Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Softwarového inženýrství

Diplomová práce

Historie ČVUT (Android mobilní aplikace)

Bc. Tomáš Krabač

Vedoucí práce: Ing. Martin Půlpitel

10. ledna 2016

Poděkování

Mé poděkování patří především Ing. Martinu Půlpitlovi za vedení této práce, jeho cenné rady a připomínky.

Dále bych rád poděkoval za spolupráci na tomto projektu Vojtěchu Pajerovi, Ing. Vojtěchu Bartošovi, pracovníkům ze společnosti Ackee s.r.o a také dalším lidem, kteří se na projektu podíleli.

V neposlední řadě bych chtěl poděkovat Prof. PhDr. Marcele Efmertové, CSc. za konzultace a korekturu historické části a Bc. Janu Veselému za finální korekturu tohoto textu.

Nakonec bych rád poděkoval své rodině a svým přátelům za podporu během studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 10. ledna 2016 .....

Odkaz na tuto práci

Krabač, Tomáš. *Historie ČVUT (Android mobilní aplikace).* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Tato diplomová práce se zabývá návrhem a implementací mobilní aplikace pro operační systém Android o vysoké škole České vysoké učení technické v Praze. Při implementaci byla použita architektura Model–View–Presenter a reaktivní programování. Součástí práce byl sběr dat a spolupráce s kolegy, kteří v rámci svých závěrečných prací vytvářeli backend a verzi této aplikace pro operační systém iOS.

Klíčová slova ČVUT, historie, Android, MVP, Model–View–Presenter, reaktivní programování, RxJava

Abstract

This master thesis deals with design and implementation of a mobile application for Android operating system. The purpose of this application is to present information about the Czech technical university in Prague. Model-View-Presenter architecture and reactive programming were used during implementation. The work done within this thesis includes collection of necessary data and collaboration with colleagues who, in scope of their own theses, have been creating the backend part and an iOS version of this app.

Keywords CTU, history, Android, MVP, Model–View–Presenter, reactive programming, RxJava

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Mobilní aplikace patří k nejmladším v oboru informačních technologií. Tyto aplikace jsou vytvářeny pro mobilní telefony nebo tablety s operačními systémy Android, iOS aj. V současné době zažíváme velký nárůst používání mobilních zařízení a s tím souvisí rychle se zvyšující počet aplikací, proto nelze svět mobilních aplikací ignorovat. V polovině roku 2015 se odhadoval počet aplikací v obchodu Google Play na 1 300 000 [3]. Uživatelé mobilních zařízení mají odlišné nároky na mobilní aplikace, preferují jednoduché přehledné ovládání a elegantní grafiku. Mobilní zařízení umí zjistit polohu uživatele, umí spolupracovat s dalšími aplikacemi jako je navigace, kontakty a může pracovat v režimu offline.

České vysoké učení technické v Praze (dále jen ČVUT) je známé jako jedna z největších a nejstarších technických univerzit v Evropě. Je vnímána širokou veřejností jako velmi prestižní institut, který je znám ve světě a umisťuje se na předních pozicích světových žebříčků univerzit [4]. Informace o ČVUT nalezneme na webových stránkách, sociálních sítích, v příručkách a letácích, pořádají se Dny otevřených dveří, škola má své vlastní videozpravodajství [5] aj. Co ovšem dosud chybí, je mobilní aplikace, která by prezentovala ČVUT.

Tato diplomová práce se zabývá vývojem mobilní aplikace na platformě Android, která může sloužit uživatelům jako zdroj informací o ČVUT, její historii a současnosti. Aplikace bude mít kromě české verze také anglickou verzi, která je důležitá, pokud uvažujeme o užití aplikace pro cizince. Pokud bude ČVUT aplikaci udržovat, může být zdrojem důležitých informací pro zájemce o studium, pro zaměstnance a studenty ČVUT nebo pro zahraniční návštěvníky. Mohla by být také využívána při prezentaci školy na různých akcích, veletrzích nebo na zahraničních cestách zaměstnanců školy.

Práce je součástí většího projektu, jehož dalšími částmi jsou bakalářská práce, která řeší verzi aplikace na platformě iOS, a diplomová práce, která se zabývá získáváním dat z webového backendu přes API rozhraní.

Struktura práce

První kapitola krátce čtenáře seznamuje s projektem Historie ČVUT, jakou roli v něm tato diplomová práce hraje a kdo se na projektu podílel.

Ve druhé kapitole jsou popsány cíle práce a je podrobněji rozebráno její zadání. Kapitola se zabývá funkčními a nefunkčními požadavky aplikace včetně těch, které nebyly realizovány, ale mohly by sloužit jako možná rozšíření do budoucna. V závěru kapitoly je určen charakter aplikace a jsou identifikovány její cílové skupiny.

Ve třetí kapitole je představen operační systém Android a jak se na jeho platformě vytvářejí nativní mobilní aplikace.

Čtvrtá kapitola se zabývá analýzou existujících řešení a průběhem sběru dat pro danou aplikaci.

Pátá kapitola se věnuje vlastnímu návrhu aplikace. Popisuje přehled funkcí aplikace, zabývá se na základě wireframů jednotlivými obrazovkami aplikace a popisuje datový model aplikace. Kapitola je věnována také použité architektuře aplikace.

Průběh realizace aplikace přibližuje šestá kapitola. Jsou v ní ve stručnosti ukázány některé použité knihovny a popsány problematické části implementace aplikace.

V sedmé kapitole jsou rozebrány možnosti testování aplikace s operačním systémem Android. Kapitola ukazuje, jak testování probíhalo, jaké přínosy mělo a k jakým změnám došlo.

V poslední kapitole je shrnuto, jakých výsledků v práci na mobilní aplikaci bylo dosaženo a jak je možné dále aplikaci rozvíjet.

Kapitola 1

O projektu

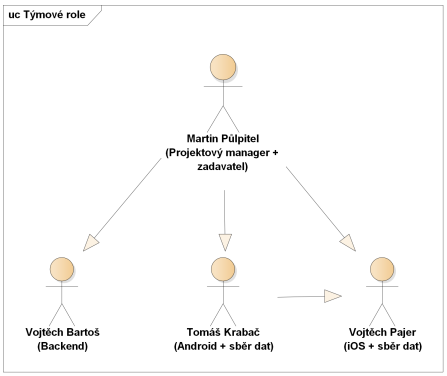
S myšlenkou vytvořit mobilní aplikaci propagující ČVUT přišel Ing. Martin Půlpitel, který vypsal na toto téma tři rámcová zadání.

V rámci prvního rámcového zadání bylo za úkol navrhnout a implementovat webovou aplikaci, která umožní vkládat a spravovat data a poskytovat je mobilním aplikacím (backend). Na této části pracoval Ing. Vojtěch Bartoš.

Druhým vypsaným rámcovým zadáním bylo navrhnout a implementovat mobilní aplikaci pro operační systém Android [6]. Součástí tohoto zadání bylo analyzovat podobné aplikace a sesbírat data pro webový backend. Toto zadání si vybral autor této práce.

Třetí rámcové zadání bylo analogické jako předchozí, ale pro platformu iOS [7]. K tomuto tématu se nikdo nepřihlásil, proto bylo stejné téma vypsané v rámci bakalářského studijního programu, na které se přihlásil Vojtěch Pajer. Kvůli této změně bylo rozšířeno zadání autora této práce o vedení bakalanta Vojtěcha Pajera.

Rozdělení týmových rolí v projektu[1] ukazuje diagram 1.1.



Obrázek 1.1: Týmové role

[1]Znázorněny jsou pouze role v rámci závěrečných prací. Projektu se účastnilo více lidí.

První závěrečná práce, zabývající se backendem, byla dokončena v létě 2015, Android aplikace je součástí této práce a iOS aplikace se plánuje dokončit v létě 2016.

Na tomto projektu se podílí také společnost Ackee s.r.o [8] (dále jen Ackee), která poskytuje technické zázemí, vytváří grafiku na základě wireframů[2], probíhají zde všechny projektové schůzky a poskytuje své pracovníky ke konzultaci. Ing. Roman Gordienko vytvořil hlavní část grafiky a poskytl konzultace UX designu[3]. Inspirace, jak správně psát Android aplikace, tipy na některé použité knihovny a konzultace probíhaly především s Ing. Davidem Bilíkem a Bc. Michalem Kučerou.

Mezi další spolupracovníky projektu patřila Prof. PhDr. Marcela Efmertová, CSc., která poskytovala konzultace a provedla korekturu historické části. S překladem historických dat pomáhal Bc. Tomáš Hána a aplikace byla prověřena 14 testery.

[2]Wireframe je návrh znázorňující rozložení prvků, jejich chování a ukazuje dostupné funkce aplikace [9].

[3]User Experience (UX) je zaměření na uživatele, jeho potřeby a schopnosti. UX design se zabývá návrhem aplikace na základě těchto znalostí [10].

Kapitola 2

Cíle práce, specifikace zadání

Cílem práce je navrhnout a implementovat mobilní aplikaci pro operační systém Android, která bude fungovat na mobilních telefonech a tabletech. Aplikace s názvem `Historie ČVUT` bude prezentovat důležité momenty v historii ČVUT, její významné vědecké osobnosti, vznik fakult a chronologii rektorů.

Po analýze požadavků zadání práce a rozmyšlení, co dále by aplikace mohla obsahovat, vznikl soupis jejích funkčních[4] a nefunkčních[5] požadavků. Během sběru dat byly některé požadavky dále upřesňovány. Tučně zvýrazněné požadavky v oddílech 2.1 a 2.2 jsou nutné pro splnění zadání, ostatní jsou nad jeho rámec.

V závěru kapitoly jsou popsány nápady, k jejichž realizaci nedošlo, ale mohou sloužit jako možná rozšíření do budoucna.

2.1 Funkční požadavky

1. **Zobrazení milníků ČVUT**
2. **Zobrazení rektorů ČVUT**
3. **Zobrazení významných vědeckých osobností ČVUT**
4. Zobrazení významných sportovců ČVUT
5. Zobrazení základních informací o ČVUT
6. Zobrazení základních informací o fakultách ČVUT

2.2 Nefunkční požadavky

1. **Aplikace bude stahovat data z webového API**

[4] Funkční požadavky specifikují chování nebo funkce aplikace [11].
[5] Nefunkční požadavky upřesňují nebo omezují, jak se aplikace má chovat [11].

2. **Aplikace bude v češtině a v angličtině**
3. **Aplikace bude fungovat na telefonu a tabletu**
4. **Aplikace bude fungovat pro OS Android**
5. Aplikace bude fungovat po stažení dat offline

2.3 Upřesnění požadavků

Nyní budou jednotlivé funkční a nefunkční požadavky podrobněji popsány.

2.3.1 Zobrazení milníků ČVUT

Aplikace bude zobrazovat důležité historické události ČVUT, jako je např. vznik ČVUT, vznik jednotlivých fakult aj. Cílem je zaujmout uživatele, proto bude každá obrazovka s milníkem obsahovat obrázek a krátký popis.

2.3.2 Zobrazení rektorů ČVUT

Aplikace bude zobrazovat přehled všech rektorů školy. Na rozdíl od milníků je počet rektorů známý. Obrazovka s rektorem bude obsahovat obrázek, základní popis jeho osoby, v jakém období byl nebo je rektorem, datum narození a případně datum úmrtí.

2.3.3 Zobrazení významných vědeckých osobností ČVUT

Aplikace bude zobrazovat seznam významných vědeckých osobností ČVUT. Každá obrazovka s osobou bude obsahovat obrázek, krátký popis v souvislosti s ČVUT, datum narození a případně datum úmrtí.

2.3.4 Zobrazení významných sportovců ČVUT

Během sběru dat se ukázalo, že ČVUT má také mnoho významných sportovců. Přestože jejich sportovní úspěchy přidávají škole na prestiži, nejsou veřejnosti moc známí. Proto bylo rozhodnuto je v aplikaci uvést také. Údaje o nich budou stejné jako u významných vědeckých osobností, viz 2.3.3.

2.3.5 Zobrazení základních informací o ČVUT

Nedá se předpokládat, že by uživatele zajímala jen historie a osobnosti, ale bylo by užitečné prezentovat mu také základní přehled o současném ČVUT a jeho fakultách. Obrazovka k tomu určená bude obsahovat obrázek charakterizující ČVUT, krátký popis ČVUT, rok založení ČVUT, počet studentů ČVUT a současného rektora.

Telefon	Tablet
orientace na výšku	orientace na šířku
malé rozlišení displeje	velké rozlišení displeje
více praktický	určen spíše pro zábavu
držení jednou rukou	držení dvěma rukama

Tabulka 2.1: Rozdíl mezi telefonem a tabletem

2.3.6 Zobrazení základních informací o fakultách ČVUT

Aplikace bude zobrazovat obrázek charakterizující danou fakultu, krátký popis, rok založení, počet studentů a současného děkana fakulty.

2.3.7 Aplikace bude stahovat data z webového API

Návrh a implementace webového API[6] je součástí jiné diplomové práce [12]. Dokumentace k webovému API je dostupná na [13]. Tato aplikace by mohla fungovat i bez webového API, proto budou dále rozebrány klady a zápory použití webového API.

Nevýhodou webového API, oproti uložení dat přímo v aplikaci, je komplikovanější a pomalejší komunikace při jejich získávání. Dále je zde provázanost, pokud je třeba rozšířit aplikaci se zachováním výhod webového API, je potřeba udělat změny nejen v aplikaci, ale i ve webovém API.

Na druhou stranu není problém kdykoliv data poskytovaná webovým API změnit. Navíc je k dispozici webové rozhraní, proto se o data může starat i laik. Zároveň se práce na tomto projektu stala zajímavější a více připravuje na praxi, kde se webové API běžně používá.

2.3.8 Aplikace bude v češtině a v angličtině

Webové API bude poskytovat data jak v češtině tak v angličtině. O tom, v jakém jazyce se data zobrazí, bude rozhodovat nastavení jazyka v zařízení. Texty jako názvy obrazovek, chybová hlášení aj. je lepší mít v aplikaci, a tak nebudou získávána z webového API.

2.3.9 Aplikace bude fungovat na telefonu a tabletu

Je potřeba zohlednit, že telefony se v některých parametrech liší od tabletů. Podstatné rozdíly pro tento projekt ukazuje tabulka 2.1. Popsané rozdíly neplatí obecně. Pro tento typ aplikace se ukázalo jako vhodnější zařízení tablet, protože díky většímu displeji často dovoluje zobrazit více dat.

[6] Web Application Programming Interface (webové API) je rozhraní pro komunikaci s webovou aplikací

2.3.10 Aplikace bude fungovat pro OS Android

Podrobněji o operačním systému Android pojednává kapitola 3. Zde je důležité určit, od jaké verze Androidu bude aplikace podporována, což je řešeno v pododdíle 3.5.

2.3.11 Aplikace bude fungovat po stažení dat offline

Je zde možnost, že se bude aplikace používat na různých propagačních akcích ČVUT, jako např. Den otevřených dveří nebo na zahraničních akcích, kde nemusí být k dispozici internet. Proto by bylo vhodné umožnit stažení všech dat, nabízených webovým API, do lokálního úložiště.

2.4 Nerealizované funkcionality

Ne všechny funkce, které vznikly nad rámec zadání závěrečných prací, se povedlo realizovat. Některé z funkcí byly zamítnuty v brzké fázi projektu kvůli nevhodnosti nebo byly upřednostněny z časových důvodů jiné funkce. Je zde však možnost pro budoucí rozšíření, proto o nich bude zmíněno v tomto oddíle.

2.4.1 Novinky na ČVUT

Jedním z prvních nápadů, který byl po konzultaci zamítnut Romanem Gordienkem[7], bylo zobrazování událostí a novinek z ČVUT. Realizace by mohla vypadat tak, že by se data agregovala z různých zdrojů jako je Facebook [14], stránky ČVUT apod., případně by někdo novinky přidával ručně.

Argumentem proti realizaci byl fakt, že informačních kanálů má ČVUT dostatek a tato funkcionalita by byla zbytečná.

2.4.2 Historie fakult

Původní myšlenka byla, že podobně jako aplikace obsahuje milníky ČVUT (viz pododdíl 2.3.1), mohla by obsahovat i milníky jednotlivých fakult. Zde realizace narazila na nedostatek dat. Přesněji řečeno se nepodařilo nalézt zdroje, které by pro každou fakultu mapovaly její historii včetně obrázků. Použití obrázků je důležité, protože by uživatele samotný text nemusel zaujmout.

2.4.3 Více informací o fakultách

Prozatím jsou v přehledu o fakultě zobrazeny nejdůležitější údaje (viz 2.3.6). Další informace, které by k fakultám mohly být doplněny, jsou obrázky budov včetně popisku a GPS lokací, insignie [15] a informace o děkanech. Další nápady k informacím o fakultách, kteří přinesli testující, budou zmíněny v kapitole o testování 7.

[7] Grafik a UX designér v Ackee s.r.o.

2.4.4 Informace o ostatních pracovištích ČVUT

Kromě fakult existuje na ČVUT i mnoho jiných významných pracovišť, ústavů aj. [16], o kterých by bylo dobré udělat přehled podobně jako u fakult zmíněných v pododdíle 2.3.6.

2.4.5 Úspěšní studenti ČVUT

Stejně jako jsou v aplikaci uvedeny významné vědecké osobnosti, mohli by být přidáni i úspěšní studenti ze současné doby.

2.4.6 Zajímavosti o ČVUT

Během sběru dat se ukázalo, že existuje mnoho zajímavostí, které by mohly uživatele zaujmout. Například, že dříve bylo součástí ČVUT současné VŠCHT[8] a ČZU[9] v jiné formě nebo že ženy se mohly oficiálně účastnit výuky až po roce 1902.

2.4.7 Jazykové verze

Aplikace dle zadání musí umět češtinu a angličtinu. Bylo by dobré přidat i další jazyky.

Další rozšíření je možnost změnit jazyk přímo v aplikaci. Současné řešení se řídí nastavením jazyka telefonu.

2.4.8 Virtuální prohlídka

Poslední ze zajímavých nápadů bylo vytvoření virtuální prohlídky. Bohužel tento nápad se ukázal jako časově velmi náročný jak na implementaci, tak na vytvoření a editaci dat. Jednou z komplikací při realizaci virtuální prohlídky byla nutnost vytvoření specifických fotografií areálu školy. Na to je třeba dobré počasí a pro nezkušenost i mnoho experimentování. Dalším faktem bylo, že část areálu Dejvic se během realizace tohoto projektu nacházela v rekonstrukci a virtuální prohlídka by neměla takový efekt, protože část areálu byla rozkopaná.

Tato myšlenka vycházela z aplikace, která bude později popsána v pododdíle 4.1.4. Na tento nápad byly vytvořeny návrhy obrazovek aplikace, které jsou součástí CD této práce.

2.5 Charakter aplikace a cílové skupiny

Při tvorbě mobilní aplikace je vhodné vymezit, k čemu bude sloužit, kdo jsou její cíloví uživatelé a co od aplikace mohou očekávat.

[8] Vysoká škola chemicko-technologická v Praze
[9] Česká zemědělská univerzita v Praze

2.5.1 Charakter aplikace

Účelem této aplikace je v přehledné a jednoduché formě podat uživateli informace o ČVUT. Aplikace je primárně určena pro prezentaci a propagaci ČVUT, nepředpokládá se její každodenní používání.

2.5.2 Cílové skupiny

Byly určeny dvě hlavní cílové skupiny. První cílovou skupinou jsou lidé, kteří již mají k ČVUT nějaký vztah. Mezi ně patří pedagogové, studenti, absolventi a jiní, kteří mají o ČVUT nějaké povědomí. Tito uživatelé by v aplikaci měli najít informace, které nevěděli nebo které by je mohly překvapit. Patří mezi ně historie, kterou ne všichni dobře znají. Některé studenty může překvapit, že jejich pedagog byl dříve rektor aj.

Druhou cílovou skupinou jsou lidé, kteří se chtějí o ČVUT něco dozvědět. Mezi ně patří zájemci o studium, zahraniční studenti a jiní. Pro ně by měla aplikace obsahovat jasné a stručné informace, měla by zaujmout designem, ale i obsahem. U této skupiny je možné předpokládat, že si aplikaci sami nestáhnou, ale že by např. na Dni otevřených dveří mohlo být k dispozici zařízení, které by si zájemci mohli půjčit. Ideálně by se mohlo jednat o tablet. Tato možnost je důležitá při návrhu aplikace, neboť se nelze spoléhat např. na to, že si uživatelé aplikaci sami nainstalují a zobrazí se jim průvodce, jak ji ovládat.

Kapitola 3

Android

Android je operační systém založený na Linux jádře [17]. Využívaný je především pro mobilní zařízení, i když jsou zde snahy o použití u inteligentních televizí (Google TV), ovládání aut aj. [18]

V této teoretické části bude vysvětleno, co je operační systém Android a jak probíhá vývoj mobilních aplikací pro tento systém. Tato kapitola nemá za cíl naučit čtenáře programovat, ale poskytnout rámcový přehled pro pochopení dalších částí této práce.

Nativní aplikace (viz 3.1) jsou psané v jazyce Java [19]. Na rozdíl od Javy tyto aplikace neběží na Java Virtual Machine [20], ale je použit virtuální stroj Dalvik, který je postupně nahrazován jeho novějším následovníkem Android Runtime (ART) [21]. Pro základní vývoj je použit `Android Framework`[10], který obsahuje základní komponenty. Dále lze při vývoji použít Google API, knihovny třetích stran, NDK[11] aj.

3.1 Nativní, hybridní a webová aplikace

Existuje více způsobů, jak vytvořit mobilní aplikaci pro Android. Mezi tyto způsoby patří `nativní aplikace`, `hybridní aplikace` a `webová aplikace`. Všechny tyto tři způsoby budou v tomto oddíle popsány.

`Nativní aplikace` znamená, že aplikace bude vyvíjena pro specifickou platformu nebo zařízení, v tomto případě pro Android zařízení. To má za následek užší spolupráci s hardwarem a operačním systémem než jiné varianty. Na druhou stranu se většinou jedná o časově nejnáročnější způsob. Tento způsob je nejrozšířenější a pro koncového uživatele tyto aplikace bývají uživatelsky nejpřívětivější.

`Hybridní aplikace` je opak nativní aplikace, tedy aplikace, která je multiplatformní. Toho lze například dosáhnout tak, že je použit framework, který

[10] Framework je softwarová struktura, která obsahuje podpůrné programy, knihovny, API, základní komponenty aj., které podporují vývoj.

[11] Native Development Kit slouží pro vývoj v nativním kódu jako je C nebo C++.

z jednoho kódu vygeneruje nativní aplikace pro konkrétní platformu. Tyto frameworky mívají řadu omezení, bývají placené, nezohledňují specifika dané platformy apod. Na druhou stranu zde může být významně ušetřen čas a lidské zdroje, protože nejsou potřební vývojáři pro všechny platformy.

Speciální případ `hybridní aplikace` je `webová aplikace`, která se přizpůsobí pro dané zařízení. To znamená, že je také multiplatformní a ke spuštění stačí pouze webový prohlížeč. Tyto aplikace mají mnohem větší omezení než předchozí dvě, protože se nemusí dostat ke všem zdrojům, které zařízení poskytuje, jako jsou např. čidla, uložené obrázky, poloha aj.

Pro vývoj aplikace tohoto projektu byl vybrán první způsob, tedy `nativní aplikace`, i když je časově náročnější, ale má nejvíce možností pro vývojáře a lze ji udělat lépe uživatelsky přívětivou.

3.2 Základní stavební prvky

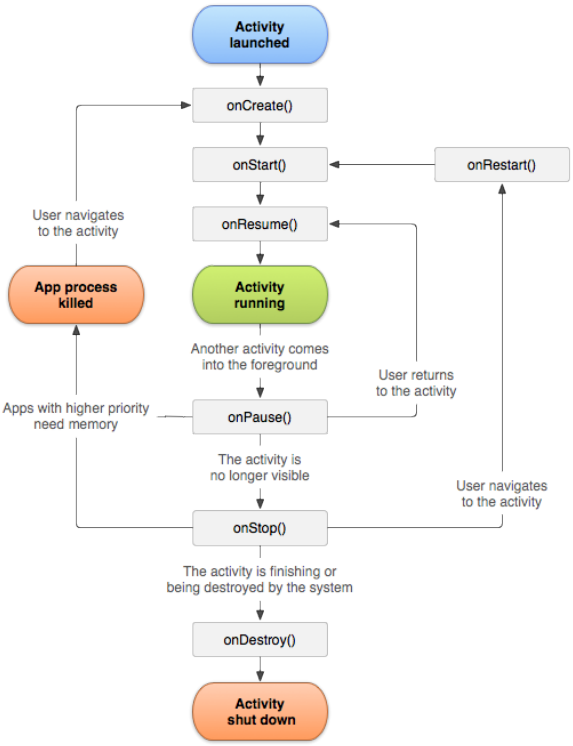
Mezi základní stavební prvky `Android Frameworku` patří `Activity`, `Service`, `Content provider` a `Broadcast reciever` [22]. Tyto prvky budou uvedeny dále.

3.2.1 Activity

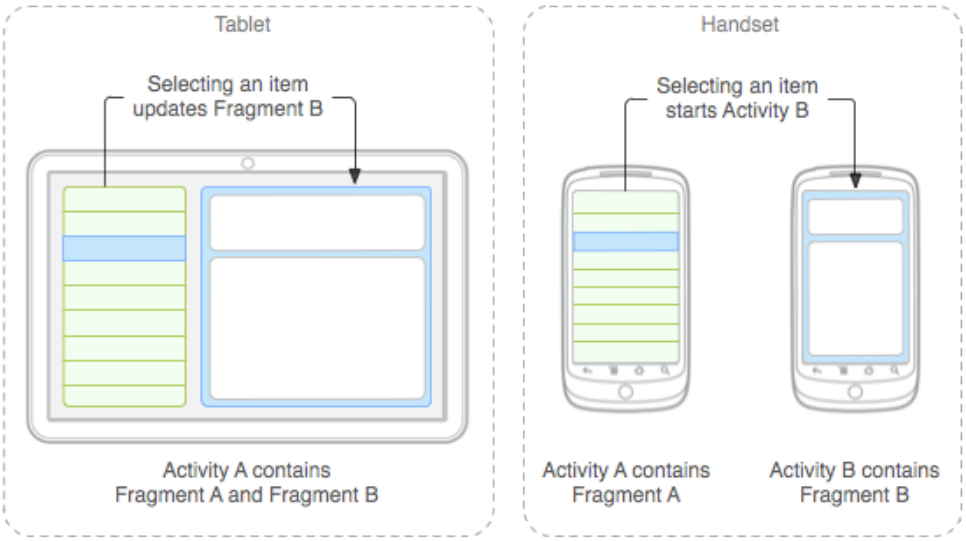
`Activity` (dále jen `Aktivita`) je třída starající se o vytvoření okna, na které se umísťují UI (User Interface) prvky. S těmito prvky uživatel interaguje. Každá `Aktivita` má svůj životní cyklus (viz diagram 3.1). `Aktivita` se může nacházet ve čtyřech stavech:

- **foreground activity** – `Aktivita` je na popředí, pro uživatele viditelná
- **visible activity** – `Aktivita` je pro uživatele viditelná, ale bývá překryta jinou `Aktivitou`, např. dialogem
- **background activity** – `Aktivita` pro uživatele není vidět, ale je stále uložena v paměti
- **empty process** – `Aktivita` byla odstraněna

V souvislosti s `Aktivitou` je dobré zmínit `Fragment` [23], který v případě jeho použití je s `Aktivitou` pevně svázán, i když má svůj vlastní životní cyklus. `Fragment` primárně slouží pro zapouzdření uživatelského rozhraní. Jedna `Aktivita` může obsahovat více `Fragmentů`. Toho se pak využívá při vytváření verze pro tablet, nebo při znovupoužití stejné části obrazovky. Pro pochopení bude uveden jako příklad telefonní seznam. Jeden `Fragment` bude zobrazovat seznam kontaktů a druhý `Fragment` bude zobrazovat detail kontaktu. V případě tabletu budou současně viditelné oba `Fragmenty` a v případě telefonu uživatel nejdřív uvidí seznam kontaktů a po kliknutí na konkrétní detail se nahradí `Fragment` za detail kontaktu. Celou situaci popisuje obrázek 3.2.



Obrázek 3.1: Životní cyklus Aktivity (zdroj: [1])



Obrázek 3.2: Fragmenty – tablet vs telefon (zdroj: [2])

3.2.2 Service

`Service` [24] je služba, která běží na pozadí a často nepotřebuje interagovat s uživatelským rozhraním. Hodí se pro dlouhotrvající procesy jako je např. stahování dat. Služba je spouštěna, zastavována a kontrolována z jiných komponent aplikace, např. z `Aktivity`, ale může běžet nezávisle na nich.

3.2.3 Content provider

`Content provider` [25] neboli poskytovatel obsahu je rozhraní, které spravuje přístup ke strukturovaným datům jiné aplikace. Mezi tato data patří např. kontakty, kalendář, aj.

3.2.4 Broadcast reciever

`Broadcast reciever` [26] je komponenta, která přijímá systémem řízené zprávy. Mezi tyto zprávy patří např. příchozí hovor, pořízení fotografie, upozornění na slabou baterii apod. Na tyto zprávy je pak možné reagovat, některé lze poslat dál nebo je pozastavit.

3.3 UI komponenty

`UI komponenty` (dále jen `komponenty`) jsou základní ovládací prvky, které uživatel vidí. `Android Framework` obsahuje mnoho takových prvků, proto budou popsány pouze základní a hlavně ty, které jsou v projektu použity.

`UI komponenty` je možné vytvářet programově nebo definovat v XML[12]. Vytváření v XML je preferovaný způsob, protože je přehlednější a vývojové prostředí umí z tohoto formátu vytvořit náhled obrazovky. Samozřejmě existují případy, kdy se programovému vytváření designu nelze vyhnout.

3.3.1 View

`View` je základní `komponenta`, od které se dědí všechny ostatní `komponenty`. Obsahuje základní metody, mezi které patří nastavení rozměrů, nastavení pozadí, reakce po kliknutí na `View` aj.

3.3.2 Textview

`Textview` slouží většinou jako textový popisek, tzn. nejde uživatelem přímo editovat a nastavování probíhá programově.

[12] XML (Extensible Markup Language) je softwarově a hardwarově nezávislý nástroj pro ukládání a přenos dat [27].

3.3.3 EditText

`EditText` je podobný `TextView` s tím, že text lze editovat uživatelem.

3.3.4 Button

`Button` neboli tlačítko je `komponenta`, která je přizpůsobena tak, aby na ni uživatel mohl snadno kliknout a poté se vykonala určitá akce.

3.3.5 ImageView

`ImageView` je `komponenta`, která slouží pro zobrazování obrázků.

3.3.6 RecycleView

`RecycleView` je `komponenta`, která obsahuje další `komponenty`, většinou stejného typu, tzv. `item`. Příklad může být seznam kontaktů, který bude obsahovat vždy jméno (`TextView`), příjmení (`TextView`) a obrázek uživatele (`ImageView`), to vše tvoří jeden `item`. Používá se v případě, kdy není známo, kolik bude položek, nebo se všechny nevejdou na obrazovku.

3.4 Layout

`Layout` je rozvržení obrazovky, které může obsahovat více `komponent`.

3.4.1 ViewGroup

Podobně jako předchozí `komponenty` měly základní prvek `View`, zde je základní prvek `ViewGroup`, který může obsahovat další `View`.

3.4.2 FrameLayout

Základní používaný `Layout` je `FrameLayout`. Organizuje `komponenty` tak, že je dává nad sebe po ose z, to znamená, že se `komponenty` překrývají. Kam budou `komponenty` umístěny lze změnit pozicováním např. na střed nebo na konkrétní stranu.

3.4.3 LinearLayout

`LinearLayout` funguje tak, že umisťuje `komponenty` vedle sebe buď podle horizontální osy nebo podle vertikální osy. Výhodou tohoto `Layoutu` je jeho jednoduchost a jeho postupným zanořováním[13] se dá sestavit téměř jakýkoliv design. Co se týče výkonu, není optimální příliš velké zanoření.

[13] Jeden `Layout` může obsahovat i jiné `Layouty`.

3.4.4 RelativeLayout

`RelativeLayout` umisťuje `komponenty` navzájem vůči sobě. To má výhodu, že nedochází tolik k zanořování, které má negativní důsledky na výkon, ale kód může být hůře čitelný. Nevýhodou je, že každá `komponenta`, vůči které se odkazuje, musí být pojmenovaná. To u předchozích `Layoutů` nebylo nutné.

3.5 Verzování Androidu

Jak se `Android Framework` a s ním operační systém vyvíjejí, postupně vznikají jejich různé verze, které bylo třeba rozlišit. Některé změny ovlivňují API systému, přístupné pro vývojáře, a jiné chování systému nebo jeho vizuální vzhled, které zajímá hlavně uživatele.

Pro různé verze API systému se zavedlo tzv. `level API`. Tyto verze jsou číselné, rostou inkrementálně, aby bylo lépe vidět, jak šly za sebou. To znamená, že se začalo od `level API` 1, pokračovalo se `level API` 2 a současná verze je 23. Toto verzování je důležité pro vývojáře, protože ovlivňuje API poskytované systémem.

Verze systému (`platform version`) naproti tomu mají vícestupňové verzování, tzn. používá se označení Android 5.1, 6.0 apod. Toto verzování je důležité spíš pro uživatele, protože ovlivňuje vzhled a chování systému.

Spolu s tím existuje i `kódové označení` verze systému, pro lepší zapamatování. Např. Android 5.0 nese `kódové označení` Lolipop. Zajímavostí je, že `kódová označení` nesou jména po nějaké sladkosti a že jsou řazena abecedně, např. Gingerbread, Honeycomb, Ice Scream Sandwitch, Jelly Bean, KitKat, Lollipop atd.

Jak spolu tyto verze souvisí, ukazuje tabulka na stránce pro vývojáře [28].

Na začátku vývoje je důležité určit, od jaké minimální verze `level API` bude aplikace podporována. Nízká verze sice zajišťuje větší dostupnost pro uživatele, ale přináší řadu omezení a problémů pro programátora, kvůli zpětné kompatibilitě.

Na základě použitých knihoven a doporučení od Ackee vývojářů bylo pro tento projekt použito `level API` 14, které je podle Google statistik [29] na 96 % aktivních zařízeních.

3.5.1 Support library

Pro lepší zpětnou kompatibilitu mezi různými verzemi Androidu existuje sada knihoven `Support library` [30]. Díky ní je možné použít některé komponenty, které se objevily v nových verzích `level API` i na starších verzích. Knihovna je pak součástí aplikace.

Komponenty, které jsou součástí `Support library`, jsou např. `ViewPager`[14] nebo `Navigation drawer`[15].

3.5.2 Google Play Services

`Google Play Services` [31] je způsob, jak může Google nové API doručovat i na telefony se starší verzí OS bez nutnosti jeho aktualizace systému, protože se jedná o aplikaci, kterou má každý telefon nainstalovaný a která se sama aktualizuje. Výhodou je, že na rozdíl od `Support library` není součástí aplikace, proto ji může používat více aplikací a šetří se tím místo. Bohužel s tím souvisí fakt, že vývojář nemůže určit, jaká verze `Google Play Services` bude na cílových zařízeních a je nutné tuto skutečnost ošetřit.

Mezi služby, které tato knihovna zprostředkovává, patří Google mapy, Google účty aj.

3.6 Android Studio

Výběr vývojového prostředí nebyl příliš složitý. `Android Studio` [32] je oficiálním vývojovým nástrojem pro Android a obsahuje vše podstatné pro tvorbu Android aplikací. Mezi další alternativy patří `Eclipse Studio`, které bylo oficiálním vývojovým nástrojem pro Android dříve. Samozřejmě jako u jiných frameworků není problém používat jiné vývojové prostředí nebo libovolný textový editor, neboť je samostatně k dispozici `Android SDK`[16]. Tím se ale přichází o komfort a výhody, vyplývající z prostředí, které je pro `Android Framework` připraveno.

3.6.1 Grandle

`Grandle` je build systém, který umí kompilovat, testovat, deployovat a publikovat aplikace. `Android Studio` obsahuje pro tyto činnosti základní úlohy (tasky). Stávající úlohy lze rozšířit nebo přidat vlastní.

Syntaxe vychází z dynamického skriptovacího jazyka *Groovy*. Kombinuje sílu a flexibilitu *Antu* s řešením závislostí a konvencemi *Mavenu*.

Obsahuje základní úlohy pro kompilaci, spuštění testů, clean[17] aj. [33].

3.6.2 Verze Javy

Podle dokumentace [34] je pro `Android Studio` potřeba minimálně Java verze 1.6. Pro Android 5.0 je již třeba verze 1.7. Nejnovější verze Javy 1.8 není

[14] `ViewPager` slouží pro plynulý přechod mezi obrazovkami, který uživatelé znají např. z galerie pro přechod mezi obrázky.

[15] `Navigation drawer` je postranní navigační lišta.

[16] `Android SDK` (Software Development Kit) je soubor nástrojů pro vývoj na Android platformě [32].

[17] Úloha *clean* smaže build a jeho dočasné soubory.

`Android Studiem` oficiálně podporována, přesto existují pluginy pro `Grandle`, které toto umožňují. V tomto projektu byl použit *Gradle Retrolambda Plugin* [35].

Důvod, proč použít Javu 1.8, bylo, že přináší mimo jiné lambda výrazy, díky kterým je zápis mnohem kratší[18]. Rozdíl kódu s použitím lambda výrazu a klasickým způsobem na přiřazení akce po stisknutí tlačítka je v ukázce 3.1, kde je vidět, jak se nahradí vytváření anonymní třídy.

Zdrojový kód 3.1: Ukázka lambda výrazu

```
1
2  //klasicky zpusob
3  mButton.setOnClickListener( new View.OnClickListener() {
4      @Override
5      public void onClick(View view){
6          // Some Action
7      }
8    });
9
10 //s lambda vyrazem
11 mButton.setOnClickListener(view -> {/*Some action*/});
```

3.6.3 Build varianty

Při vývoji se lze setkat s různými `build variantami` aplikace [36]. Ty se mohou lišit svým průběhem sestavení, nastavením obfuskace[19], nastavením podepisovacího klíče, konfigurací (např. jiná adresa API, jiné konstanty) aj.

Existují dvě základní build varianty:

1. **debug** – určena pro vývoj
2. **release** – určena pro koncové uživatele

`Debug build varianta` je určena pro vývoj, kdy se potřeba aplikaci rychle sestavit, proto mívá vypnutý `ProGuard` (viz později 3.6.4), který prodlužuje dobu sestavení. Dále se často podepisuje debug klíčem, který může být univerzální pro více aplikací, může mít nastavené testovací API a mnoho dalšího.

Oproti tomu `Release build varianta` je napojena na produkční API, podepisuje se jedinečným klíčem, spustí se `ProGuard`, který aplikaci zmenší a obfuskuje, mohou se spouštět různé druhy testů apod.

[18] Lambda výrazy samozřejmě přináší mnoho jiných výhod vyplývajících z Lamda kalkulu, jako je snadná implementace paralelismu, žádné vedlejší efekty apod. Tyto výhody se v projektu neuplatnily.

[19] Obfuskace odstraňuje nepoužitý kód, přejmenovává třídy, metody, proměnné aj., čímž je kód menší a pro člověka hůře čitelný [37].

Lze přidat libovolné množství dalších `build variant`, např. Beta, která bude sloužit pro beta testování. Ideálně by se měla podobat `Release build verzi`, tzn. mít zapnutý `ProGuard`, spouštět testy. Na rozdíl od `Release build varianty` může být stále podepsaná debug klíčem a může být stále napojena na testovací API.

Dále se lze setkat s tzv. `flavors`, které přidávají další dimenzi k `build variantám`. Např. mohou existovat dvě dimenze – verze zdarma, placená verze. Obě dvě `flavors` budou mít zároveň `Debug` i `Release build variantu` a v `Android Studiu` lze zvolit, se kterou se aktuálně bude pracovat.

Pokud by existovaly dvě `build varianty` (Debug, Release) a dvě `flavors` (free, paid), budou vytvořeny následující čtyři tasky, kterými lze vytvořit čtyři různé verze aplikace:

- freeDebug
- freeRelease
- paidDebug
- paidRelease

3.6.4 ProGuard

`ProGuard` [37] je nástroj, který slouží k obfuskaci, optimalizaci a minimalizaci kódu. Výsledkem může být menší a hůře dekompilovatelná[20] aplikace.

Jeho použití zároveň přináší riziko, že se aplikace nebude chovat dle očekávání kvůli obfuskaci. To může být například způsobeno tím, že některá z knihoven vyžaduje přesně pojmenovanou metodu a obfuskace tuto metodu přejmenuje. Existuje konfigurační soubor s pravidly, jak se má `ProGuard` zachovat, a tvůrci knihoven do dokumentace uvádějí, která pravidla nastavit.

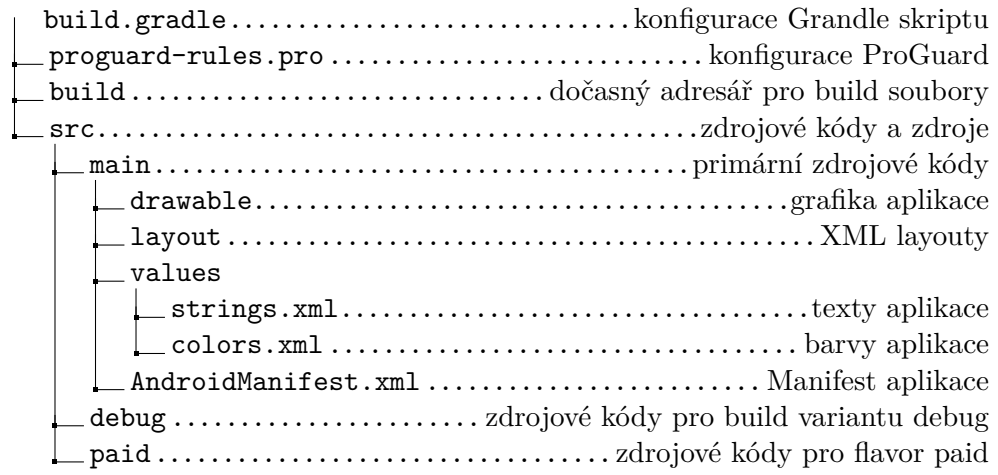
`ProGuard` je vhodný pro `Release build variantu`, protože optimalizuje kód, zmenšuje velikost aplikace a obfuskací znesnadňuje její dekompilaci. Při samotném vývoji není optimální mít tento nástroj stále zapnutý, protože při jeho využití vytvoření sestavení trvá déle.

3.7 Struktura Android projektu

V následující části bude ukázáno, jak může vypadat struktura Android projektu. Jedna z možností je vidět na obrázku 3.3. Nejsou zde zobrazeny všechny složky ani soubory, ale pouze ty základní.

Zajímavý je způsob, jakým se vytvářejí soubory např. pro různé jazykové verze [38]. Základní jazyk, většinou anglický, je v souboru *values/strings.xml*. Pro přidání češtiny se vytvoří složka *values-cs*, která obsahuje opět soubor

[20] Dekompilace je získání původních zdrojových kódu z aplikace [1].



Obrázek 3.3: Struktura Android projektu

strings.xml. Analogicky lze specifikovat, jaké zdroje se použijí pro další jazyky, orientaci displeje, rozlišení displeje apod. Toto pojmenování lze i kombinovat, např. lze určit, jaký obrázek se má zobrazit pro český jazyk při orientaci na šířku. Který zdroj bude nakonec použit, je definováno v dokumentaci [39].

3.7.1 Android Manifest

`Android Manifest`[40] je nadstavba *Java Manifestu* [41], poskytující základní informace o aplikaci, které systém potřebuje předtím, než ji spustí.

Obsahuje název balíčku, název aplikace, komponenty (`Aktivita`, `Broadcast Receivers`, `Services` aj.), oprávnění (viz 3.7.2), minimální verze `level API` aj.

3.7.2 Oprávnění

Pro lepší zabezpečení systému byl vytvořen systém oprávnění [42]. Pokud by aplikace chtěla přistupovat k citlivým údajům nebo by dělala něco, co by mohlo dle tvůrců `Android Frameworku` zmenšit komfort při používání aplikace, je třeba aplikaci udělit oprávnění.

Mezi tato oprávnění patří přístup k internetu, čtení telefonních kontaktů, zápis na paměťovou kartu aj.

Vývojář tato oprávnění deklaruje v `Android Manifestu` a uživatel pak tato oprávnění vidí při instalaci aplikace. Ve starších verzích OS Android je pouze možnost všechna oprávnění přijmout nebo nelze aplikaci nainstalovat. V nové verzi Androidu si aplikace o některá práva zažádá a je na uživateli, zda-li právo udělí nebo ne.

3.7.3 APK

`APK` je výsledný instalační soubor Androidu, podobný *JAR* souboru [43] u Javovského projektu. Jedná se o zip soubor, který obsahuje class soubory, resources, manifest. Každý takový soubor musí být podepsaný. Při vývoji se používá univerzální klíč pro všechny aplikace, ale při `Release build verzi` je nutno vytvořit nový unikátní klíč.

3.8 Příprava verze pro zveřejnění

Před uveřejněním aplikace je doporučené udělat některé kroky, které jsou popsány v tomto oddíle.

3.8.1 Google Play

`Google Play` [44] slouží jako oficiální distribuční kanál pro aplikace a multimediální obsah[21]. Lze odtud aplikace nebo multimediální obsah stahovat, nakupovat a hodnotit.

Aplikace lze instalovat i z neznámých zdrojů přímo instalačním souborem (`APK`), to je však nutné povolit, neboť tato možnost je v základním nastavení z bezpečnostních důvodů vypnuta.

Existují i jiné neoficiální kanály jako je např. *Amazon underground* [45]. Je ovšem nutné povolit výše zmíněné instalování z neznámého zdroje.

3.8.2 Příprava

Jak bylo zmíněno dříve, pro `Release build variantu` je vhodné mít povolený a správně nakonfigurovaný `ProGuard`. Dále je nutné určit, jaká minimální oprávnění aplikace bude potřebovat.

Nutným krokem pro `Release build variantu` je vytvoření podepisovacího klíče. Součástí `Android SDK` je nástroj pro vytvoření a podepsání aplikace. Vytvoření certifikátu lze udělat přes příkazovou řádku nebo ho lze vytvořit přímo v `Android Studiu`. Podepisovací nástroj lze snadno napojit do `Gradle` skriptu.

Dalším nutným krokem, který následuje po podepsání aplikace, je použít nástroj `zipalign`, který zarovná nekomprimovaný kód aplikace do bloku velikosti 4 bajtů, se kterými OS efektivněji pracuje. Tento nástroj je součástí `Android SDK` a lze ho napojit do `Gradle` skriptu.

Mezi další doporučení patří napojení na některý `crash report systém`, který slouží pro hlášení pádů aplikace[22]. I když `Google Play` nabízí základní reporty, bere pouze ty, které uživatelé sami pošlou. K nim mohou připojit libovolnou uživatelskou zprávu. V tomto projektu byla aplikace připravena pro

[21] Hudba, filmy, knihy aj.

[22] Pád aplikace znamená, že aplikace se neočekávaně ukončila.

službu `HockeyApp`[46], která posílá všechny pády aplikace ve formě výpisu zásobníku. Z toho je pak možné určit, na jakém řádku aplikace spadla. Nesmí se ovšem zapomenout, že byla použita obfuskace, tedy názvy metod ani tříd nemusí odpovídat. Naštěstí tato služba dovoluje nahrát mapování z `ProGuardu`, takže se ve výpisu zobrazí správné názvy. V této službě je vidět, na čem aplikace nejčastěji padá, u jaké verze aplikace, na jakém mobilním zařízením, na jaké verzi OS, apod. Nutno dodat, že se neposílají žádné osobní údaje.

Existuje mnoho dalších služeb, kterými lze obohatit aplikaci, např. různé analytické nástroje, přidání reklamy aj.

Posledním krokem je nahrání aplikace do `Google Play`. Zde je nutné vyplnit formulář s popisem aplikace, nahrát obrázky z aplikace aj. Poté si uživatelé mohou aplikaci stahovat.

3.9 Doteky a gesta

Pro ovládání aplikací se na Androidu používají mimo jiné i gesta [47]. V tomto oddíle budou některé z nich představeny, aby bylo později možné se na ně odkazovat. Budou zmíněna pouze ta, která se v aplikaci vyskytují nebo by se mohla vyskytovat. Jsou používány originální názvy z dokumentace, protože pro některé výrazy neexistuje jednoznačný český ekvivalent.

3.9.1 Touch

`Touch` je jednoduchý dotek, který slouží nejčastěji k výběru položky, potvrzení akce aj.

3.9.2 Swipe

`Swipe` je gesto, které připomíná tažení. Provádí se jedním prstem, který se dotkne obrazovky, poté se prst posouvá po obrazovce pouze jedním směrem a nakonec se prst zvedne. Toto gesto se používá např. v galerii pro přechod na další obrázek.

3.9.3 Pinch open

`Pinch open` je gesto pomocí dvou prstů. Prsty jsou většinou blízko sebe, položí se na obrazovku a poté se od sebe po obrazovce oddalují. Toto gesto je často použito pro přiblížení na mapě, zvětšení obrázku aj.

3.9.4 Pinch closed

`Pinch closed` je gesto pomocí dvou prstů a je opakem `Pinch open`. Prsty jsou většinou daleko od sebe, položí se na obrazovku a poté se k sobě po obrazovce přibližují. Toto gesto je často použito pro oddálení na mapě, zmenšení obrázku aj.

Kapitola 4

Analýza a sběr dat

V této kapitole budou popsány aplikace, které se zabývají historií nebo z nich byla čerpána inspirace. Dále bude popsáno, jakým způsobem probíhal sběr dat, který je nedílnou součástí této práce.

4.1 Analýza existujících řešení

Následuje analýza aplikací, které se zabývají podobnou tématikou a bylo by možné se z nich inspirovat. Všechny aplikace byly vyzkoušeny a je možné je stáhnout z `Google Play`. Výjimkou je aplikace z pododdílu 4.1.3, která existuje pouze pro operační systém iOS.

4.1.1 Aplikace Timeline – Art Museum

Aplikace `Timeline - Art Museum` [48] obsahuje galerii známých malířů a informace z jejich života. Lze se dozvědět, jaké rozměry obraz má, jakou technikou byl namalován, v jaké galerii je apod. Aplikace po zaplacení umožňuje stažení obrazů v digitální podobě ve vysokém rozlišení.

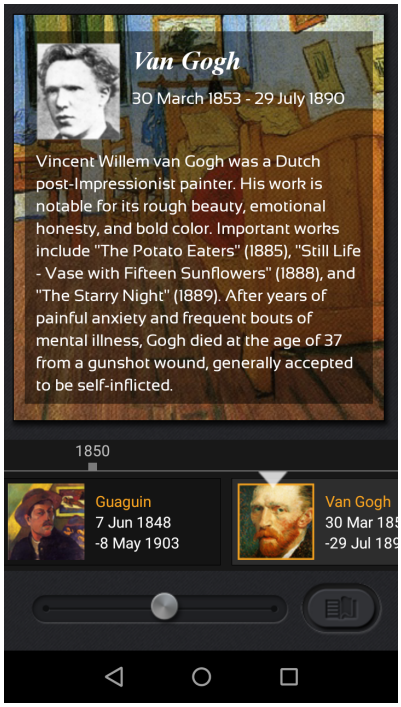
Nabízí designově zajímavý ovládací prvek pro přechod mezi umělci, který je umístěn v dolní části aplikace a reprezentuje časovou osu, kterou lze vidět na `screenshotu`[23] z aplikace 4.1. Funguje jako horizontální skrolovací prvek[24]. Kliknutím na umělce se zobrazí jeho detail. Pro rychlejší přesun po časové ose slouží ovládací prvek, který funguje podobně jako `Seekbar` [49].

Celkově aplikace působí velmi příjemným designem, používá některé netradiční ovládací prvky, přesto je ovládání intuitivní.

Aplikace je vhodná pro zobrazování osobností. Byla inspirací pro použití zobrazení rektorů, kteří jsou seřazeni chronologicky.

[23] Screenshot je snímek obrazovky aplikace.

[24] Prvek, který se dá ovládat gestem pro posunutí (viz 3.9.2).



Obrázek 4.1: Timeline – Art Museum

4.1.2 Aplikace World History

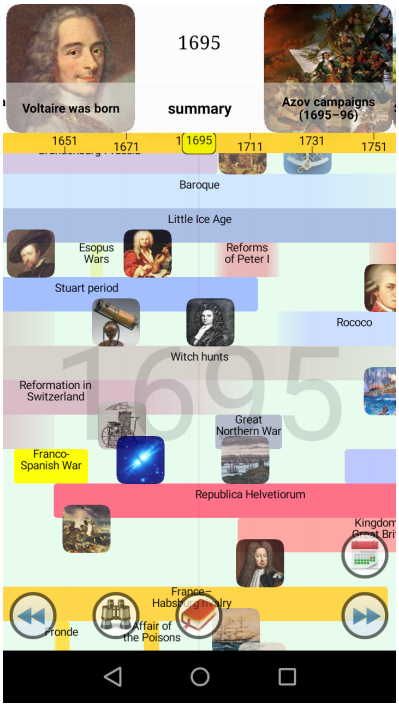
Aplikace `World History`[50] obsahuje historické události z celého světa, které se skládají z popisu a obrázků. Události je možné vyhledávat a lze z nich vytvářet záložky.

Zajímavým prvkem je zde propracovaná časová osa (obrázek 4.2), na které se dá pohybovat horizontálně i vertikálně. V horizontálním směru jsou vidět jednotlivé historické události s časovým intervalem. V případě některých událostí je použit ilustrační obrázek. Vertikální směr je použit pro zobrazení událostí, které se udály ve stejnou dobu. Po dvojitém kliknutí na obrázek se zobrazí detail události.

V horní části obrazovky se nachází lineární zobrazení událostí, pomocí kterého si uživatel může prohlížet události jednotlivě. Posun zde funguje opět horizontálně, po kliknutí na událost se zobrazí její detail. Bohužel po zobrazení události nelze snadno přejít na další událost a uživatel se musí vrátit zpět na časovou osu.

Celkově aplikace působí chaoticky, nepřehledně a používá mnoho barev. Pravděpodobně je to dáno snahou zobrazit mnoho dat najednou.

Tato aplikace může být jistě užitečnou školní pomůckou, protože je zde vidět, jak šly události za sebou nebo které probíhaly současně, ale pro milníky ČVUT se příliš nehodí. Milníků ČVUT není tolik, aby byl při stejném měřítku



Obrázek 4.2: World History

využit vertikální prostor. Při větším měřítku by zase nevyniklo, jak dlouho která událost trvala. Uživatel by musel dlouho skrolovat, pokud by mělo být zachováno lineární zobrazení. Z těchto důvodů nebyl podobný mechanismus použit.

4.1.3 Aplikace Boeing Milestones

Aplikace `Boeing Milestones` [51] mapuje historii letecké společnosti Boeing. Velkým lákadlem jsou kvalitní fotografie letadel.

Milníky Boeingu se nacházejí na časové ose, která je udělaná poměrně netradičně, jak ukazuje obrázek 4.3. Tato osa se dá otevřít `pinch open` gestem (viz 3.9.3), kde je znázorněn ilustrační obrázek a stejným gestem se uživatel dostane na textový popisek. Vše je lépe vidět na videu, které je k dispozici na [52]. Bohužel při testování na iPadu se ukázalo, že aktuální verze na iTunes tomuto videu úplně neodpovídá a často padá. Na videu jsou mnohem hezčí animace a pokud se otevře např. rok 1972, přesune se na 1970, časová osa již neodpovídá.

I přes některé nedostatky jsou nápady použité v této aplikaci velmi kreativní a posloužily jako velká inspirace pro časovou osu milníků ČVUT.

Obrázek 4.3: Boeing Milestones – časová osa

4.1.4 Aplikace Harvard Tour

Společnost *YouVisit* [53] se zabývá kompletním vytvářením virtuálních prohlídek. Jejími klienty jsou restaurace, letiště, školy aj. V této části bude popsáno, jak může vypadat jejich výsledná aplikace na konkrétní univerzitě a to `Harvard Tour` [54].

Aplikace obsahuje mapu univerzity včetně bodů zájmů, které se dají zobrazit. Každý bod zájmu může obsahovat textový popis, zvukový popis, fotky, panorama, videa a `360's`[25].

Nejzajímavější část aplikace je virtuální prohlídka (obrázek 4.4). Ta se skládá ze statických obrázků, mezi kterými se uživatel pohybuje kliknutím na tlačítko další nebo na šipky, ukazující směr dalšího bodu.

Nápad i obsah jsou velmi povedené, ale existují některé nedostatky aplikace. Mezi tyto nedostatky patří, že funguje pouze online, jsou použity staré nativní prvky, tlačítko zpět zavírá aplikaci aj.

Z této aplikace je nejzajímavější virtuální prohlídka, která by se dala realizovat i pro ČVUT.

4.2 Sběr dat

Na sběru dat se podílel autor této práce a Vojtěch Pajer, jak bylo určeno v zadání závěrečných prací. Rozdělení práce je v tabulce 4.1. V tomto oddíle budou zmíněny pouze části, na kterých se podílel autor této práce.

Stojí za zmínku, že pro zjednodušení nejen v tomto oddíle se píše o historii, rektorech, významných osobnostech ČVUT aj. Ve skutečnosti se název ČVUT začal používat až v roce 1920 a některé údaje a osobnosti jsou z doby před tímto rokem. Spíše se jedná o ČVUT a její předchozí instituce.

[25] `360's` je zobrazení zachycující plných 360° prostoru.

Obrázek 4.4: Harvard Tour – Virtuální prohlídka

	Tomáš Krabač	**Vojtěch Pajer**
Milníky ČVUT	x	
Přehled ČVUT	x	x
Přehled fakult ČVUT	x	x
Rektoři		x
Významné osobnosti	x	
Sportovci		x

Tabulka 4.1: Rozdělení práce při sběru dat

4.2.1 Milníky ČVUT

Historie ČVUT začíná od roku 1707 a od té doby se stalo mnoho událostí. Během sběru dat byl kladen důraz na to, aby texty byly krátké, výstižné a byl k nim přiřazen ilustrační obrázek. Náročným úkolem bylo vybrat důležité momenty. Jako hlavní zdroje byly použity knihy [55] a [56] a pro rychlý přehled dobře posloužila webová stránka [57]. Obrázky a novější milníky pocházejí z dalších zdrojů. S výběrem milníků, konzultací textů a s jejich korekturou pomohla Prof. PhDr. Marcela Efmertová, CSc.

4.2.2 Přehled ČVUT

Pro získání základních informací o ČVUT byla použita kniha [55], která obsahuje popis ČVUT. Statistické údaje byly převzaty z webu [58].

4.2.3 Přehled fakult

Ke sběru dat o jednotlivých fakultách jako základ posloužila opět kniha [55], která obsahuje popis jednotlivých fakult. Statistické údaje byly převzaty z webu [16]. Použité obrázky, které charakterizují jednotlivé fakulty, byly vytvořeny v rámci projektu *Sedm statečných* [59].

4.2.4 Významné vědecké osobnosti ČVUT

Významné vědecké osobnosti ČVUT jsou již zpracovány na webu [60], proto byla data převzata s mírnými úpravami, např. byly upraveny obrázky, aby lépe vyhovovaly grafické předloze, která má tvar kruhové výseče.

4.2.5 Obrázky

Většina obrázků pochází ze zdrojů blízkých ČVUT. Jedná se o knihy, na nichž pracovali akademici ČVUT nebo obrázky ze stránek fakult či projektů pro ČVUT. Bohužel ne všechny obrázky měly vhodný formát nebo rozlišení, a proto byly převzaty z jiných zdrojů jako jsou například webové články. Vzhledem k tomu, že se jedná o uzavřenou aplikaci, bylo dohodnuto s vedoucím této práce, že toto není na závadu. Pokud by se aplikace měla zveřejnit, bylo by nutné některé obrázky nahradit nebo vyřešit jejich autorská práva. Zdroje jsou součástí CD k této závěrečné práci.

4.2.6 Umístění dat

Data byla nahrána do webové aplikace [61], kterou vytvořil v rámci své závěrečné práce Ing. Vojtěch Bartoš. Data poskytovaná touto webovou aplikací používají mobilní aplikace v tomto projektu.

Kapitola 5

Návrh

V této kapitole bude popsán `use case`[26] aplikace, ze kterého budou následně vycházet obrazovky aplikace. Dále jsou uvedeny datový model a architektura aplikace.

5.1 Use case

`Use case` základních částí aplikace je na diagramu 5.1, který slouží jako základní přehled, jaké položky aplikace bude nabízet. Tyto položky budou dále podrobněji představeny jak z pohledu obrazovek, tak z pohledu ukládání dat do databáze. Jelikož se nejedná o příliš složité chování aplikace, ale jde většinou o zobrazování dat, nebudou zde dále rozebrány `use case` scénáře [62].

5.2 Wireframe

`Wireframe` je návrh znázorňující rozložení prvků, jejich chování a ukazuje dostupné funkce aplikace [9].

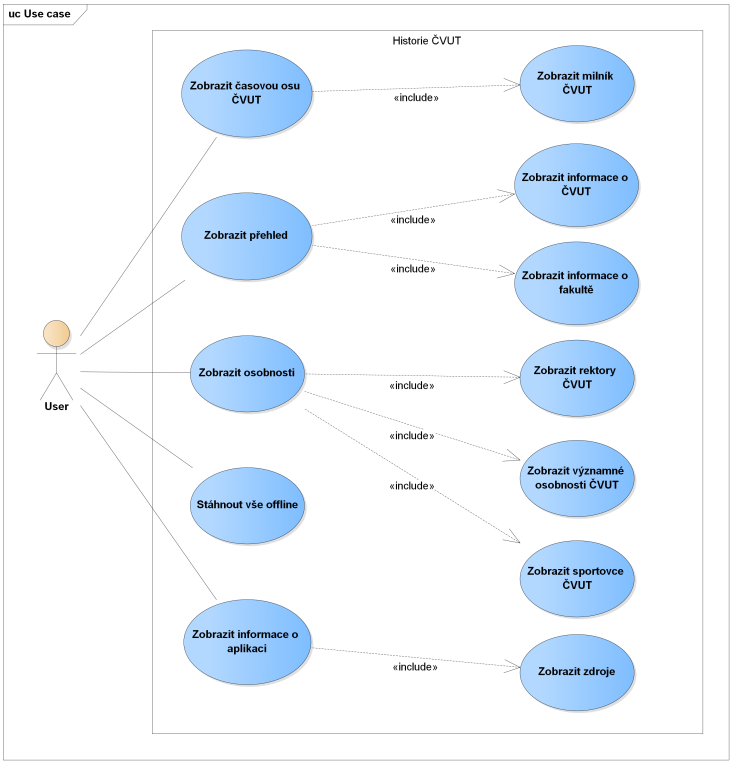
Po analýze a získání základní představy, co by aplikace mohla obsahovat, byly vytvořeny `wireframy` ve spolupráci s kolegou Vojtěchem Pajerem[27], které byly zároveň konzultovány grafikem a UX designérem ze společnosti Ackee.

Pro tvorbu `wireframů` byl vybrán software `Pencil` [63], který je zdarma a obsahuje šablony pro mobilní aplikace, i když v případě Androidu pro starší verzi operačního systému 4.0.

Jaká data budou obrazovky obsahovat byly popsány v oddíle 2.3.

[26] `Use case` (případ užití) pomáhá pochopit, jak bude systém, v tomto případě mobilní aplikace, reagovat. [62]

[27] iOS vývojář tohoto projektu.



Obrázek 5.1: Use case model

5.2.1 Hlavní menu

Aplikace bude mít několik nezávislých obrazovek, pro zobrazení milníků ČVUT, informací o fakultách atd. Proto bylo potřeba určit, jak se mezi nimi uživatel bude pohybovat.

Jednou z prvních variant bylo vytvořit hlavní obrazovku, kde by byl rozcestník. Po konzultaci s UX designérem se ukázalo, že tento způsob se v mobilních aplikací moc nepoužívá a je zastaralý. Výhodou je sice snadná implementace, ale nevýhodou je pro uživatele četnější klikání pro přechod na další obrazovky.

Další možností je použít `Tabs` [64], které v horní části obrazovky zobrazí záložky, mezi kterými je možné přepínat. Pro uživatele je přehlednější, pokud jsou záložky viditelné všechny, tzn. je jich málo. Pokud by jich bylo více, dá se k nim dostat postupným posouváním záložek. Výhodou těchto záložek je, že jsou součástí `Android Frameworku` a rovnou lze zobrazit první položku (např. milníky) na rozdíl od předchozího řešení. Nevýhodou je, že zabírají část místa obrazovky.

Poslední možností, která byla diskutována, byla `Navigační lišta` (Navigation drawer) [65], která se poslední dobou těší veliké oblibě nejen na mobilních zařízeních, ale i u webových aplikací. Funguje tak, že uživatel se do

(a) Navigační lišta (b) Časová osa (c) Detail historie

Obrázek 5.2: Kategorie Historie

`Navigační lišty` může dostat po kliknutí na tlačítko v `Action Baru`[28] nebo může lištu vysunout z levé části obrazovky gestem `swipe` (viz 3.9.2). Poté se objeví nová obrazovka, která částečně překrývá obrazovku předchozí. Skrýt se dá opět gestem, kdy se lišta zasune mimo viditelnou oblast, nebo tlačítkem zpět[29]. `Navigační lišta` je přístupná z více obrazovek v rámci jedné `Aktivity`. Její základní funkcionalita je také součástí `Android Frameworku`.

Z výše uvedených variant nejlépe vyhovuje `Navigační lišta`, která je součástí `Android Frameworku`. Nezabírá zbytečně místo na obrazovce, pokud ji uživatel nepotřebuje. Její konkrétní návrh ukazuje `wireframe` 5.2a, kde položky mají dvě sekce. První sekce obsahuje důležité položky, které nesou nějaký obsah (`Historie`, `Přehled`, `Osobnosti`). V druhé sekci jsou ostatní položky, např. informace o aplikaci, nebo možnost stažení všech dat pro použití bez internetu.

5.2.2 Kategorie Historie

Dalším úkolem bylo určit, jak zobrazovat `kategorii Historie`. Tato kategorie má dvě obrazovky, jedna obrazovka zobrazuje časovou osu s milníky a druhá ukazuje konkrétní milník.

Jako nejzajímavější pro zobrazení časové osy se po analýze existujících aplikací ukázal způsob použitý u aplikace `Boeing Milestones`, zmíněné v pododdíle 4.1.3. Časová osa bude fungovat podobně, ale bude se lišit přechodem

[28] Action Bar [66] je horní lišta obrazovky, kde mohou být umístěné některé ovládací prvky, např. hledání, nastavení aj. Součástí bývá i titulek obrazovky.

[29] Toto je základní chování, které se dá přenastavit. Např. se dá zakázat ovládání gestem, pokud se to nehodí.

(a) Rozcestník (b) Detail ČVUT (c) Detail fakulty

Obrázek 5.3: Kategorie Přehled

na milník. U zmíněné aplikace se přechod na další obrazovku prováděl gestem `pinch open` (viz 3.9.3). To se ukázalo na telefonu jako velmi neobratné. Pro přechod na milník bude použito pouze kliknutí (viz 3.9.1). `Wireframe` časové osy je na obrázku 5.2b.

Obrazovka s milníkem oproti zmíněné aplikaci nebude obsahovat jeden velký obrázek, ale bude vidět obrázek s textem. Při sběru dat se totiž ukázalo, že není k dispozici dostatek obrázků ve velkém rozlišení a že obrázky jsou s různou orientací. `Wireframe` detailu je na obrázku 5.2c. Mezi detailovými obrazovkami bude možný přechod gestem `swipe` (viz 3.9.2).

U detailové obrazovky bude použit `parallax efekt`. Ten funguje tak, že při posouvání obrazovky se obrázek posouvá jinou rychlostí, než zbytek obrazovky, a tím se vytvoří zajímavý vizuální efekt. Tento efekt je nyní velmi populární i u webových aplikací [67].

5.2.3 Kategorie Přehled

`Kategorie Přehled` zobrazuje základní informace o ČVUT a jejích fakultách. Pro rozcestník slouží podobná obrazovka použitá pro `kategorii Historie`, viz obrázek 5.3a.

Existují dva typy detailových obrazovek, první pro ČVUT, druhá pro fakulty. Liší se tím, jaká data zobrazují. Základní návrh obrazovky ukazuje `wireframe` 5.3b. Opět je použit `parallax efekt` a mezi detailovými obrazovkami bude opět možný přechod gestem `swipe`.

(a) Rozcestník (b) Rektoři (c) Seznam osobností (d) Detail osobnosti

Obrázek 5.4: Kategorie Osobnosti

5.2.4 Kategorie Osobnosti

Rozcestník pro osobnosti opět používá podobnou obrazovku jako u `kategorie Historie` a `Přehledu`, viz `wireframe` 5.4a. Obsahuje tři položky – rektoři, významné osobnosti a sportovci. Je navržena tak, že bude možné přidat další položky, např. úspěšné studenty ČVUT, děkany fakult apod.

5.2.5 Rektoři

`Wireframe` rektorů 5.4b má základ v aplikaci zmíněné v pododdíle 4.1.1. Na rozdíl od ní se místo data narození používá pro usazení na časové ose období, kdy byla osoba rektorem. Mezi rektory se dá opět přecházet gestem `swipe` nebo pro rychlejší přechod se dá použít dolní navigační lišta.

Při sběru dat se ukázalo, že někteří rektoři byli jmenováni vícekrát a ne vždy po sobě. Proto byl zvolen tento typ zobrazení, kdy na každé obrazovce je vidět obrázek rektora, kdy byl rektorem v jednom funkčním období a stručné informace o rektorovi. Pokud byl opět zvolen znovu, je zobrazena stejná obrazovka, jen s jiným funkčním období. Tento druh zobrazení byl vybrán kvůli chronologické posloupnosti.

5.2.6 Významné osobnosti a sportovci

Pro významné osobnosti a sportovce byl zvolen jiný typ zobrazení než u rektorů, protože je zde prioritou příjmení, nikoliv datum. Proto se tato obrazovka spíše podobá telefonnímu seznamu (viz obrázek 5.4c), řazenému podle příjmení, kvůli jednoduššímu vyhledávání. Detail obrazovky je na `wireframu` 5.4d.

Obrázek 5.5: Návrh databáze

5.3 Datový model aplikace

Datový model, který je potřeba pro offline použití aplikace, vycházel z dokumentace API [13] a je zobrazen na diagramu 5.5. Oproti datovým strukturám použitým v API došlo ke zjednodušení návrhu pro snadnější práci a rychlejší běh aplikace. Mezi tato zjednodušení patří např. nevytváření další tabulky pro obrázky, ale URL na obrázky jsou již součástí tabulky, kde se obrázky používají.

Do databáze se ukládají pouze textové údaje. Aby mohla celá aplikace po stažení potřebných dat fungovat offline, je potřeba ukládat i obrázky. O to se stará knihovna `Glide`, která bude zmíněna později v pododdíle 6.1.3.

Tento model je poměrně jednoduchý, ale má omezení, že offline funguje pouze jedna jazyková mutace.

5.4 Architektury pro Android

Volba architektury[30] aplikace je při vývoji obecně jedním z důležitých rozhodnutí. V tomto oddíle budou nejdříve popsány techniky, které se využívají v architekturách aplikace, a dále budou představeny tři konkrétní architektury na základě článku [69].

[30] Architektura je základní uspořádání nebo vlastnosti systému vyjádřeného v jeho komponentách a vztazích mezi nimi, včetně vztahu k prostředí a principech, které řídí jeho návrh a rozvoj [68].

5.4.1 Dependency injection

`Dependency injection` (`DI`) je návrhový vzor pro předávání závislostí mezi objekty. Základní myšlenkou tohoto vzoru je, že se určitý objekt (např. objekt, který ukládá data do databáze) nemusí starat, odkud vezme jiné objekty (např. spojení do databáze) [70].

`Android Framework` sám o sobě `DI` nepodporuje, ale existují knihovny, které usnadňují použití tohoto návrhového vzoru. Jednou z nich je i knihovna `Dagger` [71], která byla použita na tomto projektu a ze které budou následující ukázky kódu.

Tato knihovna potřebuje minimálně dvě části – `poskytovatele obsahu` a `konzumenta obsahu`.

`Poskytovatel obsahu` poskytuje dané objekty. K tomu je použita anotace *@Provides*, viz ukázka kódu 5.1. Zde se vrací pouze nový objekt, ale inicializace může být mnohem složitější.

Zdrojový kód 5.1: Ukázka poskytovatele DI

```
1  @Module
2  class ExampleModule {
3    @Provides
4    DatabaseInterface provideDatabase() {
5      return new Database();
6    }
7  }
```

`Konzument obsahu` je třída, která očekává nějaký objekt, viz ukázka 5.2. Zde se anotací *@Inject* říká, že se očekává objekt třídy *Database*. Pro tuto třídu musí existovat `poskytovatel obsahu`.

Zdrojový kód 5.2: Ukázka konzumenta DI

```
1  class ExampleClass {
2    @Inject
3    DatabaseInterface database;
4  }
```

ExampleClass se nemusí starat o inicializaci třídy *Database* a snadno ji lze vyměnit za testovací třídu, která bude mít stejné rozhraní. Zde je uveden jednoduchý příklad, ale existují i složitější případy, kde inicializace objektu obsahuje další objekty využívající `DI`. O vytváření stromu závislostí se pak stará knihovna, která případně nahlásí chybu během kompilace, pokud by nějaké vazby chyběly nebo byl vytvořen cyklus.

Výhodou `DI` je, že objekt, který ji používá, se nemusí starat, kde se jeho závislé části inicializují. O inicializaci se stará ten, kdo objekt poskytuje. Díky tomu je kód snadněji udržovatelný, lépe testovatelný a čitelnější.

5.4.2 Reaktivní programování a knihovna RxJava

Podle článku [72] je reaktivní programování definováno jako programování s asynchronními datovými proudy. V tomto pododdíle bude na základě série článků [73] popsáno reaktivní programování.

5.4.2.1 Základní pojmy

Rx (Reactive Extensions) je jedna z implementací reaktivního programování, která má podporu mnoha jazyků. Jedním z nich je i Java, kde se konkrétní knihovna nazývá RxJava [74]. Analogicky existují knihovny i pro ostatní jazyky, mezi které patří JavaScript (RxJS), Scala (RxScala) a další. Pro další účely této práce bude používána knihovna RxJava.

Datový proud (data stream) je sekvence dat obsahující prvky, kterými mohou být proměnné, vstupy od uživatele, datové struktury aj. Datový proud je vytvářen asynchronně, to znamená, že data do něj přicházejí, až když jsou k dispozici (např. uživatel postupně zadává data nebo jsou obrázky postupně stahovány z internetu).

Objekt Observable vytváří prvky datového proudu. Příkladem objektu Observable může být např. generátor náhodných čísel nebo obrázky stažené z internetu.

Objekt Subscriber zpracovává datový proud tím, že se k němu přihlásí (je zavolána metoda *subscribe*). Do té doby objekt Observable negeneruje žádná data. Objekt Subscriber má tři základní metody, které jsou volány objektem Observable:

- metoda **onNext** – metoda je vykonána ve chvíli, kdy objekt Observable vytvořil další prvek datového proudu. V parametru je předán objekt, který byl vytvořen.
- metoda **onError** – metoda je vykonána v případě chyby. Parametrem je objekt *Throwable*, neboli třída pro zpracování chyb.
- metoda **onCompleted** – metoda se vykoná na konci datového proudu. Nemá žádné parametry.

Operátor slouží k transformaci prvků datového proudu mezi objektem Observable a objektem Subscriber. Těchto operátorů může být v datovém proudu více. Operátor je prakticky další objekt Observable, který transformuje prvek předchozího Observable objektu. Mezi operátory patří např. filtry, převody na jiné datové proudy, spojení více datových proudů apod.

5.4.2.2 Jednoduchý příklad reaktivního programování

Na jednoduchém příkladě 5.3 budou vysvětlené pojmy popsány. Pro kratší zápis je použit lambda výraz, vysvětlený v pododdíle 3.6.2.

Na prvním řádku vznikl objekt `Observable`, který vytvoří `datový proud` pouze o jednom prvku a to řetězci „Hello, RxJava!“. Tento objekt byl vytvořen metodou *just*.

Další řádek používá `operátor` *map*, který z původního objektu `Observable` vytvoří další objekt `Observable`. Ten může být stejného datového typu, nebo jako v tomto případě vytvoří objekt třídy *SimpleClass*.

Poslední řádek představuje objekt `Subscriber`, který se přihlásil k odběru tohoto `datového proudu`. `Subscriber` zavolá metodu pro výpis třídy *SimpleClass*.

Zdrojový kód 5.3: Ukázka RxJava – jednoduchý příklad

```
1 Observable.just("Hello, RxJava!")
2             .map(string -> new SimpleClass(string))
3             .subscribe(simpleClass -> print(simpleClass));
```

Na uvedeném příkladě se může zdát, že je složitější oproti klasickému imperativnímu přístupu. Důležité je uvědomit si následující. `Observable` může být cokoliv, od databázového požadavku po stisk tlačítka. Prvky do `datového proudu` mohou přicházet kdykoliv a je vidět, co se s nimi děje od začátku až do konce. Mezi objektem `Observable` a `Subscriber` může být mnoho `operátorů`, které se dají snadno poskládat.

Pro zjednodušení zápis *simpleClass -> print(simpleClass)* lze nahradit zápisem *this::print*, kdy *this* může být nahrazeno objektem, nad kterým se provádí daná metoda. Tento zkrácený způsob zápisu bude používán v dalších částech pododdílu.

5.4.2.3 Další operátory reaktivního programování

Při reaktivním programování se používá velké množství `operátorů`, jejich seznam pro implementaci Rx je k dispozici v dokumentaci [75] a lze přidávat také vlastní `operátory`. Následuje popis některých `operátorů`.

Operátor `first` předá dál pouze první prvek z celého `datového proudu` a lze nastavit podmínku, kterou tento prvek musí splňovat. Analogicky funguje operátor `last`.

Operátor `take` předá dál pouze prvních n prvků z celého `datového proudu`, kdy n je parametr funkce. Analogicky funguje operátor `skip`, který přeskočí prvních n prvků.

Operátor `min` předá dál nejmenší prvek z celého `datového proudu`. Pro určení nejmenšího prvku je použit komparátor třídy. Analogicky funguje operátor `max`.

Operátor `reduce` funguje jako agregátor. Na začátku vezme první a druhý prvek a provede s nimi nějakou operaci. Výsledek této operace je další prvek, se kterým se opět provede stejná operace jako se třetím prvkem. Takto se pokračuje pro všechny prvky a výsledkem je pouze jeden prvek. Jednoduchý

příklad je sečtení čísel 1, 2 a 5, kdy se nejprve sečte 1 + 2, výsledkem jsou 3. V dalším kroku se sečte 3 + 5 a celkovým výsledkem je 8.

Operátor `filter` slouží pro filtrování prvků, parametrem je funkce. Pokud výsledkem této funkce je hodnota *true*, resp. *false*, prvek je předán dál, resp. není předán dál.

Operátor `doOnNext` vezme prvek a ho předá dál. Parametrem tohoto operátoru je funkce, která provede nějakou akci. Operátor se může hodit například pro uložení dat do databáze, logování aj., kdy není potřeba upravovat prvek v `datovém proudu`.

Operátor `merge` spojuje více datových proudů. Prvky do tohoto operátoru přicházejí z více datových proudů nezávisle na sobě a vzniká z nich jeden datový proud.

Operátor `zip` také spojuje více datových proudů. Zpracování prvků proběhne až ve chvíli, kdy jsou všechny prvky ze zdrojových datových proudů k dispozici.

Operátor `retry` provede v případě chyby nové odebíraní `datového proudu` od zdrojového objektu `Observable`, jako kdyby se zavolalo poprvé. Parametrem metody je počet pokusů opakování.

5.4.2.4 Běh operací v Androidu

Android aplikace běží po spuštění v tzv. `hlavním vlákně`. Toto vlákno zpracovává události z uživatelského prostředí, vykresluje komponenty, interaguje s UI komponentami aj. Je důležité, aby v `hlavním vlákně` běžely krátce trvající procesy. Pokud by toto vlákno bylo zablokováno například složitým výpočtem v řádu minut, nejdříve by aplikace přestala reagovat na ovládání uživatelem a později by aplikaci operační systém ukončil.

Pro dlouhotrvající operace Android aplikací se používají tzv. `pracovní vlákna`. Mezi tyto operace patří práce s databází, složité výpočty, stahování dat z internetu aj. Operace `pracovního vlákna` nemohou interagovat s UI komponentami. To má za následek, že z `pracovního vlákna` nelze například změnit hodnotu v `Textview` [76].

Knihovna `RxJava` na toto pamatuje a lze určit, na kterém vlákně poběží daná část kódu. Operátor `subscribeOn` určuje, na kterém vlákně poběží následující `Observable` objekty. Naproti tomu operátor `observeOn` říká, na kterém vlákně bude odebírat objekt `Subscriber`. Jelikož `Observable` objekty většinou zpracovávají dlouhotrvající operace, operátor `subscribeOn` má často nastavený argument `pracovní vlákno`. Naopak `Subscriber` často slouží pro zobrazování dat a proto objekt operátor `observeOn` mívá nastavený argument `hlavní vlákno`. Knihovna `RxJava` nabízí vytvoření různých `pracovních vláken`. Mezi ně patří vlákna pro vstupně/výstupní operace, výpočetní operace aj.

5.4.2.5 Ošetření chyb reaktivního programování

Ošetřování chyb lze řešit metodou *subscribe*, která může mít jako další argument funkci pro ošetření chyb. V ukázce 5.4 je v metodě *subscribe* přidána metoda *onErrorMethod*, kterou by se dále ošetřily všechny chyby, jež by mohly v celém `datovém proudu` nastat. To lze přirovnat k ošetřování výjimek v Javě pomocí bloků *try* a *catch*.

Zdrojový kód 5.4: Ukázka RxJava – ošetření chyb

```
1 Observable.just("error handling")
2         .subscribe(this::print, this::onErrorMethod);
```

Chyby lze ošetřovat také v průběhu `datového proudu`. K tomu slouží další `operátory`. Mezi ně patří operátor `onErrorReturn`. Ten v případě chyby pošle do `datového proudu` jiný prvek. To se hodí například pokud nastala chyba při stahování dat. Místo skutečných dat budou poslána fiktivní data.

5.4.2.6 Praktický příklad reaktivního programování

Na stažení informací o uživateli z internetu bude ukázán rozdíl mezi klasickým přístupem řešení asynchronních událostí a použitím knihovny RxJava. Tento příklad používá knihovnu *Retrofit* pro komunikaci s webovým API, která bude podrobněji vysvětlena v pododdíle 6.1.2, ale základní myšlenky budou vysvětleny zde.

V ukázce s klasickým přístupem 5.5 objekt `retrofit` používá metodu *getUser* pro získání uživatele. Odkud se tato data stáhnou a jak se předá argument *userId* metody *getUser*, není součástí ukázky kódu. Druhým argumentem metody *getUser* je objekt Callback<User>(), který má dvě metody. Metoda *succes* vrací asynchronně data o uživateli v parametru *user*. Metoda *failure* se provede v případě chyby.

Zdrojový kód 5.5: Ukázka Retrofit - klasický přístup

```
1  retrofit.getUser(userId, new Callback<User>() {
2    @Override
3    public void success(User user, Response response) {
4      storeUser(user);
5      printUser(user);
6  }
7
8    @Override
9    public void failure(RetrofitError retrofitError) {
10     printError(retrofitError);
11   }
12 });
```

Výše uvedená ukázka má několik nedostatků. Samotné provedení stažení dat se sice provádí na `pracovním vlákně`, ale metoda `success` běží na `hlavním vlákně`. Pokud je zde vyžadováno uložení dat do databáze, je potřeba vytvořit další asynchronní volání na `pracovním vlákně`. S tím souvisí to, že pokud by bylo potřeba se staženými daty dále pracovat, provádět některé složitější operace a nakonec data zobrazit, byl by zde podobný problém s přepínáním mezi `pracovním` a `hlavním vláknem`. Dále by se obtížně řešila situace, kdyby se požadavek stahování dat nepovedl a bylo by vhodné ho zkusit provést znovu.

Oproti tomu ukázka 5.6, využívající knihovnu RxJava, zmíněné problémy nemá. Metoda *getUser* zde vrací objekt `Observable`, konkrétně datového typu *Observable<User>*. Následuje sekvence operací, které se mají provést. V ukázce je vidět, jak za sebou sekvence následují a na jakém vlákně se mají provádět. Pokud by bylo potřeba zopakovat volání v případě neúspěchu, pouze by se použil operátor *retry*. Kód je kratší s použitím lambda výrazů a se základními znalostmi knihovny RxJava se lépe čte a snadněji rozšiřuje.

Zdrojový kód 5.6: Ukázka Retrofit - použití knihovny RxJava

```
1 retrofit.getUser(userId)
2         .subscribeOn(Schedulers.io())
3         .doOnNext(storeUser)
4         .observeOn(AndroidSchedulers.mainThread())
5         .subscribe(this::printUser, this::printError);
```

5.4.2.7 Shrnutí práce s reaktivním programováním

Práce s `reaktivním programováním` byla vysvětlena na konkrétní knihovně RxJava a byly popsány základní možnosti této knihovny.

`Reaktivní programování` může zjednodušit a zpřehlednit zdrojový kód. Řeší problém s daty, která přicházejí asynchronně a je potřeba s nimi provádět některé operace. Díky používání `operátorů` je zdrojový kód snadno rozšiřitelný. V případě Androidu knihovna řeší efektivně problém s `pracovním` a `hlavním vláknem`. Zároveň dokáže elegantně řešit některé druhy problémů, které by se při klasickém iterativním způsobu řešili obtížně. K tomu je potřeba trochu jiné přemýšlení než u klasického procedurálního programování. Reaktivní programování je moderní trend, přesto není tolik rozšířené a tím se může stát pro neznalé programátory zdrojový kód nečitelný.

Knihovna `RxJava` má v případě Androidu řadu využití. Na tomto projektu byla použita tato knihovna pro komunikaci s webovým API a pro načítání/ukládání dat do databáze. Obecně se hodí v případech, kdy je potřeba reagovat na události, které přicházejí asynchronně nebo při řízení toku dat.

5.4.3 Naivní architektura

Nejhorší ukázkou architektury je naivní architektura, kdy se jedna třída stará o vše – o zobrazování dat, komunikaci s databází a výpočty. Tato architektura je těžko udržovatelná a je obtížné znovu použít jen některé její části. V případě Androidu by existovala jen jedna třída `Aktivity`.

5.4.4 View-Model architektura

Lepší architektura je rozdělení aplikace na vrstvy, kde každá z nich má svou odpovědnost. Většina Android aplikací používá dvouvrstvou `View-Model` (`VM`) architekturu, která je pro `Android Framework` nejpřirozenější.

Vrstva `Model` je datová vrstva, která se stará o komunikaci s databází, API apod. a obsahuje `business logiku`[31].

Vrstva `View` se stará o zobrazení dat a interakci s uživatelem. Tato vrstva zároveň komunikuje s vrstvou `Model`.

U Androidu vrstvu `View` většinou představují třídy `Aktivita` nebo `Fragment`. Obě zmíněné třídy mají svůj životní cyklus, který se může kdykoliv celý restartovat. S tím souvisí, že pokud se budou provádět některé dlouhotrvající operace, musí se s tímto životním cyklem počítat, protože některé objekty v době dokončení operace nemusí existovat. Kvůli tomu se tato vrstva stává komplexní a často končí tím, že všechno je propojené se vším.

5.4.5 Model-View-Presenter architektura

Architektura `Model-View-Presenter` (`MVP`) je pokročilejší tím, že propojuje vrstvy `Model` a `View` vrstvou `Presenter`.

Vrstva `Model` zůstává stejná jako u předchozí architektury. Vrstva `View` se zjednodušila, protože na rozdíl od předchozí architektury nekomunikuje přímo s vrstvou `Model`. Má většinou jen jednu vrstvu `Presenter`. Vrstva `Presenter` komunikuje s vrstvou `Model` pomocí tzv. `interaktorů`. K tomu se hodí `DI` (viz 5.4.1), protože více `Presenterů` bude používat `interaktor` například pro komunikaci s databází [77].

Vrstva `Presenter` je v případě knihovny `Nucleus` [78], která implementuje architekturu MVP pro Android, odstíněna od událostí souvisejících s životním cyklem `Aktivity` nebo `Fragmentu` a díky tomu se vrstva `View` velmi zjednodušila. Autor knihovny doporučuje použít knihovnu `RxJava` pro řízení toku dat ve vrstvě `Presenter` [69].

Díky rozdělení aplikace na vrstvy je tato architektura čitelnější, protože je jasné, co jednotlivé části dělají. Pokud je potřeba upravit zobrazování dat, bude se pracovat s vrstvou `View` apod. Zároveň se jednotlivé části dají snadněji opakovaně použít.

[31] `Business logika` je zodpovědná za rozhodování, transformaci dat apod.

5.4.6 Shrnutí architektur

Existuje velké množství architektur, v této práci byly představeny tři z nich. Nejjednodušší `Naivní architektura` se pro tuto práci nehodila.

V projektu byla použita architektura `View-Model`, která byla později na většině míst přepsána na architekturu `Model-View-Presenter`, čímž došlo ke zpřehlednění v uspořádání kódu. Na některých místech však byla architektura `View-Model` ponechána, především na místech, kde se pouze zobrazovala data.

Kapitola 6

Realizace

Tato kapitola pojednává o tom, jak probíhala realizace aplikace. Nejdříve jsou ukázány některé použité knihovny a poté jsou rozebrány problematické části implementace.

6.1 Knihovny

Stalo se již běžnou praxí, že během vývoje se kromě oficiálního `Android Frameworku` používá i řada knihoven třetích stran.

To na jednu stranu může přinést časovou úsporu nebo větší šanci, že kód je napsán správně a efektivně.

Na druhou stranu zde hrozí riziko, že autor knihovnu přestane podporovat a nikdo nebude opravovat její chyby. Také se může stát, že v nové verzi Androidu nebude knihovna vůbec funkční. Některé knihovny jsou poměrně komplikované a naučit se je používat je časově náročné. S tím souvisí, že pokud převezme kód někdo jiný a není seznámen s touto knihovnou, může být pro něj kód hůře čitelný. Další nevýhodou je, že pokud by bylo použito moc knihoven, roste počet metod tříd. Díky tomu se může dosáhnout limitu 65 536 metod a s tím souvisejících omezení [79].

Naštěstí v případě Androidu existuje mnoho spolehlivých opensource knihoven, které se používají delší dobu i na komerčních projektech.

6.1.1 Butter Knife

`Butter Knife` [80] je knihovna, která zpřehledňuje a zkracuje kód. Její hlavní použití je při získávání referencí na `View` z `Layoutu` za použití anotací.

Pro srovnání je použit kód bez použití knihovny 6.1, kde na řádku 1 je deklarace proměnné a na řádku 12 je přiřazení do proměnné. To samé je v ukázce 6.2 knihovnou zjednodušeno na jeden řádek s číslem 1. Nutno dodat, že je třeba knihovnu inicializovat na řádku 12.

Zdrojový kód 6.1: Ukázka knihovny Butter Knife – bez použití knihovny

```
1  TextView mTxtName;
2  ImageView mImgLogo;
3
4  @Override
5  public View onCreateView(LayoutInflater inflater,
6                           ViewGroup container,
7                           Bundle savedInstanceState) {
8    View rootView = inflater.inflate(R.layout.fragment,
9                                     container,
10                                    false);
11
12   mTxtName = (TextView) rootView
13                          .findViewById(R.id.txt_name);
14   mImgLogo = (ImageView) rootView
15                          .findViewById(R.id.img_logo);
16
17   return rootView;
18 }
```

Zdrojový kód 6.2: Ukázka knihovny Butter Knife – s použitím knihovny

```
1    @Bind(R.id.txt_name) TextView mTxtName;
2    @Bind(R.id.img_logo) ImageView mImgLogo;
3
4    @Override
5    public View onCreateView(LayoutInflater inflater,
6                             ViewGroup container,
7                             Bundle savedInstanceState) {
8      View rootView = inflater.inflate(R.layout.fragment,
9                                       container,
10                                      false);
11
12     ButterKnife.bind(this, rootView);
13     return rootView;
14   }
```

Pro ještě větší zjednodušení existuje rozšíření [81] do `Android Studia`, které tento kód z vybraného `Layoutu` generuje.

6.1.2 Retrofit

Jednou z velice známých knihoven je `Retrofit` [82], která usnadňuje komunikaci s REST API. Umí mapovat JSON[83] formát do `POJO`[32] a naopak. Zvládá všechny základní HTTP požadavky – GET, POST, PUT a DELETE včetně manipulace hlavičky a URL.

Ukázka definice API `endpointu` 6.3 ukazuje, jak je tato definice minimalistická. Pro úplnost bude dále vysvětleno, co znamenají jednotlivé parametry. Parametr *@Path("id") String userId* dosazuje hodnotu atributu *userId* do adresy endpointu[33], kdy *@Path("id")* odpovídá části adresy *{id}*.

Třída *UserResp* je `POJO`, do které se namapují hodnoty z JSON[34] podle odpovídajících atributů. Jak by vypadala nejjednodušší definice, je v ukázce 6.4, kde je odpověď z API ve formátu JSON.

Tato metoda vrací kolekci `Observable` objektů[35], tzn. používá se knihovna `RxJava` a lze použít `reaktivní programování`.

Zdrojový kód 6.3: Ukázka Retrofitu

```
1 @GET("/users/{id}")
2 Observable<UserResp> getUser(@Path("id") String userId)
```

Zdrojový kód 6.4: Ukázka Retrofitu Response

```
1  //JSON
2  {
3    "name": "Jan",
4    "surname": "Novak",
5  }
6
7  //Class
8  public class UserResp {
9    public name;
10   public surname;
11 }
```

6.1.3 Glide

Knihovna `Glide` [85] usnadňuje načítání a cachování obrázků z internetu. V ukázce kódu 6.5 si lze všimnout, jak snadno lze načíst obrázek ze zadané URL do `ImageView`.

[32] POJO je označení pro obyčejnou třídu Javy, která neimplementuje žádné rozhraní, nedědí od žádného předka a obsahuje většinou jen gettery nebo settery [84].

[33] Endpoint je URL, na kterou se připojuje klientská aplikace a odkud se získávají data.

[34] Lze použít i jiné formáty, např. XML.

[35] Starší způsob fungoval pomocí tzv. callbacků.

Zdrojový kód 6.5: Ukázka knihovny Glide

```
Glide.with(getActivity())
.load("http://www.example.com/image.jpg")
.into(imgLogo);
```

6.1.4 Sqlbrite

Knihovna `Sqlbrite` [86] zaobaluje standardní komponenty `Android Frameworku`, které se starají o práci s databází. Tato knihovna dovoluje používat knihovnu `RxJava` pro `reaktivní programování`.

6.1.5 Sqlbrite DAO

Knihovna `Sqlbrite DAO` [87] je rozšíření předchozí knihovny o dvě části, `Data Access Object` (`DAO`) a `ObjectMapper`.

`DAO` je návrhový vzor, který přidává další vrstvu, která odděluje logiku aplikace od datové vrstvy. Např. *CustomerDAO* bude mít metody pro přidání zákazníka, editaci zákazníka a smazání zákazníka. Tyto metody budou nezávislé na implementaci a poté je jedno, jestli se data budou načítat z databáze, internetu aj.

`ObjectMapper` je `mapper`, který generuje kód za pomocí anotací pro snadnější práci s `ContentValues`[36]. V ukázce 6.6 byla vygenerována třída *CustomerMapper*. Je vidět, že při použití tohoto `mapperu` je kód přehlednější a díky typové kontrole bezpečnější.

Zdrojový kód 6.6: Ukázka ObjectMapper

```
//klasicky zpusob
ContentValues cv = new ContentValues();
cv.put("id", id);
cv.put("firstname", "Hannes");
cv.put("lastname", "Dorfmann");

//s pouzitim ObjectMapper
ContentValues cv = CustomerMapper.contentValues()
                                 .id(1)
                                 .firstname("Hannes")
                                 .lastname("Dorfmann");
```

6.1.6 Dagger

Knihovna `Dagger` [71] je implementace `Dependency injection`, která byla popsána v pododdíle 5.4.1.

[36]Kolekce, která obsahuje dvojice hodnot sloupec v databázi a její hodnota

6.1.7 Nucleus

Knihovna `Nucleus` [78] usnadňuje implementaci `MVP` na platformě Android, která byla popsána v pododdíle 5.4.5. Jako hlavní výhodu této knihovny autor uvádí odstínění vrstvy `Presenter` od událostí související s životním cyklem `Aktivity` nebo `Fragmentu`

6.1.8 TwoWayView

Knihovna `TwoWayView` funguje podobně jako `RecycleView`, která je součástí `Android Frameworku`. Důvod jejího použití bude vysvětlen v pododdíle 6.5.2.

6.1.9 Calligraphy

Knihovna `Calligraphy` [88] dovoluje snadno přidat vlastní fonty ve formátu ttf/otf. Komponenta `TextView` má atributy, kterými lze nastavit pouze fonty ze základní sady Android fontů.

6.2 Ackee šablona

Při vývoji byla použita Ackee šablona pro `MVP` architekturu. Tato šablona obsahuje jednoduchý projekt se základní kostrou programu s `MVP` architekturou. Nakonfigurovaná je i Java 1.8, jsou připraveny základní knihovny, je nakonfigurovaný skript `Grandle`, je nastavený nástroj `Proguard` apod.

Šablona zároveň obsahuje upravené třídy `Aktivit` a `Fragmentů` pro rychlejší implementaci běžných scénářů obrazovek, sadu utilit pro běžné programovací problémy pro Android, mezi které patří např. zjištění, jestli je zařízení připojeno k internetu aj.

6.3 Realizace na základě PSD

Realizace vycházela z grafických podkladů firmy Ackee. Grafické podklady byly ve formátu PSD[37]. Z těchto podkladů lze získat barvy, ikonky, rozměry elementů, druhy písma, velikosti fontů, vzdálenosti mezi elementy aj.

Za zmínku stojí, jak lze získat z grafických podkladů např. velikost tlačítka. Problém spočívá v tom, že obrázek měří vzdálenosti v pixelech, cm apod., zatímco na Androidu se doporučuje používat tzv. Density-independent pixel (DP), tedy pixely nezávislé na hustotě displeje. To v praxi znamená, že tlačítko bude mít na obrazovce vždy stejnou velikost, i když budou mít dvě obrazovky různou hustotu pixelů.

Pro DP platí vztah $PX = DP \cdot \frac{DPI}{160}$, kde DPI je hustota zařízení. Grafik použil šablonu pro rozlišení xxhdpi (extra-extra-high), což je podle [90] přibližně 480 DPI. Po dosazení do rovnice vyplývá, že 1 DP je při této hustotě

[37] PSD je datový formát pro uchování grafických dat [89].

přibližně 3 PX a na základě toho lze již dopočítat, jak velké má být tlačítko, jak moc daleko mají být elementy od sebe aj.

6.4 Navigační lišta

`Navigační lišta` byla vybrána jako hlavní menu, odůvodnění je v pododdíle 5.2.1. Během realizace se zjistilo, že je nutné rozhodnout, zda použít oficiální knihovnu, knihovnu třetí strany nebo si udělat vlastní implementaci `Navigační lišty`.

Oficiální knihovna `Android Design Support Library` obsahuje komponentu `Navigation View`. Tímto způsobem lze velmi rychle implementovat `Navigační lištu` na pár řádcích kódu a budou dodržena základní pravidla pro správný návrh designu. Problém nastává ve chvíli, kdy je třeba se řídit designem podle grafické předlohy, kde je použit jiný font, jiné mezery, jiný oddělovač apod. Přizpůsobení vzhledu je u této knihovny v současné době problém.

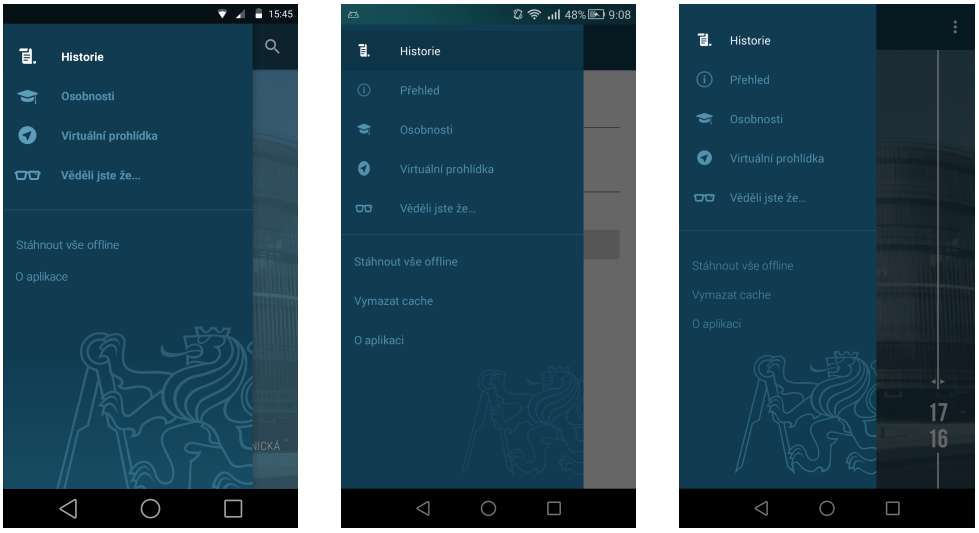
Další možností je použít knihovnu třetí strany. Těch existuje mnoho a některé jsou více přizpůsobitelné nebo nabízejí více možností, např. změnu velikosti písma. Bohužel nebyla nalezena knihovna, kterou by se dalo snadno dosáhnout výsledků, aby `Navigační lišta` vypadala dle předlohy.

Poslední možností je vytvořit si vlastní `Navigační lištu`. Ta se nemusí dělat úplně od začátku, neboť základ pro vysouvací postranní lištu již existuje [91]. Chybí zde `business logika`, např. když se klikne na druhou položku, měla by se předchozí deaktivovat apod. Tato možnost je časově nejnáročnější, ale lze tím dosáhnout vzhledu, který je velmi podobný grafickému návrhu.

Srovnání je vidět na obrázcích 6.1, kde 6.1a je grafická předloha, jak by měl výsledek ideálně vypadat. Na obrázku 6.1b je vidět, že řádky mají vždy stejné odsazení, je zde použit jeden font na rozdíl od předlohy, která má pro každou sekci jiný font apod. Poslední obrázek 6.1c se již blíží grafické předloze, i když má své nedostatky kvůli zvolené implementaci. Lze si všimnout, že logo aplikace není připnuté ke spodní části obrazovky. To je dáno tím, že telefony mají různá rozlišení a pokud by se trvalo na přichycení k dolní části obrazovky a zachování velikosti loga, na některých zařízeních by logo zabíralo velkou část obrazovky a nebyly by vidět všechny položky.

6.5 Boeing efekt

Nejnáročnější část z celé implementace bylo napodobení efektu, který byl použit v aplikaci `Boeing Milestones`, probranou v pododdíle 4.1.3, konkrétně efekt pro zobrazení časových událostí. Zde se postupně ukazovaly problémy, které bylo nutné vyřešit a které zde budou postupně probrány.



(a) Grafická předloha (b) Navigation view (c) Vlastní implementace

Obrázek 6.1: Srovnání různých implementací navigační lišty

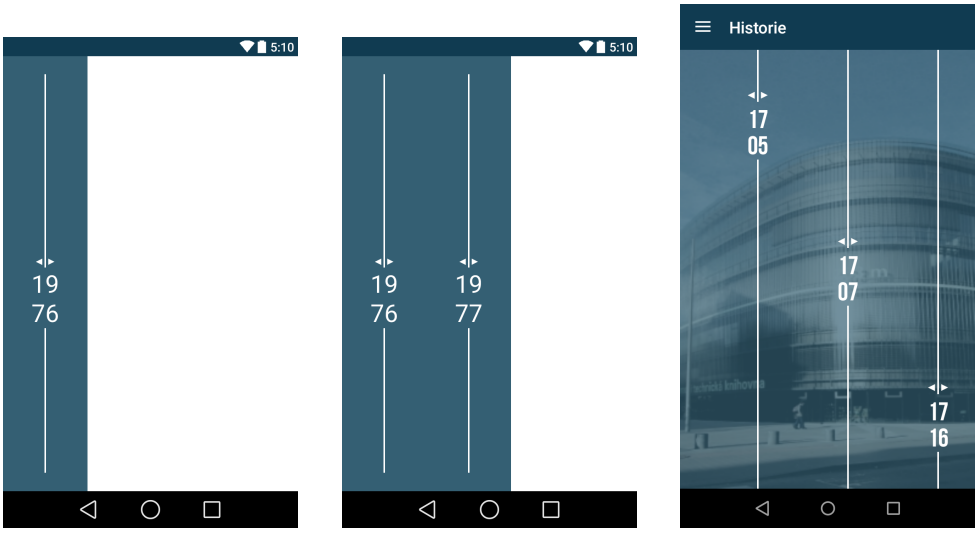
6.5.1 Zobrazení dat

První problém byl, jak efektivně zobrazovat větší počet dat (např. letopočty na časové ose), které se budou zobrazovat horizontálně. Existuje komponenta `HorizontalScrollview`, která všechny komponenty `View` uvnitř sebe dovoluje skrolovat. To ovšem není příliš elegantní způsob, protože všechny komponenty `View` se pak drží v paměti a spíše se doporučuje použít komponentu `RecyclerView`, která recykluje své komponenty `View` a tím šetří čas pro zobrazení i paměť zařízení. V době začátku implementace tato komponenta ještě neexistovala a podobná komponenta `ListView` neuměla horizontální orientaci. Proto byla použita knihovna `TwoWayView`, zmíněná v pododdíle 6.1.8.

S tím souvisí další věc, jak poskládat komponenty `View`, aby vždy představovaly jeden rok. To je docíleno tím, že se vytvořil jeden `Layout`, který se vždy opakuje. Jeho konkrétní zobrazení je na obrázku 6.2, kde na prvním obrázku 6.2a je zobrazení jednoho roku. Druhý obrázek 6.2b znázorňuje přidání dalšího roku. Na posledním obrázku 6.2c je poté ukázka, jak vypadá výsledek v aplikaci. Pro lepší vizuální dojem je na začátku a na konci časové osy přidána výplň. Zároveň na pozadí je ilustrační obrázek.

6.5.2 Animace

Vytvoření animace, která se spustí po kliknutí na rok, se ukázala jako velká komplikace. Animace vypadá podobně jako otevření opony v divadle. Obrazovka 6.2c se rozdělí na dvě části, kde střed je bílá linka, levá část se posouvá vlevo a pravá část se posouvá vpravo.



(a) Jedna položka (b) Dvě položky (c) Výsledek

Obrázek 6.2: Časová osa

Inspirace, jak vytvořit tento efekt, byla převzata z článku [92]. Tento princip bude popsán v několika krocích:

1. Uživatel klikne na rok.
2. Vytvoří se snímek `Layoutu`, na kterém je časová osa.
3. Tento snímek se rozdělí na dvě části od středu, podle toho, kde uživatel klikl na rok. V případě roku 1705 na obrázku 6.2c je to první linka.
4. Vytvoří se nová obrazovka, která bude obsahovat detail roku a bude mít dvě komponenty `ImageView`, které překrývají detailovou obrazovku.
5. Levá komponenta `ImageView` bude reprezentovat „levou část opony“, pravá komponenta `ImageView` bude reprezentovat „pravou část opony“.
6. Současně se spustí animace nad oběmi komponentami `ImageView`, která „otevře oponu“.

`Android Framework` dovoluje animovat komponentu `View` mnoha způsoby. V této práci byl použit způsob, kdy se komponentě `View` předají parametry, na kterých souřadnicích má skončit a jak dlouho se má animovat. Zároveň umí spustit více animací najednou.

Analogicky funguje i animace pro „zavírání opony“, kde se ale skrývá další problém, který je zapříčiněný tím, že uživatel může přejít rovnou z jednoho roku na druhý, aniž by se vrátil na časovou osu. Problém je, že v paměti je

uložen pouze snímek obrazovky pro rok, kterým se uživatel dostal na detail. Aby toto fungovalo, obrazovka s časovou osou je překryta detailovou obrazovkou[38]. Pokaždé, když uživatel přejde na další rok, nastaví se časová osa tak, aby byl daný rok přibližně ve středu[39] a vytvoří se nový snímek obrazovky.

Za zmínku stojí, že pokud by se místo komponenty `TwoWayView` použila komponenta `RecyclerView`, tak by zde byl další problém. Posun časové osy by se zřejmě z optimalizačních důvodů neprojevil, pokud by `Layout` s časovou osou nebyl viditelný.

Jak je vidět, i když se na začátku zdál tento způsob jednoduchý, ukázalo se zde několik problémů. Pokud by se daný problém měl řešit znovu, stálo by za to vyzkoušet animaci jednotlivých `itemů` komponenty `RecycleView`, tedy bez vytváření snímků obrazovky. Animace těchto `itemů` však není jednoduchá.

6.6 Stažení všech dat

Jak si uživatel prochází aplikaci, postupně se data stahují a ukládají do databáze nebo v případě obrázků do paměti zařízení.

V aplikaci je také možnost, jak stáhnout všechna data pro pozdější použití bez internetu. K tomu je použita služba, která se spustí a postupně stahuje data ze všech `endpointů` a ukládá je do databáze. Zároveň knihovna `Glide`, která slouží pro zobrazování obrázků z internetu, ukládá data do paměti zařízení pro offline použití.

6.7 Zobrazování obrázků

Některé obrázky musely být kvůli grafickému návrhu upraveny. Jedná se především o obrázky osobností, kdy profilová fotka je kruhová výseč a to v mnoha případech znamenalo, že osoba byla špatně vycentrovaná a měla oříznutou hlavu. Kvůli velkému počtu osobností byly prozatím upraveny pouze nejhorší případy.

6.8 Neaktuální API

Webové API bylo dokončeno dříve než mobilní aplikace. Později se ukázalo, že aplikace by mohla obsahovat další data. Dočasným řešením bylo některá data vložit přímo do aplikace. Konkrétně jde o obrázky děkanů a počty studentů u fakult.

[38] Toto není běžný způsob, běžně je obrazovka nahrazena jinou obrazovkou.

[39] Výjimkou jsou krajní roky, které zůstávají po stranách.

Kapitola 7

Testování

Testování je nedílnou součástí softwarového inženýrství a existuje mnoho druhů testů. Některé z nich mají za cíl odhalit chyby v `business logice` aplikace, jiné se snaží aplikaci shodit. V případě Androidu některé testy usnadňují testování různých verzí tohoto operačního systému, jiné pomáhají testovat aplikace na rozdílných zařízeních. Tyto testy jsou dobré pro vývoj, ale neodhalí, jak bude aplikace působit na uživatele.

7.1 Testování na Androidu

V tomto oddíle budou probrány testy platformy Android. Některé z nich jsou všeobecně známé techniky v softwarovém inženýrství, další jsou více specifické pro mobilní vývoj.

7.1.1 Unit testy

`Unit testy` neboli jednotkové testy jsou základní testy, které testují nějakou jednotku, nejčastěji třídu nebo metodu. Testovací API pro Android vychází z JUnit[40] a rozšiřuje ji o specifika pro Android, např. životní cyklus `Aktivity`. K tomu se používají tzv. `mockovací objekty` [93], které volají místo skutečných objektů jejich zjednodušené varianty upravené pro testování [94].

Výhodou těchto testů je, že jsou rychlé a není třeba kompilovat celou aplikaci, protože testují jen určitou část za použití `mockovacích objektů`. Z toho vyplývá, že ani není třeba spouštět celou aplikaci.

Tyto testy nebyly na tomto projektu použity, protože aplikace neobsahuje žádnou složitou `business logiku`, kterou by bylo potřeba testovat.

[40] JUnit je testovací framework pro Javu.

7.1.2 Monkey testy

Zajímavým testováním jsou tzv. `monkey testy`, které simulují chování uživatele. Přitom je možné pořizovat snímky obrazovky. Toto testování může probíhat dvojím způsobem.

První utilita `Monkey` [95] simuluje pseudonáhodné stisky kláves, gest, dotyků apod. Toto testování je vhodné pro stress testy, kdy se testování provádí s cílem shodit aplikaci.

Další utilita `Monkeyrunner` [96] dovoluje programově simulovat stisky kláves a tím vytvořit scénář, kterým uživatel prochází aplikaci. To je vhodné pro testování na více zařízeních a různých verzích API, zda se aplikace chová vždy stejně.

Ani jedna z těchto možností nebyla použita, protože se neočekával velký přínos pro projekt a z časových důvodů.

7.1.3 Emulátor vs reálné zařízení

Při komplexnějších testech je již třeba spustit celou aplikaci. Na to jsou dvě možnosti, buď lze použít `emulátor zařízení`[41] nebo testovat aplikaci na skutečném zařízení.

Součástí `Android SDK` je `emulátor zařízení` a pro základní otestování je dostačující. Výhodou tohoto emulátoru je, že lze určit některé parametry zařízení, mezi které patří verze API, rozlišení obrazovky, velikost RAM aj. Dále emulátor poskytuje možnosti simulace polohy, falešné SMS, lze prohlížet jinak nepřístupné soubory, např. databáze aplikace apod.

Nevýhodou emulátoru je, že nedokáže nahradit skutečná zařízení, která mají velmi podobné parametry, ale přesto se mohou chovat jinak, než by se dalo očekávat.

Při testování bylo upřednostňováno fyzické zařízení, konkrétně telefony Desire HD (Android 4.4), Honor 6 (Android 5.1) a tablet Nexus 7 (Android 5.0).

7.2 Testování webového API

Na začátku tohoto projektu nebylo webové API ještě hotové. Tento problém byl vyřešen použitím služby `Apiary`[98]. Tato služba nahrazuje skutečné webové API tím, že na domluveném rozhraní předává testovací data. Zároveň služba slouží jako dokumentace webového API.

[41] Virtuální mobilní zařízení, které běží na počítači [97].

7.3 Uživatelské testování

Testování aplikace uživatelem je velmi často používaný způsob, protože odhalí mnoho chyb, které při vývoji a používání aplikací vznikají. Také v tomto projektu testy odhalily nedostatky. Jak probíhalo testování aplikace Historie ČVUT, co bylo na jeho základě upraveno a výsledky testování budou následovat.

Na začátku bylo důležité určit, jak bude testování probíhat. Nejdříve bylo testujícímu položeno deset úvodních otázek. Po zaznamenání odpovědí měl testující možnost si aplikaci vyzkoušet. Na základě práce s testovanou aplikací odpověděl na doplňující otázky, případně proběhla diskuze. Nakonec byl za spolupráci odměněn sladkostí.

Testování se zúčastnilo čtrnáct testujících osob, z toho jedenáct testujících studovalo na ČVUT. Snahou bylo vybrat testující ze všech fakult, ale nepodařilo se najít testující z Fakulty dopravní a Fakulty jaderné a fyzikálně inženýrské.

Aplikace byla testována na uživatelích, kteří běžně používají některé Android zařízení. Kromě toho se testů zúčastnili uživatelé, kteří používají primárně jiný mobilní operační systém nebo nemají tolik zkušeností s dotykovými zařízeními.

7.3.1 Úvodní otázky

Otázky měly za cíl zjistit vztah testujícího k ČVUT, jeho zkušenosti s Androidem a znalosti o ČVUT. Při vytváření úvodních otázek bylo snahou, aby testující na ně uměl alespoň přibližně odpovědět a otázky byly jednoduché. Následuje seznam úvodních otázek:

1. Jsi z ČVUT, pokud ano, z jaké fakulty?
2. Kolik má ČVUT fakult a umíš je vyjmenovat?
3. Jak se jmenuje rektor ČVUT?
4. Pokud jsi z ČVUT, jak se jmenuje tvůj děkan?
5. Kam sahají počátky ČVUT?
6. Kolik myslíš, že bylo tou dobou studentů?
7. Kolik přibližně má ČVUT studentů?
8. Která fakulta ČVUT je největší?
9. Která fakulta ČVUT je nejmladší?
10. Jaké jsou tvé zkušenosti s Androidem na stupnici od 1 do 10 (čím větší číslo, tím větší zkušenosti)

Odpovědi na úvodní otázky nelze brát jako statistiky, které charakterizují studenty ČVUT. Testu se účastnil malý počet osob a výsledek je ovlivněn skladbou testujících, protože někteří měli výhodu v odpovědích na otázky vzhledem k fakultě, na které studují. Navíc výsledky byly ovlivněny testujícími, kteří nestudují na ČVUT. Průzkum nebyl důležitou součástí práce, ale posloužil jako zdroj informací pro použití aplikace i pro studenty ČVUT.

Třetina studentů uměla vyjmenovat všechny fakulty, dvě třetiny respondentů si nevzpomnělo na jednu až čtyři fakulty. Rektora ČVUT znalo 82 %, svého děkana znalo 91 % studentů. Jaká je nejmladší fakulta vědělo 79 % testujících. Odpověď na největší fakultu vědělo 64 % studentů.

Největším problémem byla otázka „Kam sahají počátky ČVUT?“, u které 45 % studentů vědělo správné století a správnou odpověď věděl jen jeden student. Odhady na počet studentů byly v některých případech velmi překvapivé, polovina studentů uvedla rozmezí 20 000 – 30 000 studentů (správná odpověď je 24 500), jeden testující odpověděl 200 000 studentů.

Tento malý průzkum ukázal, že by aplikace mohla mít smysl i pro studenty ČVUT.

7.3.2 Testování aplikace

Hlavní testování aplikace Historie ČVUT proběhlo na zapůjčeném tabletu Nexus 7 s verzí Androidu 5.0 Lollipop. Tablet byl předán ve vertikální poloze se spuštěnou aplikací tak, jako by ji uživatel poprvé spustil, tj. s obrazovkou Historie a otevřenou navigační lištou. Aplikace byla používána v režimu offline (viz 2.3.11).

Testující měli za úkol pracovat s aplikací a pokusit se nalézt odpovědi na předchozí otázky. Během testování autor této práce zaznamenával, jak uživatelé s aplikací pracují.

V případě, že si testující mysleli, že nepotřebují další informace nebo nevyužili všechny možnosti aplikace, došlo k nápovědě v užívání aplikace. Poté měli možnost si aplikaci dále vyzkoušet nebo přejít na další část testování.

7.3.3 Doplňující otázky

Poslední část měla za úkol formou doplňujících otázek a diskuzí získat další informace, které by pomohly vylepšit a zkvalitnit aplikaci.

1. Která kategorie tě nejvíc zaujala? (Historie, přehled, osobnosti)
2. Co tě na ní nejvíc zaujalo?
3. Která kategorie tě zaujala nejméně? (Historie, přehled, osobnosti)
4. Kvůli čemu tě zaujala nejméně?
5. Máš k aplikaci nějaké komentáře, nápady, připomínky?

Výsledkem doplňkových otázek je, že nejoblíbenější kategorií je kategorie Přehled. Nejčastějším důvodem byla přehlednost s dostatkem informací a zajímavé grafické zpracování.

Nejméně oblíbenou kategorií byla kategorie Historie, která většinu testujících neoslovila kvůli nezájmu o dějiny.

7.3.4 Výsledky testování aplikace

V tomto pododdíle budou popsány výstupy z testování aplikace. Nejdříve budou probrány obecné problémy chování aplikace, na které testující narazili. Další části budou věnovány jednotlivým obrazovkám aplikace, jak na ně uživatelé reagovali, jaké měli připomínky a jaké vznikly nápady na zlepšení.

7.3.4.1 Obecné problémy

Polovina testujících měla problém, že neobjevila gesto pro přechod mezi detailovými obrazovkami nebo ho objevila jen náhodou. Přechod mezi obrazovkami dělali tak, že klikli na detail, vrátili se zpět na předchozí obrazovku a klikli na další detail atd. Pokud náhodou toto gesto objevili nebo jim toto gesto bylo v závěru ukázáno, většinou ji u dalších kategorií dále využívali.

Výše uvedený problém je možné řešit několika způsoby. Jednou z možností je při prvním spuštění ukázat průvodce aplikace, ve kterém by bylo vidět, že jde použít určité gesto pro přechod mezi obrazovkami. Nevýhodou tohoto řešení je, že aplikaci si uživatel může prohlížet na jiném zařízení, než vlastním, např. v rámci různých akcí. Dalším z možných řešení je použít `Tabs` [64], kde by byly zobrazeny všechny roky a uživatel by se mezi nimi plynule posouval.

Další řešení, které bylo nakonec použito, je přidání šipek v detailové obrazovce, kterými se uživatel po kliknutí dostane na další detailovou obrazovku. Tato verze byla vyzkoušena na dvou testujících a ukázala se jako lepší.

Některé nejasnosti vyplývaly z neznalosti konkrétního zařízení nebo z různých verzí Androidu. Uživatel může být zvyklý, že jeho zařízení má tlačítko zpět hardwarové, zatímco testovaný tablet měl tlačítko zpět softwarové.

Další problémy, které se při testování vyskytly, nebyly tak časté a většinou vyplývaly z neznalosti Androidu. Mezi tyto problémy patřilo klikání na tlačítko titulku místo tlačítka zpět aj.

7.3.4.2 Navigační lišta

Ukázalo se, že nezkušení nebo méně zkušení uživatelé Androidu měli problém objevit `Navigační lištu` i přes to, že na začátku ji měli otevřenou, tedy ve stavu, v jakém by se aplikace spustila po nainstalování. Několikrát se stalo, že si prohlédli `kategorii Historie`, která se objeví po schování `Navigační lišty`, ale pak nevěděli, jak se dostat zpět do `Navigační lišty` nebo si neuvědomili, že zde nějaká byla.

`Navigační lišta` zůstala zachována, protože se jedná o obvyklý prvek a běžný uživatel Androidu s ním nemá problém. Kliknutí na tlačítko zpět bylo upraveno, aby se ukázala `Navigační lišta` místo toho, aby se aplikace zavřela. Další možností na zvážení je změna pořadí kategorií v `Navigační liště`, neboť `Kategorie Přehled` se jeví oblíbenější než `kategorie Historie`.

7.3.4.3 Kategorie Historie

V této kategorii uživatele zaujala časová osa, která je dělaná netradičně. S jejím ovládáním neměl nikdo problém. Některé testující zarazilo znázornění letopočtů, kde číslovka je z důvodů úspory místa na dvou řádcích, přesto později pochopili, co znamená.

I když tato kategorie nebyla mezi testujícími příliš oblíbená, své zájemce si našla. Použití obrázků a netradiční zpracování časové osy se ukázaly jako správná cesta.

7.3.4.4 Kategorie Přehled

Tato kategorie byla mezi testujícími nejoblíbenější z důvodu přehlednosti, velkého počtu obrázků a použitého `parralax efektu`.

Nadpisy „počet bakalantů“, „počet magistrantů“ a „počet doktorandů“, které jsou použity v přehledu fakult, způsobily u některých testujících údiv. Na základě kontextu většina testujících po nějaké době význam nadpisů pochopila. Jde o to, že „počet bakalantů“ je jinými slovy „počet studentů bakalářského programu“ a to je pro nadpis poněkud dlouhé slovní spojení. Jedním z řešení by bylo předělat design obrazovky fakult, aby delší nadpis tolik nevadil.

Prvek, který v aplikaci testujícím nejčastěji chyběl, byla mapa s informacemi, kde se která fakulta nachází. Nápad implementovat mapu do aplikace byl uvažován v souvislosti s virtuální prohlídkou (viz 2.4.8).

Další nápady v této kategorii, které by bylo vhodné realizovat, jsou přidání prorektorů, proděkanů, odkazy na stránky fakult nebo ČVUT a přidání dalších institucí[42].

Objevily se i nápady, které pro tuto aplikaci nejsou vhodné. Někteří testující by v aplikaci chtěli podrobné studijní plány. Tyto plány se často mění, jsou obsáhlé a zřejmě by bylo problematické je v aplikaci udržovat aktuální. Studijní plány a jiné rozsáhlé údaje nejsou vhodné pro tuto aplikaci, snazší by bylo se na tyto informace z aplikace odkazovat.

7.3.4.5 Kategorie Osobnosti

I tato kategorie si našla své zájemce a vzniklo mnoho nápadů na zlepšení a rozšíření.

[42] Mezi další instituce ČVUT patří Masarykův ústav vyšších studií aj. [16]

Řazení rektorů je od nejstaršího po současného. Pro rychlejší přechod mezi rektory byla použita dolní lišta. Někteří testující by uvítali rychlejší přesun na současného rektora. Jeden z nich navrhl, že by nad seznamem rektorů byl `Seekbar` [49], kterým by se dalo rychle přesunout na konec seznamu rektorů. Záměrem bylo, aby si uživatelé uvědomili, kolik rektorů ČVUT mělo. Zda-li něco takového realizovat, je tedy otázkou k zamyšlení. Někteří studenti v položce rektoři s překvapením zjistili, že jejich vyučující byl rektorem nebo je zaujalo, kolik mělo ČVUT rektorů.

U významných vědeckých osobností se ukázalo, že většinu zajímalo jen pár osobností, nejčastěji Křižík a Doppler, které znali. Proto by stálo za zvážení, zda nepřidat významné osobnosti ze současnosti, které by pro uživatele byli aktuálnější, i když z historického hlediska ne tak významné. Pokud uživatele napadlo použít tablet na šířku, zaujalo je současné zobrazení seznamu osob a detailu osoby.

Sportovci jsou kategorie, která některé testující zaujala a pro jiné nebyla tak zajímavá. Ukázalo se, že by bylo vhodnější uvést na seznamu sportovců, z jaké oblasti sportu jsou. Zároveň by testující uvítali sportovce ze současnosti, než ty historické.

Jeden testující poznamenal, že by uvítal známé osobnosti ze současné doby, které zde studovali, např. Petr Nárožný, David Vávra aj.

7.3.5 Chyby při testování

Testování přineslo mnoho důležitých poznatků, přesto se osoba provádějící testy dopustila několika chyb a je si vědoma, že některé věci lze příště zlepšit.

Během testování došlo u dvou testujících k pádu aplikace. Pokud by byl nasazen nástroj pro crash reporty, bylo by snazší dohledat, čím byly pády způsobeny.

Testování by bylo vhodné provádět na více zařízeních a v různých verzích Androidu, než bylo v tomto projektu realizováno.

Nebyla použita metoda, kdy se zkouší různé verze aplikace s cílem zjistit, která z nich je více uživatelsky přívětivá. Např. jedna verze aplikace by používala jiný ovládací prvek než druhá.

7.3.6 Závěry z testování

Většina testujících hodnotila aplikaci pozitivně, považovali ji za jednoduchou a přehlednou. Mnoho účastníků testování by uvítalo, pokud by se aplikace dále rozvíjela.

Část testujících zaujala časová osa, protože se jedná o netradiční ovládací prvek. Dalším zajímavým prvkem pro uživatele byl `parralax efekt`, který je použit u některých detailových obrazovek.

Někteří testující navrhli zlepšení aplikace a její rozšíření. Jedná se o přidání mapy s umístěním fakult, rozšíření informací o ČVUT a fakultách včetně osobností školy.

Na základě testování aplikace v ní byly provedeny některé přínosné změny. Mezi nejdůležitější patří přidání šipek pro přechod mezi detailovými obrazovkami. Další provedenou změnou aplikace je akce po kliknutí na tlačítko zpět, kdy místo toho, aby se aplikace zavřela, dojde k otevření navigační lišty. Dále bylo opraveno několik drobných chyb v aplikaci, překlepy, byla upravena některá data aj.

Uživatelské testování se ukázalo jako časově náročnější, ale mělo větší přínos pro projekt, než testy usnadňující vývoj.

Závěr

Hlavním cílem této diplomové práce bylo navrhnout a implementovat mobilní aplikaci pro operační systém Android, která bude prezentovat důležité momenty v historii ČVUT, významné vědecké osobnosti, vznik jednotlivých fakult a chronologii rektorů školy. Na základě těchto požadavků byla vytvořena mobilní aplikace Historie ČVUT a do aplikace byly přidány některé položky nad rámec zadání. Jedná se o stručné informace o ČVUT, stručné informace o jednotlivých fakultách a informace o sportovcích ČVUT.

Součástí práce byl průzkum podobných aplikací, konkrétně aplikací Timeline – Art Museum, World History, Boeing Milestones a Harvard Tour. Na sběru a přípravě dat pro aplikaci jsem spolupracoval s bakalářským studentem, který řešil verzi této aplikace pro operační systém iOS. Tohoto studenta jsem metodicky vedl a tvořili jsme spolu wireframy, na jejichž základě byla designérem vytvořena grafika aplikace.

Aplikace na platformě Android je nyní plně funkční a je propojena s webovým API, do kterého byla data nahrána a odkud aplikace získává data. Po stažení dat je možné aplikaci používat offline. Aplikace je lokalizována do českého a anglického jazyka.

Funkčnost aplikace byla ověřena na několika různých mobilních zařízeních a bylo provedeno čtrnáct uživatelských testů, které ověřily navržené uživatelské rozhraní. Většina testujících hodnotila aplikaci jako jednoduchou a přehlednou. Na základě testování aplikace došlo k přidání šipek jako další možnost pro přechod mezi detailovými obrazovkami. Současně byla provedena úprava ukončení aplikace. Během testování se dále ukázalo, že uživatelům by vyhovovalo přidání mapy s objekty ČVUT. S tím souvisí nápad použít v aplikaci virtuální prohlídku, který je v této práci popsán. Zadání diplomové práce bylo splněno nad rámec jeho požadavků.

Při práci na tomto projektu jsem se naučil používat architekturu Model-View-Presenter, vyzkoušel si reaktivní programování a dozvěděl se zajímavé informace o své škole.

Předpokládá se, že aplikace by mohla sloužit jako propagace ČVUT. Bylo

by však nutné dořešit problém s autorskými právy obrázků, protože prozatím byly použity pouze pro vzdělávací účely, ale při použití jako oficiální aplikace ČVUT by mohl nastat právní problém.

Původně se měla aplikace jmenovat Historie ČVUT, jak napovídá název této diplomové práce. Jelikož byly přidány i údaje ze současnosti, bylo by vhodné zvážit změnu názvu aplikace.

Aplikace bude představena na rektorátu ČVUT, kde se rozhodne, zda-li chce ČVUT aplikaci používat, případně ji dále rozvíjet.

Literatura

[1] Google International: *Activity*. [online]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/reference/android/app/Activity.html

[2] Google International: *Tablets and handsets*. [online]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/guide/practices/tablets-and-handsets.html

[3] Gary Sims: Google Play Store vs the Apple App Store: by the numbers (2015). [online], duben 2015, [cit. 10.11. 2015]. Dostupné z: http://www.androidauthority.com/google-play-store-vs-the-apple-app-store-601836/

[4] QS Quacquarelli Symonds: *Top Universities*. [cit. 10.11. 2015]. Dostupné z: http://www.topuniversities.com/universities/czech-technical-university-prague

[5] České vysoké učení technické v Praze: *ČVUToviny*. [online]. [cit. 10.11. 2015]. Dostupné z: https://www.youtube.com/playlist?list=PLAQ3p6oe_wT7rXfQJH1iW392n6kLDUBPQ

[6] Google International: *Android*. [online]. [cit. 10.11. 2015]. Dostupné z: https://www.android.com

[7] Apple Inc.: *Co je iOS*. [online]. [cit. 10.11. 2015]. Dostupné z: http://www.apple.com/cz/ios/what-is/

[8] Ackee s.r.o.: *Mobile apps and web development | Ackee*. [online]. [cit. 10.11. 2015]. Dostupné z: https://www.ackee.cz/

[9] Usability.gov: Wireframing. [online], [cit. 10.11. 2015]. Dostupné z: http://www.usability.gov/how-to-and-tools/methods/wireframing.html

[10] Usability.gov: User Experience Basics. [online], [cit. 10.11. 2015]. Dostupné z: http://www.usability.gov/what-and-why/user-experience.html

[11] Eriksson, U.: Functional vs Non Functional Requirements. [online], [cit. 10.11. 2015]. Dostupné z: http://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/

[12] Bartoš, V.: *Webový backend projektu Historie ČVUT*. Diplomová práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2015.

[13] Bartoš, V.: *HisČVUT API*. [online]. [cit. 10.11. 2015]. Dostupné z: http://docs.hiscvut.apiary.io/

[14] Facebook: *Facebook*. [online]. [cit. 10.11. 2015]. Dostupné z: https://www.facebook.com

[15] Tayerlová, M.: Insignie - rektorský řetěz a žezlo. [online], [cit. 10.11. 2015]. Dostupné z: https://www.cvut.cz/insignie-rektorsky-retez-a-zezlo

[16] Lindnerová, J.: Fakulty a pracoviště. [online], [cit. 10.11. 2015]. Dostupné z: https://www.cvut.cz/fakulty-a-pracoviste

[17] Hoffman, C.: Android is Based on Linux, But What Does That Mean? [online], [cit. 10.11. 2015]. Dostupné z: http://www.howtogeek.com/189036/android-is-based-on-linux-but-what-does-that-mean/

[18] Murphy, M. L.: *Android 2*. Brno: Computer Press, a.s., 2011, ISBN 9788025131947.

[19] Oracle Corporation: Learn About Java Technology. [online], [cit. 10.11. 2015]. Dostupné z: https://www.java.com/en/about/

[20] Oracle Corporation: *The Java® Virtual Machine Specification*. [online]. [cit. 10.11. 2015]. Dostupné z: https://docs.oracle.com/javase/specs/jvms/se7/html/

[21] Google International: *ART and Dalvik*. [online]. [cit. 10.11. 2015]. Dostupné z: https://source.android.com/devices/tech/dalvik/

[22] Google International: Fundamentals. [online], [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/guide/components/fundamentals.html

[23] Google International: *Fragment*. [online]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/reference/android/support/v4/app/Fragment.html

[24] Google International: *Service*. [online]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/guide/components/services.html

[25] Google International: *Content provider*. [online]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/guide/topics/providers/content-providers.html

[26] Google International: *Broadcast Receiver*. [online]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/reference/android/content/BroadcastReceiver.html

[27] W3Schools: *Introduction to XML*. [online]. [cit. 10.11. 2015]. Dostupné z: http://www.w3schools.com/xml/xml_whatis.asp

[28] Google International: What is api level. [online], [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/guide/topics/manifest/uses-sdk-element.html

[29] Google International: Dashboards. [online], [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/about/dashboards/index.html

[30] Google International: *Support library*. [online]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/intl/pt-br/tools/support-library/index.html

[31] Google International: *Overview of Google Play Services*. [online]. [cit. 10.11. 2015]. Dostupné z: https://developers.google.com/android/guides/overview

[32] Google International: *Android Studio*. [software]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/tools/studio/index.html

[33] Google International: *Android Plugin for Gradle*. [software]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/tools/building/plugin-for-gradle.html

[34] Google International: *Installing Android Studio*. [online]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/intl/es/sdk/installing/index.html?pkg=studio

[35] Evan Tatarka: *Gradle Retrolambda Plugin*. [software]. [cit. 10.11. 2015]. Dostupné z: https://github.com/evant/gradle-retrolambda

[36] Google International: *Build Variants*. [online]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/tools/building/plugin-for-gradle.html#buildVariants

[37] Google International: *Proguard*. [software]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/tools/help/proguard.html

[38] Google International: *Localizing with Resources*. [online]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/guide/topics/resources/localization.html

[39] Google International: *How Android Finds the Best-matching Resource*. [online]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/guide/topics/resources/localization.html

[40] Google International: *Android Manifest*. [online]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/guide/topics/manifest/manifest-intro.html

[41] Oracle Corporation: *JAR Manifest*. [online]. [cit. 10.11. 2015]. Dostupné z: http://docs.oracle.com/javase/7/docs/technotes/guides/jar/jar.html#JAR_Manifest

[42] Google International: *How Android Finds the Best-matching Resource*. [online]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/guide/topics/security/permissions.html

[43] Oracle Corporation: *JAR*. [online]. [cit. 10.11. 2015]. Dostupné z: http://docs.oracle.com/javase/7/docs/technotes/guides/jar/jar.html

[44] Google International: *Google Play*. [online]. [cit. 10.11. 2015]. Dostupné z: https://play.google.com

[45] Amazon: *Amazon underground*. [online]. [cit. 10.11. 2015]. Dostupné z: http://www.amazon.com/b?node=11350978011

[46] HockeyApp: *HockeyApp*. [online]. [cit. 10.11. 2015]. Dostupné z: http://hockeyapp.net

[47] Google International: *Gestures*. [online]. [cit. 10.11. 2015]. Dostupné z: https://www.google.com/design/spec/patterns/gestures.html

[48] Netsco: *Timeline - Art Museum*. [software]. [cit. 10.11. 2015]. Dostupné z: https://play.google.com/store/apps/details?id=kr.netsco.artist

[49] Google International: *Seekbar*. [online]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/reference/android/widget/SeekBar.html

[50] Solvapps: *World History*. [software]. [cit. 10.11. 2015]. Dostupné z: https://play.google.com/store/apps/details?id=solveraapps.chronicbrowser_wg_en

[51] The Boeing Company: *Boeing Milestones*. [software]. [cit. 10.11. 2015]. Dostupné z: https://itunes.apple.com/us/app/boeing-milestones/id510294916

[52] James Kowalski: *Boeing Milestones iPad App*. [software]. [cit. 10.11. 2015]. Dostupné z: https://www.behance.net/gallery/5069831/Boeing-Milestones-iPad-App

[53] YouVisit: *YouVisit*. [online]. [cit. 10.11. 2015]. Dostupné z: http://www.youvisit.com/

[54] YouVisit: *Harvard Tour*. [software]. [cit. 10.11. 2015]. Dostupné z: https://play.google.com/store/apps/details?id=com.yc360.college.harvard.harvard

[55] Tayerlová, M.: *Česká technika*. Praha: České vysoké učení technické v Praze, druhé vydání, 2004, ISBN 8001031659.

[56] Lomič, V.; Lomičová-Jirsáková, M.: *Vznik, vývoj a současnost Českého vysokého učení technického v Praze*. Praha: SNTL, 1982.

[57] České vysoké učení technické v Praze: Historie ČVUT. [online], [cit. 12.11. 2015]. Dostupné z: http://intranet.cvut.cz/cs/historie

[58] Dubnová, D.: Informace pro uchazeče o studium. [online], [cit. 10.11. 2015]. Dostupné z: https://www.cvut.cz/informace-pro-uchazece-o-studium

[59] České vysoké učení technické v Praze: Sedm statečných ČVUT. [online], [cit. 12.11. 2015]. Dostupné z: http://www.sedmstatecnych.cz/

[60] České vysoké učení technické v Praze: Osobnosti ČVUT. [online], [cit. 12.11. 2015]. Dostupné z: http://intranet.cvut.cz/fotobanka/osobnosti/osobnosti

[61] Bartoš, V.: Webové rozhraní Historie ČVUT. [online], [cit. 12.11. 2015]. Dostupné z: https://hiscvut.ackee.cz/login

[62] Usability.gov: Use Cases. [online], [cit. 10.11. 2015]. Dostupné z: http://www.usability.gov/how-to-and-tools/methods/use-cases.html

[63] Evolus: *Pencil project*. [software]. [cit. 10.11. 2015]. Dostupné z: http://pencil.evolus.vn

[64] Google International: *Tabs*. [online]. [cit. 10.11. 2015]. Dostupné z: https://www.google.com/design/spec/components/tabs.html

[65] Google International: *Navigation drawer*. [online]. [cit. 10.11. 2015]. Dostupné z: https://www.google.com/design/spec/patterns/navigation-drawer.html

[66] Google International: *Action Bar*. [online]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/design/patterns/actionbar.html

[67] Brown, J.: What Is Parallax Web Design? [online], [cit. 10.11. 2015]. Dostupné z: https://www.unleashed-technologies.com/blog/2013/08/15/what-parallax-web-design-%E2%80%93-definitions-tips-considerations

[68] IEEE Std: Defining architecture. [online], [cit. 10.11. 2015]. Dostupné z: http://www.iso-architecture.org/ieee-1471/defining-architecture.html

[69] Mikheev, K.: Introduction to Model View Presenter on Android. [online], [cit. 10.11. 2015]. Dostupné z: https://github.com/konmik/konmik.github.io/wiki/Introduction-to-Model-View-Presenter-on-Android

[70] Fowler, M.: Inversion of Control Containers and the Dependency Injection pattern. [online], [cit. 10.11. 2015]. Dostupné z: http://www.martinfowler.com/articles/injection.html

[71] Square Open Source: *Dagger*. [software]. [cit. 10.11. 2015]. Dostupné z: http://square.github.io/dagger

[72] Staltz, A.: The introduction to Reactive Programming you've been missing. [online], [cit. 10.11. 2015]. Dostupné z: https://gist.github.com/staltz/868e7e9bc2a7b8c1f754

[73] Lew, D.: RxJava collection. [online], [cit. 10.11. 2015]. Dostupné z: http://blog.danlew.net/tag/rxjava/

[74] ReactiveX: RxJava. [online], [cit. 10.11. 2015]. Dostupné z: https://github.com/ReactiveX/RxJava

[75] ReactiveX: *Alphabetical List of Observable Operators*. [online]. [cit. 10.11. 2015]. Dostupné z: https://github.com/ReactiveX/RxJava/wiki/Alphabetical-List-of-Observable-Operators

[76] Google International: *Threads*. [online]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/guide/components/processes-and-threads.html#Threads

[77] Johns, R.: Model-View-Presenter: An MVP pattern for IBM Ready Apps. [online], [cit. 10.11. 2015]. Dostupné z: https://developer.ibm.com/open/2015/10/22/model-view-presenter-mvp-for-ibm-ready-apps/

[78] Konstantin Mikheev: *Nucleus*. [software]. [cit. 10.11. 2015]. Dostupné z: https://github.com/konmik/nucleus

[79] Google International: *Building Apps with Over 65K Methods*. [online]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/intl/es/tools/building/multidex.html

[80] Jake Wharton: *Butter Knife*. [software]. [cit. 10.11. 2015]. Dostupné z: https://github.com/JakeWharton/butterknife

[81] Avast: *Butter Knife Zelezny*. [software]. [cit. 10.11. 2015]. Dostupné z: https://github.com/avast/android-butterknife-zelezny

[82] Square Open Source: *Retrofit*. [software]. [cit. 10.11. 2015]. Dostupné z: http://square.github.io/retrofit

[83] Ecma International: *JSON*. [online]. [cit. 10.11. 2015]. Dostupné z: http://www.json.org/

[84] Quinstreet Enterprise: *POJO*. [online]. [cit. 10.11. 2015]. Dostupné z: http://www.webopedia.com/TERM/P/POJO.html

[85] Bump Technologies: *Glide*. [software]. [cit. 10.11. 2015]. Dostupné z: https://github.com/bumptech/glide

[86] Square: *Sqlbrite*. [software]. [cit. 10.11. 2015]. Dostupné z: https://github.com/square/sqlbrite

[87] Hannes Dorfmann: *Sqlbrite DAO*. [software]. [cit. 10.11. 2015]. Dostupné z: https://github.com/sockeqwe/sqlbrite-dao

[88] Christopher Jenkins: *Calligraphy*. [software]. [cit. 10.11. 2015]. Dostupné z: https://github.com/chrisjenx/Calligraphy

[89] Adobe Systems: *PSD*. [online]. [cit. 10.11. 2015]. Dostupné z: http://www.adobe.com/devnet-apps/photoshop/fileformatashtml/#50577409_72092

[90] Google International: *Screen support*. [online]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/guide/practices/screens_support.html

[91] Google International: *Creating a Navigation Drawer*. [online]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/training/implementing-navigation/nav-drawer.html

[92] Zabłocki, K.: Pinch to reveal animation, like in Boeing Milestones. [online], [cit. 12.11. 2015]. Dostupné z: http://merowing.info/2012/07/pinch-to-reveal-animation-like-in-boeing-milestones/

[93] Chaffee, A.; Pietri, W.: Unit testing with mock objects. [online], [cit. 12.11. 2015]. Dostupné z: http://www.ibm.com/developerworks/library/j-mocktest/

[94] Google International: *Testing Fundamentals*. [online]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/tools/testing/testing_android.html

[95] Google International: *UI/Application Exerciser Monkey*. [software]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/tools/help/monkey.html

[96] Google International: *Monkeyrunner*. [software]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/tools/help/monkeyrunner_concepts.html

[97] Google International: *Android Emulator*. [software]. [cit. 10.11. 2015]. Dostupné z: http://developer.android.com/tools/help/emulator.html

[98] Apiary Inc.: *Apiary*. [online]. [cit. 10.11. 2015]. Dostupné z: https://apiary.io/

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

ART Android Runtime

DAO Data Access Object

DI Dependency Injection

DP Density Independent Pixel

DPI Dots per inch

ČVUT České vysoké učení technické

ČZU Česká zemědělská univerzita

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

JVM Java Virtual Machine

MVP Model–View–Presenter

NDK Native Development Kit

OS Operační systém

POJO Plain Old Java Object

PSD Photoshop Document

PX Pixel

RAM Random Access Memory

SDK Software Development Kit

SMS Short Message Service

UI User Interface

URL Uniform Resource Locator

UX User Experience

VM View-Model

VŠCHT Vysoká škola chemicko-technologická

XML Extensible Markup Language

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `app presentation`
  - `screenshots` ..... snímky obrazovek aplikace
  - `videos` ..... videa aplikace
- `exe` ..... adresář se spustitelnou formou implementace
- `sources` ..... zdroje dat
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `DP_Krabač_Tomáš_2016.pdf` ..... text práce ve formátu PDF
- `wireframes` ..... wireframy aplikace