Aktivita,

services[5], broadcast receivers[6] a content providers[7]. U každé z komponent musí být popsány její základní vlastnosti jako například jméno Kotlin, nebo Java třídy. Mohou zde být popsány intent filtry, které popisují, jak může být aplikace spuštěna.[20]

V Android Manifestu jsou zapsány i jednotlivé oprávnění, které aplikace potřebuje ke svému fungování. Oprávnění se mohou vztahovat k hardwaru zařízení, k zabezpečeným částem systému nebo přístupu k jiným aplikacím. Na druhou stranu jsou zde popsány oprávnění, které musí obsahovat další aplikace, které chtějí přistupovat k jejímu obsahu.[20]

V neposlední řadě jsou v manifestu uvedeny softwarové a hardwarové požadavky na zařízení, což definuje, jakým zařízením bude aplikace v Google Play nabízena. [20]

Při používání Android Studia se AndroidManifest vygeneruje automaticky při založení aplikace s vyplněnými základními nezbytnými částmi, které jsou potřebné k prvotnímu běhu aplikace.[20]

```
1  <manifest package="com.example.myapp"
2  ... >
3      <application
4      ... >
5          <activity android:name=".MainActivity"
6          ... >
7              ...
8          </activity>
9      </application>
10 </manifest>
```

Příklad 1: Zápis aktivity (komponenty) do AndroidManifestu

[5]Komponenta, která umožňuje běh aplikace na pozadí i v případě, že uživatel již používá jinou aplikaci

[6]Komponenta, která umožňuje příjem zpráv, které jsou posílány napříč celým OS

[7]Komponenta, která pomáhá s prací s daty - ukládání, sdílení

```
<manifest ... >
    <uses-permission
      android:name="android.permission.INTERNET"/>
    ...
</manifest>
```

Příklad 2: Zápis oprávnění pro přístup k internetu do AndroidManifestu

```
<manifest
   ... >
    <uses-feature
         android:name="android.hardware.sensor.compass"
            android:required="true" />
    ...
</manifest>
```

Příklad 3: Zápis hardwarových požadavků do AndroidManifestu

2.2.2 Gradle

Gradle je balíčkovací nástroj, díky kterému mají vývojáři možnost importovat do svých aplikací knihovny nejen třetích stran. Při založení nové aplikace v Android Studiu se Gradle vytvoří sám a již obsahuje veškeré knihovny potřebné ke spuštění. Vytvoří se hned dva Gradle soubory – jeden na aplikační a druhý na projektové úrovni.[21]

Chceme-li do své aplikace přidat nový balíček s knihovnou, musíme najít aplikační Gradle soubor, ve kterém nalezneme sekci „dependencies“, do které je nutné napsat adresu knihovny. Druhý Gradle soubor se používá k přidávání repositářů. Co do jakého Gradle souboru připsat, nalezneme vždy v dokumentaci k požadované knihovně.[21]

2.2.3 Aktivita

Třída „Activity“ je nejdůležitější komponentou pro fungování aplikace pro Android. Programovací paradigma je, že aplikace se spouští s metodou „main()“. Jinak je tomu v případě programování pro Android, kde je všeobecně známá metoda „main()“ zastoupena právě instancí aktivity a jejími specifickými callbacky[8], které odpovídají životním cyklům aplikace.[22]

Tato změna vychází z rozdílného pohledu na používání mobilních a desktopových aplikací. Desktopové aplikace po spuštění začínají na stejném místě, na stejné obrazovce. Kdežto mobilní aplikace mohou pokaždé začínat na jiných místech - jinak se otevře aplikace, kterou uživatel spustí ze své domovské obrazovky, jinak stejná aplikace, která je spuštěna z notifikace, jinak aplikace, kterou spustí jiná aplikace. Aktivita by se dala označit za vstupní bod pro komunikaci aplikace s uživatelem.[22]

Aktivita poskytuje prostor na obrazovce, kde je vykresleno UI aplikace. Tento prostor obvykle vyplňuje celou obrazovku telefonu, ale může být i menší. V podstatě jedna aktivita představuje jednu obrazovku aplikace. Nejčastěji jedna z aktivit je označována jako „MainActivity“ a objeví se jako první obrazovka po spuštění aplikace.[22]

Aktivita prochází v průběhu svého života několika životními cykly (stavy). Ke správnému chodu aktivity je zapotřebí implementovat callbacky jednotlivých stavů. Nejdůležitějším callbackem je „onCreate()“, který je volán ve chvíli, kdy systém vytvoří aktivitu. Inicializují se zde jednotlivé části aktivity - vytváření view, vkládání dat do objektů, listů . . . [22]

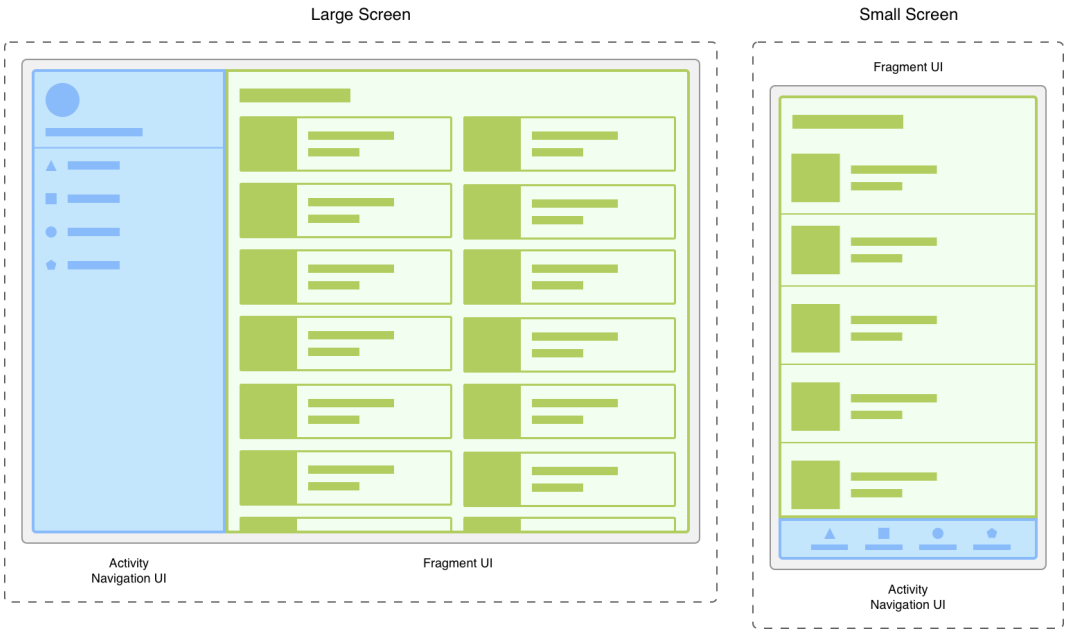
2.2.4 Fragment

Fragment reprezentuje znovupoužitelnou část UI aplikace. Fragment pracuje se svými vlastními layouty, vlastními životními cykly a může obsluhovat vlastní

[8] Callback se pokusím vysvětlit na příkladu. Třída A zavolá třídu B, aby vykonala nějakou činnost. Ve chvíli, kdy je činnost dokončena, informuje třída B třídu A pomocí Callbacku a poskytne i výsledky operace.

vstupy. Fragment není schopen existovat a pracovat sám o sobě, musí mít svého hostitele - aktivitu.[23]

Používání fragmentů dovoluje v UI aktivity například dělení obrazovky na více částí. Do aktivity je vhodné umístit veškeré prvky, které jsou shodné pro celou aplikaci např. navigační panel. Je daleko snazší ovládat či řídit UI obrazovky za běhu aplikace právě skrze fragment než přes aktivitu. Příkladem může být využití fragmentů k jinému zobrazení na zařízeních s odlišnou velikostí.[23]



Obrázek 1: Využití fragmentů podle velikosti zařízení

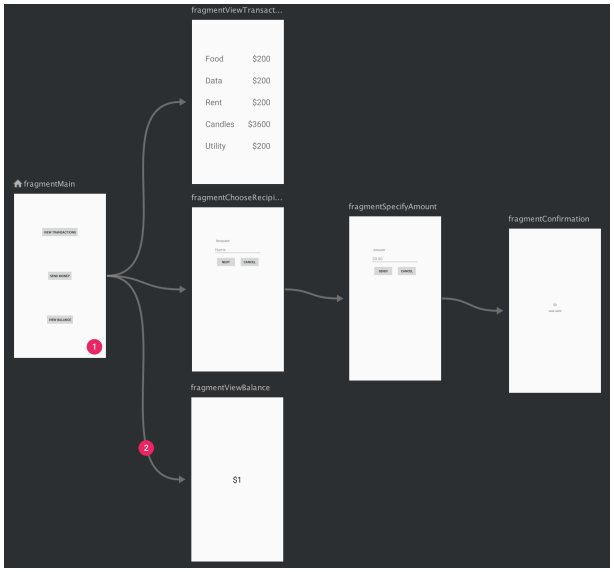
Ve chvíli, kdy aktivita projde alespoň stavem „started“ nebo vyšším, lze začít pracovat s fragmenty - lze je přidávat, nahrazovat, odebírat, . . . Vzhledem k tomu, že pod fragmentem stále žije aktivita, jsou všechny tyto změny zaznamenávány a ukládány právě do aktivity, což umožňuje se zpětně vracet v jednotlivých změnách, využití viz. Navigace.[23]

Jak již bylo řečeno, fragment je znovupoužitelný. Jeho instance mohou být použity například v jedné či ve více aktivitách, nebo dokonce v jednom či ve vícero fragmentech. Z toho důvodu by měl fragment vykonávat pouze činnosti,

které je schopen vykonávat sám bez cizí pomoci, tzn. neměl by být závislý na jiných fragmentech či aktivitách (kromě hostitelské role).[23]

2.2.5 Navigace

Navigace je nástrojem, který umožňuje uživateli přecházet v aplikaci z jedné obrazovky na druhou, tzn. z jednoho fragmentu na druhý. Navigace se skládá ze tří klíčových částí tj. Navigační graf, hostitel a kontroler. [24]



Obrázek 2: Příklad navigačního grafu

Navigační graf je XML soubor, který obsahuje veškeré informace potřebné k navigování. Zobrazuje jednotlivé body (obrazovky) a k nim vedoucí cesty, kterými se může uživatel k obrazovce „proklikat“, podobně graf v diskrétní matematice. Hostitel je kontejner, do kterého jsou zobrazovány jednotlivé obrazovky (viz obrázek 2). Kontroler je objekt, který zajišťuje samotné přechody z obrazovky na obrazovku a vkládá je do hostitele.[24]

Principiálně navigace funguje tak, že vývojář označí výchozí místo aplikace (obrazovka po spuštění) a poté cesty, kam se z výchozí obrazovky uživatel může dostat. Stejný princip je využíván u dalších obrazovek. Tímto způsobem postupně vzniká neorientovaný graf. Výchozí obrazovka je taktéž označena jako

poslední, což znamená, že „víc“ zpátky se už uživatel při zmáčknutí tlačítka „Zpět“ nemůže v aplikaci dostat. Dostane se již jen z aplikace ven.[25]

2.2.6 Layout

Layout je vizuální struktura aplikace - v jednotlivých aktivitách, fragmentech. Všechny prvky v této struktuře využívají hierarchii mezi View a ViewGroup objekty. View reprezentuje nějaký objekt na obrazovce, který uživatel vidí a většinou s ním i interaguje. Kdežto ViewGroup je neviditelný objekt, který skládá jednotlivé View nebo dokonce další ViewGroup objekty.[26]

Jednotlivé View prvky označujeme jako widgety. Využívat můžeme mnoho předdefinovaných widgetů jako například tlačítko, (editovatelné) textové pole, posuvníky nebo widgety od vývojářů, po přidání dependency do Gradle. ViewGroup objekty se označují jako Layout. V Androidu můžeme použít několik druhů layoutů, kde každý z nich rozdílně pracuje s rozložením jednotlivých widgetů.[26]

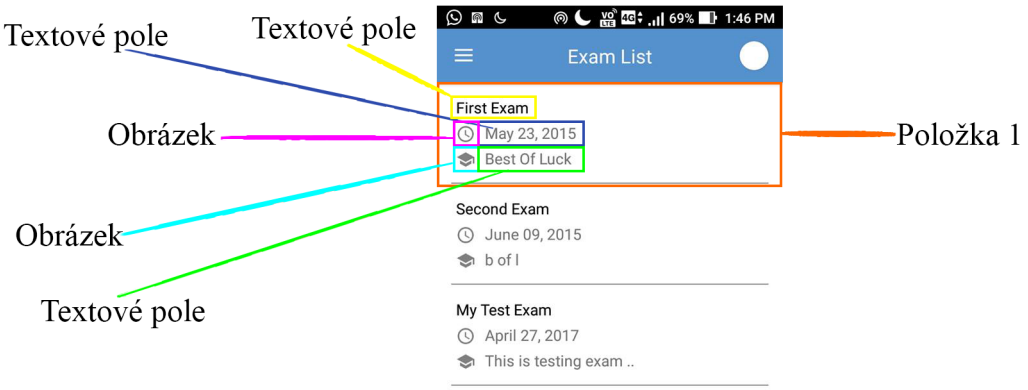
Layout může být definován v samostatném XML souboru, který má svou syntaxi pro práci s widgety a layouty, nebo může být programově vytvářen až za běhu aplikace. UI v odděleném XML souboru poskytuje snazší práci při vytváření nebo úpravě jednotlivých layoutů, manipulací s widgety díky zabudovanému editoru s okamžitým náhledem.[26]

2.2.7 RecyclerView

RecyclerView lze popsat jako jakýsi hybrid mezi widgetem a layoutem. Jedná se o prvek, do kterého se vkládají widgety (tudíž by se dal popsat jako Layout), ale zároveň se sám vkládá do Layoutu jako widget.[27]

Jedná se o nástroj, který velice snadno a efektivně je schopen zobrazit velké množství dat. Vývojář definuje jednotný Layout pro všechny položky zobrazované RecyclerView, který je dynamicky vytváří dle potřeby. Jak už název napovídá RecyclerView recykluje své jednotlivé prvky. Ve chvíli kdy jsou

„vyscrollované“ z obrazovky, zničí jejich View, které znovupoužije pro nový prvek, který se díky scrollu zobrazí na obrazovce. Tato recyklace má za následek zvýšení výkonu, vylepšuje odezvu aplikace a snižuje spotřebu energie.[27]



Obrázek 3: Příklad recyclerView

2.2.8 ViewModel

Třída ViewModel je navržena pro ukládání dat spojená s UI a životními cykly aplikace. Android framework řídí veškeré životní cykly UI kontrolerů jako jsou aktivity a fragmenty. Tento framework rozhoduje o zničení nebo znovuvytvoření UI kontrolerů v důsledku uživatelových akcí se zařízením nebo procesy samotného zařízení - jako je například otočení obrazovky.[28]

Při těchto akcích dochází ke zničení a znovuobnovení UI kontrolerů, což vede ke ztrátám dat. Jednoduchá data mohou být ukládána v SavedInstances do bundle[9] a opětovně vytažena z bundle v metodě „onCreate()“. Právě ViewModel umožňuje zachování objektů či většího množství dat jako je například bitmapa.[28]

```
1 class MyViewModel : ViewModel() {
2     private val users:
3       MutableLiveData<List<User>> by lazy {
```

[9]Balíček, do kterého se vkládají jednoduchá data ve formě klíč-hodnota. Tento balíček lze „posílat“ mezi fragmenty/aktivitami

```
4           MutableLiveData<List<User>>()}
5
6       fun getUsers(): LiveData<List<User>> {
7           return users
8       }
9   }
```

Příklad 4: Vytvoření ViewModelu

K použití ViewModelu je zapotřebí vytvoření nové třídy, která bude rozšířená právě o třídu ViewModel. V této třídě vytvoříme proměnnou typu MutableLiveData[10], jež bude datového typu, který požadujeme.[28]

Implementace našeho ViewModelu v aktivitě by mohla vypadat následovně:

```
1  override fun onCreate(savedInstanceState: Bundle?) {
2     val model: MyViewModel by viewModels()
3      model.getUsers().observe(this, Observer<List<User>>{
4        users ->
5            // Zde uz pracujeme se samotnym Listem
6          })
7      }
```

Příklad 5: Implementace ViewModelu

Vzhledem k tomu, že ViewModel pracuje asynchronně, je zapotřebí získávat z něj data též asynchronně. To umožňuje observer. Observer je zavolán vždy, když dojde ke změně dat na ViewModelu, např. při uložení nového seznamu uživatelů z příkladu 4.[28] [29]

2.2.9 SharedPreferences

SharedPreferences je jeden ze způsobů jak „trvale“ ukládat malé množství dat pomocí páru klíč-hodnota podobně jako v kolekci Map. Do SharedPreferen-

[10] Observerovatelný nositel dat

ces lze ukládat primitivní datové typy int, float, boolean a jeden neprimitivní datový typ String.[30]

Na rozdíl od SavedInstances jsou Shared Preferences uloženy v telefonu po celou dobu, kdy je aplikace nainstalovaná.[30]

SharedPreferences lze využít v nepřeberném množství situací, například k ukládání uživatelova nastavení, které je potřeba použít v různých částech aplikace nebo k ukládání uživatelova seznamu oblíbených položek v rámci aplikace, které samozřejmě musí být k dispozici i po ukončení aplikace.[30]

K využívání SharedPreferences je zapotřebí vytvořit proměnnou, díky které k datům budeme přistupovat.

```
1 val sharedPref = activity?
2                     .getPreferences(Context.MODE_PRIVATE)
```

Příklad 6: Vytvoření proměnné k přístupu do SharedPreferences

Zapsání dat do SharedPreferences může probíhat následujícím způsobem:

```
1 var newHighScore = 99
2 with (sharedPref.edit()) {
3     putInt("HIGHEST_SCORE", newHighScore)
4     apply()
5 }
```

Příklad 7: Zapsání nového nejvyššího skóre do SharedPreferences

Zápis probíhá pomoci metody put a příslušného datového typu, který ukládáme. Parametry metody jsou klíč, pod kterým hodnotu ukládáme a nakonec samotná hodnota. „apply()“ uloží data asynchronně do SharedPreferences. „apply()“ lze nahradit „commit()“, což uloží data synchronně, data jsou uložena ihned, ale tato metoda by měla být volána z hlavního procesního jádra a mohla by zastavit renderování UI, tudíž by se aplikace vizuálně „zasekla“.[31]

Naopak čtení dat z SharedPreferences může probíhat takto:

```
1 val highScore = sharedPref.getInt("HIGHEST_SCORE"), 0)
```

Příklad 8: Čtení nejvyššího skóre z SharedPreferences

Čtení probíhá obdobně jako zápis, pouze s tím rozdílem, že místo metody put voláme metodu get a příslušný datový typ. Argumenty této metody jsou klíč, pod kterým byla hodnota uložena a výchozí hodnota. Výchozí hodnota se použije v případě, že by v SharedPreferences nebyla nalezena žádná hodnota uložena pod zadaným klíčem. [31]

2.2.10 SavedInstances

Zobrazovaní různých UI v rámci jedné či více aplikací je jejich nedílnou součástí. Uživatel očekává, že po opuštění aplikace, zůstane ve stavu, ve kterém ji zanechal. Ale fungování operačního systému Android tomuto zabraňuje. Po opuštění aplikace, potažmo aktivity UI systém zničí a při opětovném spuštěním ho tvoří znovu, což ale vede k tomu, že aplikace po svém znovuspuštění nemusí vypadat stejně - již vyplněný formulář bude prázdný, na obrazovce bude zobrazeno výchozí UI aktivity, nebo fragmentu, přestože před odchodem byl uživatel již na jiném bodě aplikace.[32]

Abychom byli schopni zabránit této mezeře mezi očekáváním uživatele a chováním systému se využívá metoda „onSaveInstanceState()“, ve které vývojář uloží potřebné informace ke znovuspuštění aplikace na stejném místě, jako ji uživatel opustil.[32]

Aplikace uchovává data z této metody po celou dobu svého života, alespoň na pozadí telefonu. Ve chvíli, kdy je úplně ukončena, data z této metody „zapomíná“.[32]

Stejně jako u SharedPreferences se data ukládají pomocí páru klíč-hodnota.

3 Kotlin

Kotlin lze představit jako odlehčenou moderní verzi Javy. Kotlin je staticky typovaný programovací jazyk, který běží v běhovém prostředí Javy, což umožňuje běh aplikací napsané v Kotlinu všude, kde je nainstalovaná Java. Navzdory tomu, že se Kotlin svou syntaxí liší od Javy, jeho zkompilovaný kód je s ní plně kompatibilní, díky čemuž lze při psaní aplikací v Kotlinu využívat knihovny napsané v Javě.[33]

V roce 2017 byl Kotlin označen společností Google za oficiální programovací jazyk aplikací pro jejich operační systém Android.[34]

Kotlin je velice podobný programovacím jazykům Java, C# nebo například Swift. Od samého počátku vývoje tohoto jazyka oznámila firma JetBrains, že se jedná o Open Source, což je důvodem, že programy napsané v Kotlinu jdou spustit skoro na všech operačních systémech.[33]

Kotlin je velice všestranný jazyk, který se nepoužívá pouze k vývoji aplikací pro zařízení s operačním systémem Android. Kotlin lze využít všude, kde bychom programovali v Javě, JavaScriptu nebo dokonce díky velkému množství knihoven, některým dokonce oficiálním, k tvorbě webových aplikací.[33] [35]

3.1 Proměnné a jejich datové typy

Jak již bylo zmíněno výše, Kotlin je staticky typovaný jazyk, což znamená, že u všech proměnných by měl být explicitně uveden jejich datový typ. Úmyslně napsáno, že by proměnné měly být deklarované se svým datovým typem. Ve skutečnosti je tomu ovšem pouze v malém množství scénářů. Kompilátor si k proměnným vhodný datový typ doplní automaticky a před spuštěním datové typy zkontroluje.[33]

Příkladem může být zapsání desetinného čísla. Při zapsaní desetinného čísla bez explicitního definování datového typu, kompilátor doplní datový typ Double. Problém nastává ve chvíli, kdy chceme pracovat s datovým typem

Float. V tu chvíli je potřeba explicitně deklarovat proměnnou s datovým typem.[33]

Dynamické typovaní se na první pohled může zdát jako skvělé ulehčení práce, což bez pochyby je, ale nese s sebou i několik úskalí. Zdrojový kód nemůže být automaticky kontrolován a to může způsobit situace, že očekáváme práci celočíselnou proměnnou, ale místo toho se vrátí proměnná datového typu String.[33]

Kotlin obsahuje dva druhy proměnných - „val“ a „var“. Proměnná označená var je klasická proměnná, kterou známe například z Javy - její hodnota může být několikrát přepsána. Opakem této var proměnné je val proměnná, jejíž hodnotu nelze změnit.[33]

Kotlin je tzv. Null Safety jazyk, což umožňuje pracovat se situacemi, kde je hodnota proměnné, popř. objektu „null“ v době kompilace. Proměnná, která může nabývat nulové hodnoty musí být explicitně specifikována.[35]

```
1 val name = "Jakub␣Vojacek"
2 var index = 1
3 var bmi: Float = 23,1
4 var gradeFromTheDefenseOfTheBachelorsThesis: Double? = null
```

Příklad 9: Zápis proměnných

3.2 Metody vyššího řádu

Metody v Kotlinu jsou tzv. „prvotřídní“ tzn. že mohou být ukládány do proměnných či datových struktur, vkládány do jiných metod jako parametry či mohou být návratovou hodnotou z jiných metod vyšších řádů.[36]

Metody vyšších řádů jsou takové funkce, které přijímají jako parametr nebo jejich návratová hodnota je funkce.[36]

Pro ukládání metod do proměnných má Kotlin speciální notici, která obsahuje stejnou signalizaci jako klasická metoda.[36]

Zápis je následující:

```
1 (A, B) -> C
```

Příklad 10: Obecný zápis metody do proměnné s dvěma parametry a návratovou hodnotou

Tento příklad deklaruje, že do metody vstupují dva argumenty datového typu „A“ a „B“ a návratová hodnota je datového typu „C“. Funkce samozřejmě nemusí přijímat žádný parametr a ani nemusí mít návratovou hodnotu. [36]

Zápis takové to funkce by obecně byl:

```
1 () -> Unit
```

Příklad 11: Obecný zápis metody do proměnné bez parametrů a návratové hodnoty

Konečný zápis metody do proměnné by vypadal takto:

```
1 val onClick: () -> Unit = ...
```

Příklad 12: Zápis metody do proměnné bez parametrů a návratové hodnoty

3.3 Porovnání s Javou

Jak bylo zmíněno výše Kotlin je podobný programovacím jazykům Java, je s ní i zpětně kompatibilní, ale jejich syntaxe není stejná, je nanejvýš podobná. V tabulce 1 je několik příkladů.

	Kotlin	**Java**
Středníky	Mohou se psát, ale není to vyžadováno	Musí se psát, jinak není kód přeložitelný
Zápis proměnné	`val maxNum = 99`	`int maxNum = 99;`
Zápis veřejné metody s návratovou hodnotou	`fun getMaxNum(): Int{` `}`	`public int getMaxNum(){` `}`
Zápis privátní metody bez návratové hodnoty	`private fun doSomething() {` `}`	`public void doSomething(){` `}`

Zápis metody s int parametry	`fun multipleNum(a: int, b: int) {`
	`}`
		`public void multipleNum(int a, int b)`
		`{`
		`}`

Tabulka 1: Příklady porovnání syntaxe Kotlinu a Javy

3.3.1 Cykly

Kotlin přidává, oproti Javě, několik cyklů pro práci s kolekcemi, které výrazně zjednodušují práci. Zde je několik příkladů:

- forEachIndexed
  - odvádí stejnou práci jako forEach tj. prochází celý List, položku po položce, ale zároveň vrací pozici (index) jednotlivých položek[37]
- any
  - prochází celý list a vrací True v případě, že alespoň jedna položka splňuje podmínku zapsanou lambda výrazem[38]
  - `val isTrue = listOfTruesAndFalses.any{it == true}`
- filter
  - prochází celý list a vrací položky, které splňují podmínku zapsanou lambda výrazem[39]
  - `val listOfTrues = listOfTruesAndFalses.filter{it == true}`

3.3.2 Datové třídy

Na rozdíl od Javy, není potřeba zdlouhavě deklarovat konstruktory, proměnné, do kterých se ukládají data, jednotlivé gettry a settry, metodu toString() apod. V Kotlinu stačí vytvořit třídu a to označit jako „data“ a všechno ostatní za nás udělá kompilátor.[40]

```
1 data class User(
2         val name: String,
3         val surname: String)
```

Příklad 13: Zápis datové třídy User

3.3.3 Rozšiřující funkce

Kotlin nabízí možnost tzv. rozšiřující funkce (originálně „Extension function“ s novou funkcionalitou k již existující třídě. Extension funkce v Javě dostupné nejsou, ale přesto existuje způsob, jak tuto funkcionalitu převést do Javy - vytvoření nové třídy, která bude dědit z původní třídy. [40]

```
1 fun User.parseToJSON(): String{
2     ...
3     return ...
4 }
```

Příklad 14: Zápis rozšiřující funkce ke třídě User

```
1 val user = User("Jakub", "Vojacek")
2 val jsonString = user.parseToJSON()
```

Příklad 15: Použití rozšiřující funkce ke třídě User

4 Užitečné až potřebné knihovny

4.1 Retrofit

Retrofit je REST klient knihovna pro Kotlin i Javu, která umožňuje ověřování a interakci s API pomocí posílání síťových žádostí, využívající OkHTTP, což je nízkoúrovňový klient pro Android od stejné společnosti - Square.io. [41] [42]

Tato knihovna stahuje data v JSONu[11] nebo XML z API webu přímo do aplikace, kde s nimi lze dále pracovat, například jejich převodem do jednotlivých objektů.[42]

Retrofit je dostupný zna zařízeních, na kterých běží Android 5 a vyšší. [42]

4.1.1 Implementace

```
1 dependencies {
2 implementation 'com.squareup.retrofit2:retrofit:2.9.0'
3 }
```

Příklad 16: Implementace Retrofit knihovny do Gradle

4.1.2 Použití

Příklad použití Retrofitu bude ukázáno v praktické částí bakalářské práce v kapitole 5.1.2.

4.2 GSON

GSON je knihovna pro Javu i Kotlin, která převádí Kotlin/Java objekty do JSON souboru, či JSON soubor do příslušného Kotlin/Java objektu.[43]

[11] JavaScript Object Notation - způsob zápisu dat určených pro přenes nezávisle na platformě

4.2.1 Implementace

```
dependencies {
implementation 'com.google.code.gson:gson:2.8.6'
}
```

Příklad 17: Implementace GSON knihovny do Gradle

4.2.2 Použití

```
val users = mutableListOf(
      User("Jakub", "Vojacek"),
      User("Radek", "Zadek"))

// prevod objektu do JSONu
val jsonString = Gson().toJson(users)

// prevod z~JSONu do objektu
val listOfUsers2 = Gson().
            fromJson(
               jsonString,
               Array<Users>::class.java).toMutableList()
```

Příklad 18: Použití GSONu

4.3 Glide

Glide je knihovna, která umožňuje stažení a zobrazení obrázků, fotografií či animovaných GIFů z internetu. Pomocí Glide lze obrázkům dodatečně v aplikaci změnit rozlišení nebo je otočit. [44] Jednoduše řečeno je Glide knihovna pro práci s obrázky z internetu.

4.3.1 Implementace

```
repositories {
  google()
  jcenter()
}

dependencies {
implementation 'com.github.bumptech.glide:glide:4.11.0'
annotationProcessor 'com.github.bumptech.glide:compiler
   :4.11.0'
}
```

Příklad 19: Implementace Glide knihovny do Gradle

4.3.2 Použití

```
Glide.with(this)
      .load("http://goo.gl/gEgYUd")
      .into(imageView)
```

Příklad 20: Použití Glide

5 Praktická část

V rámci praktické části mé bakalářské práce budou představeny 3 aplikace. První dvě poslouží jako ukázkové, ve kterých budou představeny základní věci, které použiji v rozšířeném měřítku v hlavní aplikaci práce, v aplikaci Katedra informatiky PF JU. V práci budou vidět ukázky zdrojových kódů, celé zdrojové kódy jsou dostupné na přiloženém DVD.

5.1 Ukázkové aplikace

5.1.1 Hudební databáze

První ukázkovou aplikací je Hudební databáze, která jak už název napovídá, slouží k uchovávání skladeb, které uživatel do aplikace zadá. Jednotlivé skladby si může zobrazit, upravit popřípadě smazat.

Tato aplikace je zaměřena na představení úplných základů tvorby aplikací pro operační systém Android.

Příkladové zdrojové kódy lze nalézt v kapitole B Zdrojové kódy k aplikaci Hudební databáze.

Celý proces začíná vytvořením nového projektu s prázdnou aktivitou v Android Studiu. Po vytvoření se postupně začne stahovat do projektu Gradle a poté do něj všechny základní balíčky.

První užitečnou věcí je vytvoření nové třídy, jež dědí z třídy „Application()“ a ve které vytvoříme globální proměnnou „appContext“ (název může být samozřejmě jiný). Tato proměnná umožňuje přístup k aplikačnímu kontextu i ve chvíli, kdy se např. fragment nedostane do životního cyklu, ve kterém bude dostupná Aktivita, z níž je možné kontext získat. Viz zdrojový kód MusicApp.

Tuto novou třídu musíme přidat do Manifestu, aby s ní aplikace byla schopná pracovat. Viz zdrojový kód Přidání MusicApp do Manifestu.

Po vytvoření projektu nalezneme ve složce Layout předvytvořený Layout „activity_main.xml“, do kterého vložíme libovolný z vybraných kontejnerů

(hostitele), do nichž později budeme vkládat jednotlivé fragmenty, které mají být zobrazeny.

Dalším základním bodem aplikace je její navigace. Lze použít Navigační Graf, který jsem představil v kapitole 2.2.5 nebo navigaci řešit jinak, např. si vytvořit vlastní Navigační Kontroler a změny fragmentů do kontejneru řídit ručně. Obě možnosti vykonávají stejnou práci. V případě Navigačního Grafu, lze navigaci nastavit pomocí grafického rozhraní. Dalším rozdílem je, že v Navigačním Grafu se nastavují jednotlivé cesty, což v případě použití vlastního Navigačního Kontroleru není nutné. Viz zdrojový kód Navigační kontroler.

Vzhledem k tomu, že tato aplikace bude zobrazovat více fragmentů a každý z nich bude moci navigovat na další, je zapotřebí, aby každý z nich měl přístup k vytvořenému Navigačnímu Kontroleru. Kdybychom vytvořili instanci navigačního kontroleru do každého z nich, tak bychom se dopouštěli zbytečné duplicity kódu. Proto vytvoříme „BaseFragment“, jenž bude obsahovat veškeré části kódu, které jsou společné pro všechny fragmenty. Následně nově vytvořený fragment bude samozřejmě dědit z našeho „BaseFragmentu“. Instance Navigačního Kontroleru vytvoříme i v MainActivity. Viz zdrojový kód BaseFragment.

Dále vytváříme jednotlivé fragmenty a jejich Layouty dle potřeby a požadovaných funkcionalit.

Tato aplikace představuje jakousi databázi skladeb, proto musíme definovat, jaké informace o skladbě bude databáze obsahovat. Pro tento účet vytvoříme novou datovou třídu SongModel, která bude jednotlivé skladby definovat. Viz zdrojový kód SongModel.

Bohužel databáze nepracuje s takovýmto modelem, ale pracuje s tzv. „entitami“. Což je de facto stejná datová třída jako SongModel s tím rozdílem, že obsahuje Entity notaci. V mém případě rozdíl mezi oběma datovými třídami není, ale mohla by nastat situace, že bychom do Entity ukládali informace jinak, než bychom s nimi pracovali v aplikaci v podobě Modelu. Viz zdrojový kód SongEntity.

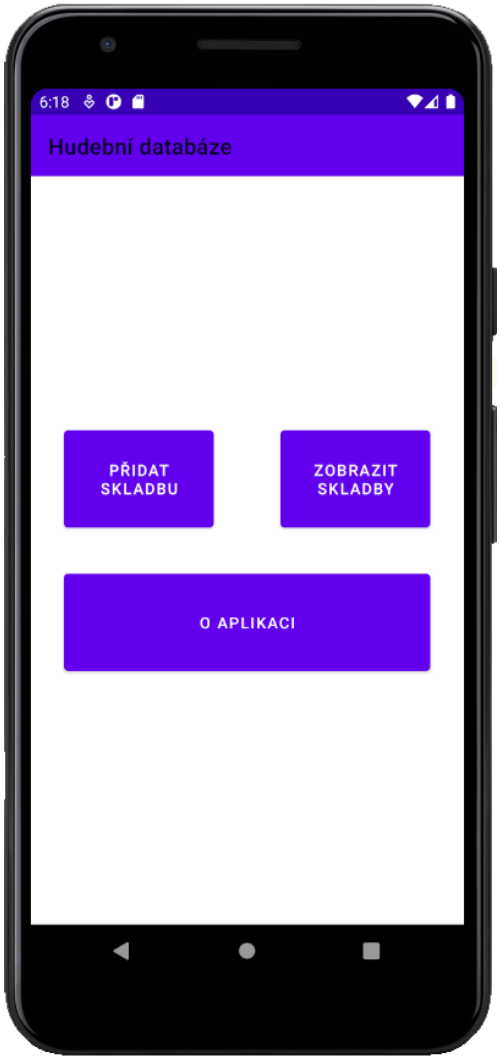
K převodu mezi SongModelem a SongEntitou vytvoříme „extension“ funkci.

Další nezbytnou částí této mini aplikace je ukládání dat do databáze. Viz zdrojový kód Database.

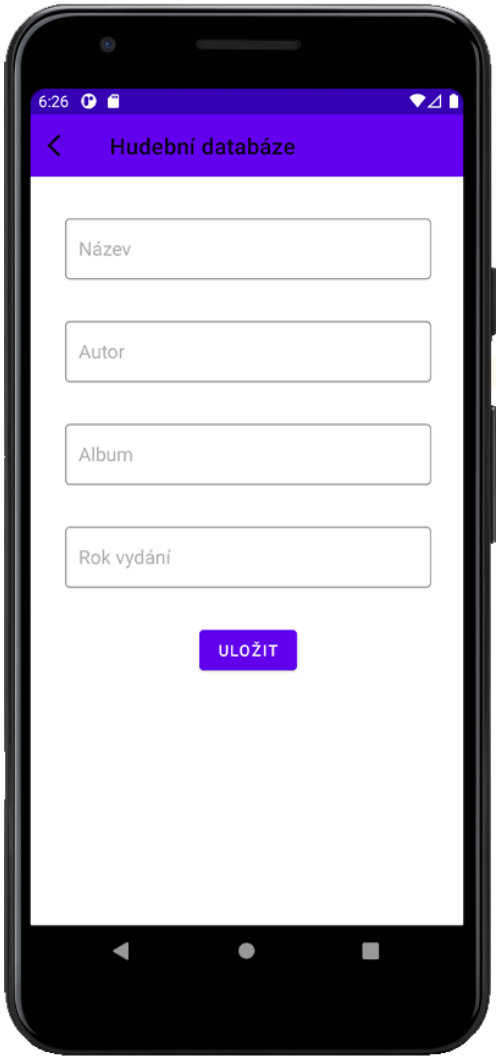
K samotné databázi je zapotřebí vytvořit třídu obsahující metody s „Query“ notací, které budou volat patřičný SQL dotaz. Viz zdrojový kód Dao.

Samotný průchod aplikací by vypadal tak, že po vstupu do aplikace si uživatel může vybrat, zda-li chce přidat novou skladbu, zobrazit všechny skladby nebo si zobrazit informace o aplikaci. Když se uživatel dostane na obrazovku s výpisem všech skladeb, tak si jednotlivé skladby může rozkliknout, čímž si zobrazí veškeré informace o skladbě. Následně ji zmáčknutím tlačítka „Upravit“ editovat nebo ji smazat z databáze.

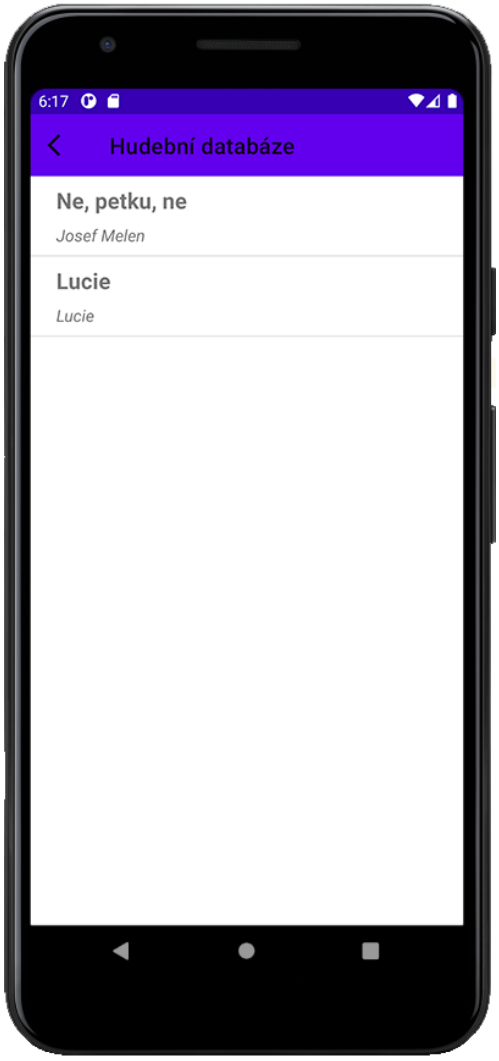
Snímky obrazovky z průchodu aplikace naleznete na následujících stranách...



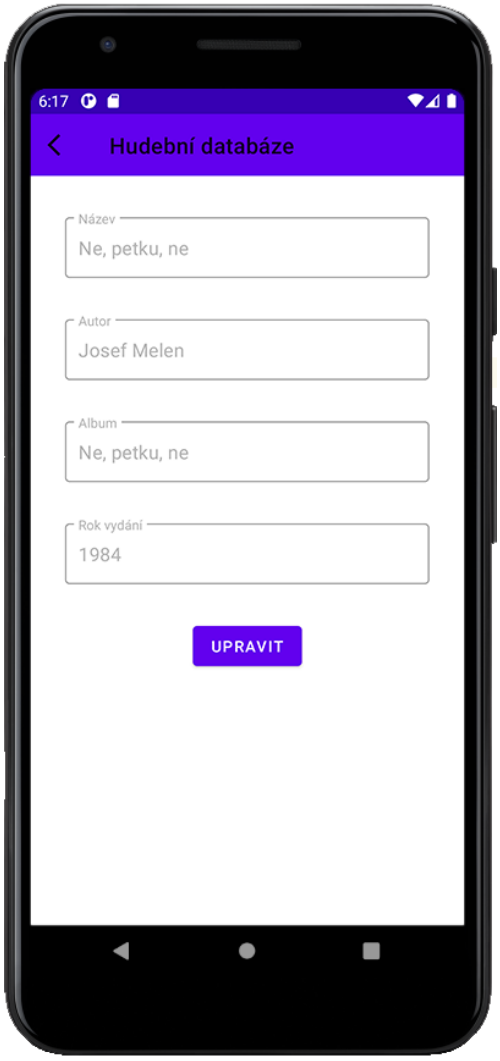
Obrázek 4: Obrazovka menu aplikace



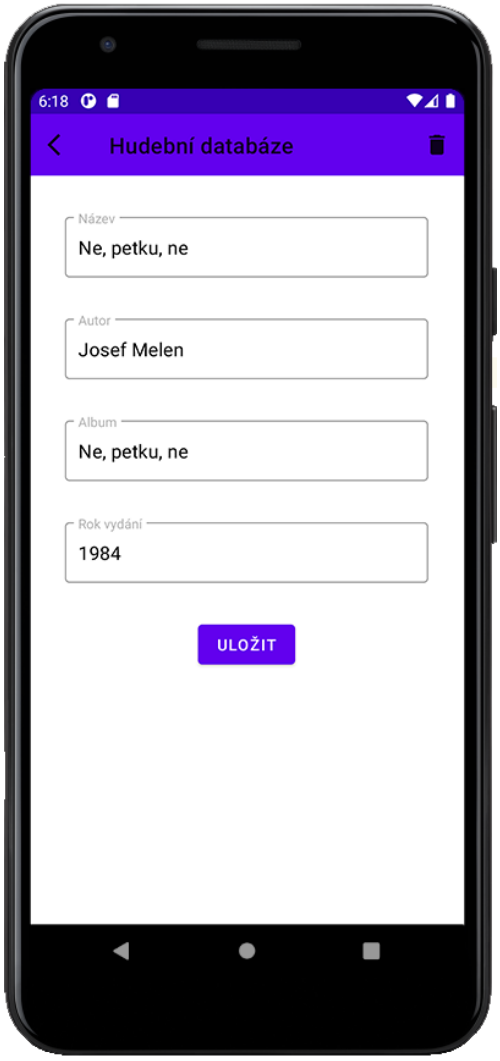
Obrázek 5: Obrazovka přidání skladby



Obrázek 6: Obrazovka seznamu skladeb



Obrázek 7: Obrazovka detailu skladby



Obrázek 8: Obrazovka úpravy skladby

Obrázek 9: Obrazovka o aplikaci

5.1.2 Listování uživateli

Druhým, posledním, ukázkovým programem je aplikace na Listování uživateli. Tato aplikace má velice jednoduchý layout, avšak jejím cílem je čtenáři představit použití knihoven Gson, Retrofit a Glide v praxi.

Fungování aplikace je velice prosté. Aplikace se připojí na server, odkud stáhne data ve formátu JSON, tato data převede do objektu.

K tomuto účelu využiji stránky `randomuser.me` poskytující API, které vygeneruje náhodného uživatele. Dle jejich slov „*Jako Lorem Ipsum, ale pro lidi.*“

Toto API poskytuje nepřeberné množství informací o náhodně vygenerovaných uživatelích, mj. jméno, příjmení, adresu nebo fotografii v podobě url adresy. Zmíněné informace o uživateli aplikace zobrazuje.

Nejprve je zapotřebí do Gradle přidat potřebné knihovny. Vzhledem k tomu, že je tato aplikace velice primitivní, není zapotřebí vytvářet žádný Navigační Kontroler, protože se vše odehrává na jedné obrazovce. Z tohoto důvodu využijeme pouze MainActivity a její Layout.

Do Layoutu vložíme potřebné widgety - dvě TextView, jeden ImageView a tlačítko.

Na MainActivity vytvoříme metodu „FetchData“, která jak už název napovídá, bude stahovat data z API. Do této metody vložíme požadavek na stažení dat z API. Tento požadavek má dvě override metody - onFailure (co se má dít, když se požadavek nezdaří) a onResponse(co se stane, když požadavek vrátí data). V onFailure metodě může být například zpráva o tom, že se požadavek nepodařilo vyřídit.

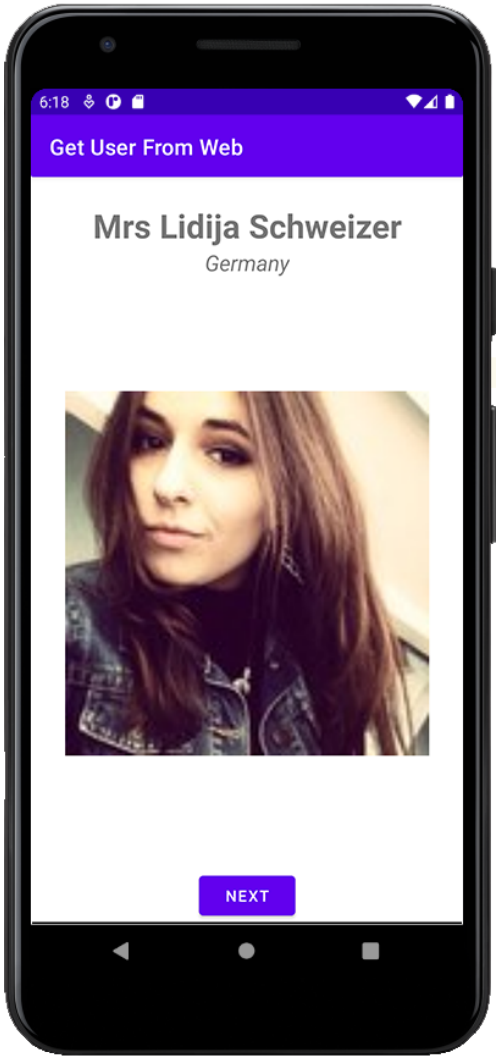
V onResponse metodě se pokusíme stažený JSON string převést pomocí GSONu do objektu, který si vytvoříme dle struktury dat. Android Studio nabízí pluginy třetích stran, které vytvoří datovou třídu podle JSON stringu. Tento plugin funguje výborně pouze na jednodušší strukturu dat. Složitější bych určitě vytvářel ručně, protože v případě, že se převod nezdaří, aplikace

spadne.

Poté už pouze stačí nastavit texty příslušným textovým polím a pomocí Glidu zobrazit obrázek. Vzhledem k tomu, že ovlivňujeme vizuální stránku aplikace, tu má na svědomí jiné vlákno, než to, které zajišťuje stahování dat z internetu, je zapotřebí tyto změny volat z UI vlákna. Na závěr už zbývá zvolit, odkud se bude metoda fetchData() volat.

Celý zdrojový kód této aktivity je dostupný v příloze C Zdrojové kódy k aplikaci pro listování uživateli.

Aplikace vypadá následovně.



Obrázek 10: Obrazovka aplikace pro zobrazení uživatelů

5.2 Aplikace Katedra informatiky PF JU

Hlavním cílem bakalářské práce bylo naprogramovat aplikaci v Kotlinu pro OS Android zobrazující data, která jsou k dispozici na webu Katedry informatiky Pedagogické fakulty Jihočeské univerzity v Českých Budějovicích. Oproti webovým stránkám má výhodu, že je k dispozici, vyjma prvního spuštění, i v případě, že zařízení nemá přístup k internetu. Při každém spuštění aplikace, je-li zařízení připojeno k internetu, se databáze aplikace aktualizuje na nejnovější data.

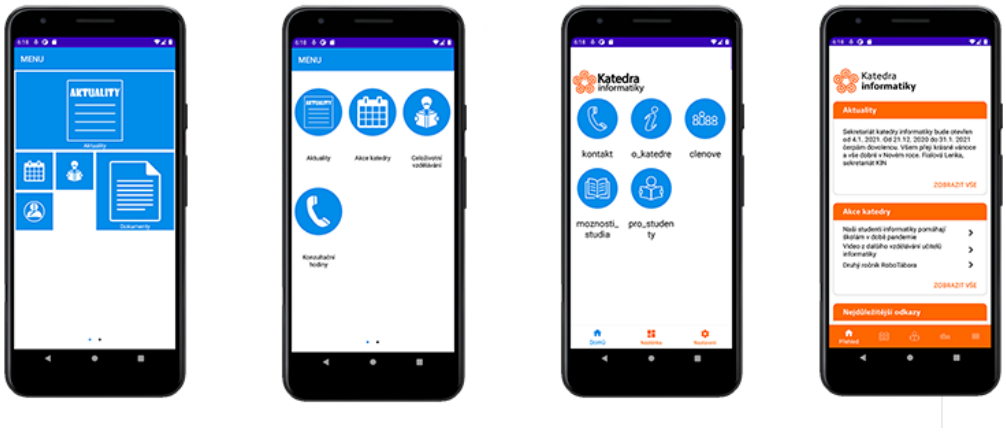
Další výhodou je, že si uživatel může přidávat své často navštěvované stránky v rámci katedry, popř. kantory do oblíbených, takže je bude mít vždy po ruce.

5.2.1 Počáteční problémy

Prvním problémem, který bylo potřeba vyřešit bylo získávání dat ze stránek katedry. Tato data nejsou v žádné centrální databázi a nebylo cílem mnou vytvořené aplikace, aby zobrazovala webové stránky katedry přes WebView[12]. Tento problém jsem vyřešil díky WordPressu, na kterém je web umístěn. WordPress umožňuje přístup k datům na webu pomocí svého API. Návratovou hodnotou tohoto API jsou data z požadované stránky v JSON formátu.

Velkým vývojem si prošel i samotný vzhled aplikace. Originální vzhled aplikace vycházel z prostředí OS Microsoft WindowsPhone. Poté následovalo několik dalších vzhledů, závěrem jsem se rozhodl katederní aplikaci vzhledově sjednotit s univerzitní aplikací StuduJU. Celou evoluci vzhledu hlavní obrazovky aplikace zobrazuje následující obrázek.

[12] Widget, který umožňuje zobrazení webových stránek



Obrázek 11: Obrazovka aplikace pro zobrazení uživatelů

První dvě verze na hlavní obrazovce zobrazovaly menu oblíbených (první a druhý screen zleva na obrázku 11) a po „swipe“ vpravo se skrývalo menu všech položek (veškeré záložky z webu katedry). Změnou byl pouze tvar oblíbených položek.

Hlavní obrazovka ve třetí verzi už byla zcela změněna. Obsahovala pouze hlavní záložky z webu a ty po kliku přenesly uživatele do podmenu s příslušnými podzáložkami. Výhodou toho řešení bylo (a je), že celá aplikace byla plně řízena daty, která přišla z webu, tzn. veškeré úpravy, které by se provedly na webu by se projevily i v aplikaci, např. nová hlavní záložka v menu. Na dalších záložkách v navigačním „bottombaru“ se nacházela nástěnka (oblíbené, aktuality a akce katedry) a nastavení aplikace.

Z tohoto konceptu sešlo z části kvůli přehlednosti aplikace a do jisté míry kvůli sjednocení vzhledu aplikace s univerzitní. Nutno dodat, že aplikace je stále plně řízena daty z webu, pouze jsou některá data upravována, aby se zobrazovala na požadovaných místech.

Poslední problém, který zbývalo vyřešit se stalo zobrazení příspěvků z Twitter účtu katedry, který je využíván pro aktuality. Přestože je profil katedry volně přístupný (bez registrace na Twitteru si lze profil zobrazit), Twitter od druhé verze svého API volný přístup nepovoluje. Vývojář, který chce API používat si musí vygenerovat speciální Tokeny, které přístup umožňují. Přestože je

aplikace OpenSource, tak nemůžu zveřejnit celý kód a jeden soubor, z důvodu obsahu citlivých dat, zůstane neveřejný.

5.2.2 Fungování aplikace

Po zapnutí aplikace se zobrazí logo katedry a začnou se stahovat data z webu. Po stažení a zpracování dat se zobrazí Přehled (hlavní obrazovka), který obsahuje tzv. boxy:

- Aktualit
- Akcí katedry
- Nejdůležitějších odkazů
- Záhlavních odkazů
- Oblíbených

Všechny zmíněné boxy, kromě boxu oblíbených, zobrazují data z příslušné části webu katedry. Všechny boxy lze libovolně po přehledu přesouvat (lze měnit jejich pořadí) nebo je lze vypnout či zapnout.

Pomocí navigačního bottombaru se uživatel může dále přesouvat na záložky Možnosti studia, Pro studenty, Členové katedry a Další. První tři zmíněné záložky obsahují příslušná podmenu (opět dle webu katedry). Každé z podmenu přesune uživatele na příslušnou stránku (dále jen Detail). Záložka Další obsahuje záložky z webu s názvem O katedře. Dále zde uživatel nalezne možnosti nastavení aplikace a další záložky spojené s aplikací - aktualizace dat a oblíbené.

Každá obrazovka obsahuje pouze upravený RecyclerView, do kterého jsou dynamicky přidávány vlastní widgety.

Detail obrazovka zobrazuje data z jednotlivých stránek katedry, které dostává z API v podobě HTML kódu. Aplikace je pomocí HTML Parseru převede na příslušné widgety podle typu (text widget, obrázkový widget, widget pro tabulku, ...), přidá je do RecyclerView a zobrazí.

5.2.3 Ikony

Aplikace zobrazuje ke každé položce v menu ilustrativní ikonu, kterou, bohužel, nedostává z dat. Veškeré zobrazované ikony jsou předem vytvořené a přiřazovány k jednotlivým položkám na základě jejich slugu[13]. Z tohoto důvodu aplikace neumí automaticky přidělit nově přidané položce do menu její vlastní ikonu. Stejný problém nastane v případě změny slugu položky. V takových případech se místo ilustrativní ikony zobrazí pouze logo katedry. Tato situace má pouze dvě možná řešení:

1. Katedra by nějakým způsobem předávala s daty i ikonu

2. Při vytvoření nové položky se bude muset do projektu s aplikací přidat nová ručně vytvořená ikona

5.2.4 Průchod aplikací

Komentovaný průchod celou aplikací si můžete prohlédnout na následujícím videu. Přímý odkaz na video naleznete taktéž v příloze A.

Obrázek 12: QR kód pro zobrazení videa s komentovanou prohlídkou

[13] Část URL adresy identifikující konkrétní stránku

5.2.5 Testování a publikace

Aplikaci jsem v průběhu vývoje testoval především na Android Emulátoru, jenž je součástí Android Studia, ve kterém jsem vyvíjel. Poté, co jsem aplikaci dodělal do takové fáze, kdy jsem byl přesvědčen, že by měla spatřit světlo světa, tak jsem ji nahrál na Google Play pouze do interního testování. Jelikož jsem dlouholetým uživatelem jablečných mobilních telefonů, neměl jsem možnost aplikaci testovat na svém telefonu.

Interní testování spočívá v tom, že nastavíte pomocí Google účtů okruh „testerů“, kteří mají přístup k aplikaci. Do tohoto týmu jsem přidal několik vybraných spolužáků i pana doktora Pexu, kteří aplikaci vyzkoušeli.

Samotné nahrání aplikace na Google Play je velice snadné. Uživatel, který chce svou aplikaci zveřejnit na Google Play se nejprve musí zaregistrovat jako vývojář do Google Play Console a zaplatit vstupní poplatek (cca 500,- Kč). Poté už uživatel může, po vyplnění nezbytných dotazníků o aplikaci, nahrát své dílo na Google Play ať už veřejně nebo do výše zmíněného interního testování. Poté už vývojář čeká, zda-li Google jeho aplikaci schválí, nebo nikoliv.

5.2.6 Dostupnost aplikace

Aplikace Katedra informatik PF JU je dostupná ke stažení na českém Google Play. Přímý odkaz na Google Play naleznete taktéž v příloze A.

Obrázek 13: QR kód pro stažení aplikace z Google Play

6 Závěr

Bakalářská práce se zabývá vývojem aplikací v programovacím jazyce Kotlin pro mobilní telefony s operačním systémem Android. Skládá z teoretické a praktické části.

V teoretické části popisuji důležité změny, které se odehrály v jednotlivých verzích OS Android od verze 5.0, což je mimochodem i nejnižší kompatibilní verze pro aplikace vzniknuvší v rámci bakalářské práce. Dále čtenáři představuji některé důležité části a widgety pro vývoj Android aplikací. V posledních kapitolách teoretické části jsem představil programovací jazyk Kotlin a některé zajímavé, velice užitečné, knihovny pro vývoj aplikací.

Troufám si tvrdit, že Kotlin by mohl nahradit Javu ve výuce programování na školách. Vše, co je možné naprogramovat v Javě, lze též vytvořit (i převést) v Kotlinu. Navíc Kotlin nabízí mnoho výhod oproti Javě. Kotlin lze zkrátka popsat jako odlehčenou moderní verzi Javy.

V praktické části jsem představil celkem tři aplikace. První dvě jsou cvičné, ve kterých ukazuji základy pro vývoj aplikací. Ze zdrojových kódů těchto cvičných aplikací vychází z větší části i poslední představená aplikace Katedra informatiky PF JU. Téměř všechny zdrojové kódy jsou k dispozici na přiloženém DVD.

Seznam použité literatury a zdrojů

[1] CERVANTES, Edgar. What is Android? Everything you need to know about Google's OS. *Android Authority* [online]. 2020 [cit. 24. 11. 2020]. Dostupné z: https://www.androidauthority.com/what-is-android-328076/

[2] KODYTEK, Samuel. Lekce 1 - Úvod do Android programování v Kotlin a vývojového prostředí. *ITnetwork* [online]. [cit. 24. 11. 2020]. Dostupné z: https://www.itnetwork.cz/kotlin/android/uvod-do-android-programovani-v-kotlin-a-vyvojoveho-prostredi

[3] CALLAHAM, John. The history of Android: The evolution of the biggest mobile OS in the world. *Android Authority* [online]. [cit. 24. 11. 2020]. Dostupné z: https://www.androidauthority.com/history-android-os-name-789433/

[4] ŠLIK, Jáchym. Potvrzeno: Huawei ztratí přístup ke službám Googlu včetně Obchodu Play. *Svět mobilně*[online]. 2019 [cit. 30. 11. 2020]. Dostupné z: https://www.svetmobilne.cz/huawei-nejspis-prijde-o-pristup-ke-sluzbam-googlu-vcetne-obchodu-play/7634

[5] ŠLIK, Jáchym. Google varuje, že systém od Huawei nebude tak bezpečný jako běžný Android *Svět mobilně* [online]. 2019 [cit. 30. 11. 2020]. Dostupnéz:https://www.svetmobilne.cz/google-varuje-ze-system-od-huawei-nebude-tak-bezpecny-jako-bezny-android/7700

[6] LEE, Tyler. Huawei's having a terrible time in Western Europe. *Phandroid* [online]. 2020 [cit. 30. 11. 2020]. Dostupné z: https://phandroid.com/2020/11/26/huaweis-having-a-terrible-time-in-western-europe/

[7] Android Lollipop. *Android Developers* [online]. [cit. 20. 11. 2020]. Dostupné z: `https://developer.android.com/about/versions/lollipop`

[8] Android 5.0 APIs. *Android Developers* [online]. [cit. 20. 11. 2020]. Dostupné z: `https://developer.android.com/about/versions/android-5.0`

[9] Android 5.1 APIs. *Android Developers* [online]. [cit. 20. 11. 2020]. Dostupné z: `https://developer.android.com/about/versions/android-5.1`

[10] Android 6.0 Changes. *Android Developers* [online]. [cit. 22. 11. 2020]. Dostupné z: `https://developer.android.com/about/versions/marshmallow/android-6.0-changes`

[11] Android 6.0 APIs. *Android Developers* [online]. [cit. 22. 11. 2020]. Dostupné z: `https://developer.android.com/about/versions/marshmallow/android-6.0`

[12] Android 7.0 for Developers. *Android Developers* [online]. [cit. 22. 11. 2020]. Dostupné z: `https://developer.android.com/about/versions/nougat/android-7.0`

[13] Android 7.1 for Developers. *Android Developers* [online]. [cit. 22. 11. 2020]. Dostupné z: `https://developer.android.com/about/versions/nougat/android-7.1`

[14] Android 8.0 Features and APIs. *Android Developers* [online]. [cit. 22. 11. 2020]. Dostupné z: `https://developer.android.com/about/versions/oreo/android-8.0`

[15] Android 8.1 Features and APIs. *Android Developers* [online]. [cit. 24. 11. 2020]. Dostupné z: `https://developer.android.com/about/versions/oreo/android-8.1`

[16] Android 9 features and APIs. *Android Developers* [online]. [cit. 24. 11. 2020]. Dostupné z: https://developer.android.com/about/versions/pie/android-9.0

[17] Eternal s.r.o. - Vývoj mobilních aplikací a webových portálů. *Eternal s.r.o* [online]. [cit. 24. 11. 2020]. Dostupné z: http://www.eternal.cz/

[18] Android 10 for Developers. *Android Developers* [online]. [cit. 24. 11. 2020]. Dostupné z: https://developer.android.com/about/versions/10/highlights

[19] Privacy in Android 11. *Android Developers* [online]. [cit. 24. 11. 2020]. Dostupné z: https://developer.android.com/about/versions/11/privacy

[20] App Manifest Overview. *Android Developers* [online]. [cit. 25. 11. 2020]. Dostupné z: https://developer.android.com/guide/topics/manifest/manifest-intro

[21] KODYTEK, Samuel. Lekce 3 - Neobjevujte kolo, použijte Gradle. *ITnetwork* [online]. [cit. 04.12.2020]. Dostupné z: https://www.itnetwork.cz/kotlin/android/neobjevujte-kolo-pouzijte-gradle

[22] Introduction to Activities. *Android Developers* [online]. [cit. 25. 11. 2020]. Dostupné z: https://developer.android.com/guide/components/activities/intro-activities

[23] Fragments. *Android Developers* [online]. [cit. 25. 11. 2020]. Dostupné z: https://developer.android.com/guide/fragments

[24] Navigation. *Android Developers* [online]. [cit. 25. 11. 2020]. Dostupné z: https://developer.android.com/guide/navigation

[25] Principles of navigation. *Android Developers* [online]. [cit. 25. 11. 2020]. Dostupné z: https://developer.android.com/guide/navigation/navigation-principles

[26] Layouts. *Android Developers* [online]. [cit. 25. 11. 2020]. Dostupné z: https://developer.android.com/guide/topics/ui/declaring-layout

[27] Create dynamic lists with RecyclerView. *Android Developers* [online]. [cit. 25. 11. 2020]. Dostupné z: https://developer.android.com/guide/topics/ui/layout/recyclerview

[28] ViewModel Overview. *Android Developers* [online]. [cit. 25. 11. 2020]. Dostupné z: https://developer.android.com/topic/libraries/architecture/viewmodel

[29] Observer. *Android Developers* [online]. [cit. 25. 11. 2020]. Dostupné z: https://developer.android.com/reference/android/arch/lifecycle/Observer

[30] „Prateek_Aggarwal". Shared Preferences in Android with Examples. *Geeks For Geeks* [online]. [cit. 25. 11. 2020] Dostupné z: https://www.geeksforgeeks.org/shared-preferences-in-android-with-examples/

[31] Save key-value data. *Android Developers* [online]. [cit. 25. 11. 2020]. Dostupné z: https://developer.android.com/training/data-storage/shared-preferences

[32] Saving UI States. *Android Developers* [online]. [cit. 25. 11. 2020]. Dostupné z: https://developer.android.com/topic/libraries/architecture/saving-states

[33] KODYTEK, Samuel. Lekce 1 - Úvod do jazyka Kotlin, platformy a IntelliJ. *ITnetwork* [online]. [cit. 01. 12. 2020]. Dostupné z:

https://www.itnetwork.cz/kotlin/zaklady/uvod-do-jazyka-kotlin-platformy-a-intellij

[34] Kotlin Programming Language. *Geeks For Geeks* [online]. [cit. 01. 12. 2020] Dostupné z: https://www.geeksforgeeks.org/kotlin-programming-language/

[35] SOUHRADA, Václav. Kotlin – jazyk, který je dobré znát. *eMan* [online]. [cit. 01. 12. 2020]. Dostupné z: https://www.eman.cz/blog/kotlin-jazyk-ktery-dobre-znat/

[36] Higher-Order Functions and Lambdas. *Kotlin Programming Language* [online]. [cit. 02. 12. 2020] Dostupné z: https://kotlinlang.org/docs/reference/lambdas.html

[37] forEachIndexed. *Kotlin Programming Language* [online]. [cit. 02. 12. 2020] Dostupné z: https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/for-each-indexed.html

[38] any. *Kotlin Programming Language* [online]. [cit. 02. 12. 2020] Dostupné z: https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/any.html

[39] Filtering Collections. *Kotlin Programming Language* [online]. [cit. 02. 12. 2020] Dostupné z: https://kotlinlang.org/docs/reference/collection-filtering.html

[40] Java vs Kotlin. *eduCBA* [online]. [cit. 02.12.2020]. Dostupné z: https://www.educba.com/java-vs-kotlin/

[41] ODHIAMBO, Paul. Kotlin and retrofit network calls. *DEV Community* [online]. [cit. 02. 12. 2020]. Dostupné z: https://dev.to/paulodhiambo/kotlin-and-retrofit-network-calls-2353

[42] Consuming APIs with Retrofit. *Codepath* [online]. [cit. 02. 12. 2020] Dostupné z: `https://guides.codepath.com/android/consuming-apis-with-retrofit`

[43] google/gson. *GitHub* [online]. 2008 [cit. 02. 12. 2020]. Dostupné z: `https://github.com/google/gson`

[44] bumptech/glide *GitHub* [online]. [cit. 02. 12. 2020]. Dostupné z: `https://github.com/bumptech/glide`

Seznam obrázků

Seznam zdrojových kódů

A Přílohy

1. DVD - na přiloženém DVD se nachází
   - Plné znění mé bakalářské práce pod názvem bp_vojacek.pdf
   - Video s komentovaným průchodem aplikace pod názvem pruchod_aplikaci.mp4
   - AndroidStudio projekt Hudební databáze pod názvem hudebni_databaze
   - AndroidStudio projekt Listovní uživateli pod názvem listovani_uzivateli
   - AndroidStudio projekt Katedra informatiky PF JU pod názvem kin_pf_ju
2. Video s komentovaným průchodem aplikace KIN PF JU - `https://youtu.be/ucdgQhnnWIc`
3. Odkaz na stažení aplikace KIN PF JU - `https://play.google.com/store/apps/details?id=cz.vojacek.kin_pf_jcu`

B Zdrojové kódy k aplikaci Hudební databáze


```
1  class MusicApp : Application() {
2      override fun onCreate() {
3          super.onCreate()
4          instance = this
5      }
6      companion object {
7          lateinit var instance: MusicApp
8      }
9  }
10 val appContext: Context get() = MusicApp.instance.applicationContext
```

Příklad 21: MusicApp

```
1 <manifest ...>
2   <application
3       android:name=".MusicApp"
4       ...>
5       ...
6   </application>
7 </manifest>
```

Příklad 22: Přidání MusicApp do Manifestu

```
1  open class NavigationController(fragmentManager: FragmentManager, containerId: Int) :
2      FragmentNavigator(containerId, fragmentManager) {
3
4      fun navigateToPrevious() {
5          fragmentManager.popBackStack()
6      }
7
8      fun emptyBackStack() = fragmentManager.backStackEntryCount <= 0
9
10     fun navigateTo(fragment: BaseFragment) {
11         show { fragment }
12     }
13
14     fun navigateToAndClear(fragment: BaseFragment) {
15         showOnlyAndClear { fragment }
16     }
17 }
18
19
```

```
20 open class FragmentNavigator(val containerId: Int, val fragmentManager: FragmentManager) {
21     inline fun <reified T : Fragment> show(createFragment: () -> T) {
22         fragmentManager.showWithArgs { createFragment() }
23     }
24
25     inline fun <reified T : Fragment> showOnlyAndClear(createFragment: () -> T) {
26         fragmentManager.showOnlyAndClearWithArgs { createFragment() }
27     }
28
29     inline fun <reified T : Fragment> FragmentManager.showWithArgs(createFragment: () -> T) {
30         takeIf { it.notDisplayed<T>() }?.commit {
31             replace(containerId, createFragment.invoke())
32             addToBackStack(null)
33         }
34     }
35
36
37
38
```

```
39  inline fun <reified T : Fragment> FragmentManager.showOnlyAndClearWithArgs(create: () -> T)
        {
40      takeIf { it.notDisplayed<T>() }?.clearStack()?.commit {
41          replace(containerId, create.invoke())
42      }
43  }
44
45  inline fun <reified T : Fragment> FragmentManager.notDisplayed(): Boolean {
46      val currentFragment: Fragment? = findFragmentById(containerId)
47      return with(currentFragment) { this?.javaClass != T::class.java }
48  }
49
50  fun FragmentManager.clearStack(): FragmentManager {
51      for (i in 0 until this.backStackEntryCount) {
52          this.popBackStack()
```

Příklad 23: Navigační kontroler

```
1  open class BaseFragment : Fragment() {
2      override fun onActivityCreated(savedInstanceState: Bundle?) {
3          super.onActivityCreated(savedInstanceState)
4          (requireActivity() as MainActivity).supportActionBar?.setDisplayHomeAsUpEnabled(!
               getNavigator().emptyBackStack())
5          (requireActivity() as MainActivity).supportActionBar?.setHomeButtonEnabled(!
               getNavigator().emptyBackStack())
6      }
7
8      protected fun getNavigator() =
9          NavigationController(
10             requireActivity().supportFragmentManager,
11             R.id.container
12         )
13 }
```

Příklad 24: BaseFragment

```
@Parcelize
data class SongModel (
    val id: Int,
    val name: String,
    val artist: String,
    val album: String,
    val year: Int
): Parcelable
```

Příklad 25: SongModel

```
@Entity(tableName = "songs", indices = [Index(value = ["id"])])
data class SongEntity (
    @PrimaryKey
    val id: Int,
    val name: String,
    val artist: String,
    val album: String,
    val year: Int
)

fun SongEntity.toModel():SongModel{
    return SongModel(this.id, this.name, this.artist, this.album, this.year)
}
fun MutableList<SongEntity>.toModel(): MutableList<SongModel>{
    val list = mutableListOf<SongModel>()
    this.forEach{
        list.add(it.toModel())
    }
    return list
```

Příklad 26: SongEntity

```
1  private const val DB_NAME = "MusicDatabase.db"
2
3  @Database(
4      entities = [
5          (SongEntity::class)
6      ],
7      version = 1
8  )
9
10 abstract class MusicDatabase : RoomDatabase() {
11     companion object {
12
13         fun getSongDao(): Dao {
14             return get().getSongDao()
15         }
16
17         private fun get(): MusicDatabase {
18             return Room.databaseBuilder(
19                 appContext, MusicDatabase::class.java,
```

```
                DB_NAME
            ).build()
        }
    }

    abstract fun getSongDao(): Dao

}
```

Příklad 27: Database

```
1  @Dao
2  interface Dao{
3      @Query("SELECT␣*␣FROM␣songs␣ORDER␣BY␣id␣DESC")
4      fun getAll(): LiveData<MutableList<SongEntity>>
5
6      @Query("SELECT␣*␣FROM␣songs␣WHERE␣id␣=:id")
7      fun getSongById(id: Int): LiveData<SongEntity?>
8
9      @Query("DELETE␣FROM␣songs␣WHERE␣id␣=:id")
10     fun deleteSongById(id: Int)
11
12     @Insert(onConflict = OnConflictStrategy.REPLACE)
13     fun save(entity: SongEntity)
14 }
```

Příklad 28: Dao

C Zdrojové kódy k aplikaci pro listování uživateli


```
1  class MainActivity : AppCompatActivity() {
2      override fun onCreate(savedInstanceState: Bundle?) {
3          super.onCreate(savedInstanceState)
4          setContentView(R.layout.activity_main)
5          try {
6              fetchData()
7          } catch (e: Exception) {
8          }
9
10         findViewById<Button>(R.id.btn_generate).setOnClickListener {
11             try {
12                 fetchData()
13             } catch (e: Exception) {
14             }
15         }
16     }
17
18
```

```
19      private fun fetchData() {
20          val url = "https://randomuser.me/api/"
21          val request = Request.Builder().url(url).build()
22          OkHttpClient().newCall(request).enqueue(object : okhttp3.Callback {
23              override fun onFailure(call: Call, e: IOException) {
24                  runOnUiThread {
25                      Toast.makeText(applicationContext, "Neco se nepovedlo. Zkontrolujte sve
                            pripojeni a zkuste to, prosim, znovu.", Toast.LENGTH_SHORT).show()
26                  }
27              }
28
29              override fun onResponse(call: Call, response: Response) {
30                  val jsonString = response.body()?.string()
31                  try {
32                      val user = Gson().fromJson(jsonString, Data::class.java)
33                      runOnUiThread {
34                          findViewById<TextView>(R.id.txt_name).text = "${user.results.first().
                                name.title} ${user.results.first().name.first} ${user.results.first
                                ().name.last}"
```

```
35                      findViewById<TextView>(R.id.txt_state).text = user.results.first().
                            location.country
36                      val src = user.results.first().picture.large
37                      Glide.with(this@MainActivity)
38                              .load(src)
39                              .into(findViewById(R.id.image_face))
40                  }
41              } catch (e: Exception) {
42                  runOnUiThread {
43                      Toast.makeText(applicationContext, "Neco se nepovedlo. Zkuste to,
                            prosim, znovu.", Toast.LENGTH_SHORT).show()
44                  }
45              }
46          }
47      })
48  }
```

Příklad 29: MainActivity aplikace pro listování uživateli