UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2017 Tomáš Valko

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Programování task systému: Mobilní aplikace

Tomáš Valko

Bakalářská práce

2017

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2015/2016**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Tomáš Valko**
Osobní číslo:	**I13241**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Programování task systému: Mobilní aplikace**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem práce bude implementace komplexního Task systému. Systém bude mít komplexní rozměr a jeho tvorba bude směřována tak, aby celé řešení mělo potenciál ke komercializaci, nebo skutečnému praktickému využití. Vývoj celého systému bude řádně řízen dle příslušných ISO norem. Výstupem práce bude kromě vlastní aplikace také komplexní programátorská a uživatelská dokumentace. Z důvodu rozsahu zadaného tématu bude studentem vypracována jen dílčí část problematiky.

Řešenou dílčí problematikou bude tvorba mobilní aplikace, které bude schopná vizualizovat a manipulovat s daty Task systému. Student se ve své práci zaměření na problematiku programování pro mobilní zařízení; popíše využité prostředky a vývojové nástroje a problematiku datového spojení s databází a s prací v offline režimu.

Funkčnost vlastní aplikace student demonstruje v emulovaném i reálném prostředí. Implementace bude postačovat pro jednu mobilní platformu.

Rozsah grafických prací:

Rozsah pracovní zprávy: **35 - 45 stran**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

*** PECINOVSKÝ, Rudolf. OOP: Naučte se myslet a programovat objektově. Brno: Computer Press, a.s., 2010. ISBN 978-80-251-2126-9.**

*** KNUTH, D. E.: Umění programování - Základní algoritmy, Brno, Computer Press 2008, ISBN: 978-80-251-2025-5.**

*** WRÓBLEWSKI, Piotr. Algoritmy: datové struktury a programovací techniky. Vyd. 1. Překlad Marek Michalek, Bogdan Kiszka. Brno: Computer Press, 2004, 351 s. ISBN 80-251-0343-9.**

*** KEOGH, Jim; DAVIDSON, Ken. Datové struktury bez předchozích znalostí : průvodce pro samouky. Vyd 1. Brno : Computer Press, 2006. 223 s. ISBN 80-251-0689-6.**

Vedoucí bakalářské práce: **Ing. Josef Brožek**
Katedra informačních technologií

Datum zadání bakalářské práce: **31. října 2015**

Termín odevzdání bakalářské práce: **13. května 2016**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Mgr. Josef Horálek, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2016

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 10. 5. 2017 ……

Poděkování

Rád bych na tomto místě poděkoval své rodině a přátelům za podporu během studia. Dále bych rád poděkoval Petře Zimové za konzultace ohledně designu mobilní aplikace. Také bych rád poděkoval vedoucímu mé práce, panu Ing. Brožkovi, za trpělivost s mým ne vždy stoprocentním přístupem a jeho cenné rady, které mě obohatily nad rámec této práce.

Anotace

Práce popisuje problematiku programování pro mobilní zařízení s operačním systémem Android. V práci jsou postupně představeny jednotlivé etapy vývoje mobilní aplikace a uvedené postupy jsou demonstrovány při implementaci task systému. Dále jsou rozebrány architektonické vzory a možnosti jejich použití s důrazem na implementaci MVP. Součástí práce je také popis řešení komplikací při vývoji aplikace – konkrétně problémů vývojového prostředí, nedostatků Android Frameworku, komunikace se serverem zahrnující autentizaci a autorizaci, práce v offline režimu, vlastní implementace objektově relačního mapování a tvorba automatizovaných testů.

Klíčová slova

mobilní aplikace, Android, Java, softwarové architektury, offline režim, spojení s databází, JSON

Title

Task system implementation: Mobile application

Annotation

The bachelor’s thesis discusses development for mobile device with Android OS. It describes phases of creating a mobile application. The approach is demonstrated in task system implementation. This thesis also explores software architecture patterns with focus on implementation of MVP. Part of the thesis is a description of solutions of development complications – specifically IDE problems, disadvantages of Android Framework, a communication with server including authentication and authorization, offline mode working, own implementation of ORM and creation of automated tests.

Keywords

mobile application, Android, Java, software architecture, offline mode, database connection, JSON

Obsah

Seznam zkratek

ART	Android Runtime
BIOS	Basic Input-Output System
CPU	Central Processing Unit
DAO	Data Access Object
DI	Dependency Injection
GNU	GNU’s not UNIX
GPL	General Public License
(G)UI	(Graphical) User Interface
HW	Hardware
ICT	Information and Communication Technologies
IDE	Integrated Development Environment
JDK	Java Development Kit
JRE	Java Runtime Environment
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MAC	Media Access Control
MVC	Model View Controller
MVP	Model View Presenter
MVVM	Model View ViewModel
OOP	Object-Oriented Programming
OOPL	Object-Oriented Programming Language
ORM	Object-Relational Mapping
OS	Operation System
PC	Personal Computer
POJO	Plain Old Java Object
RDB	Relational Database
SDK	Software Development Kit
SQL	Structured Query Language
VM	View Model

Seznam obrázků

Úvod

Vývoj mobilních aplikací patří mezi nejmladší obory ICT. O velikosti tohoto oboru svědčí, že v červnu 2016 bylo v největším distribučním kanále s aplikacemi Google Play Store dostupných více než 2 200 000 aplikací. [1] Skutečný rozvoj započal poté, co byly představeny první chytré telefony s dotykovým displejem. To proběhlo v roce 2007, kdy Apple představil svůj první mobilní telefon iPhone, a o rok později, kdy byly představeny i první telefony s OS Android.

Dnešní moderní mobilní telefon již neslouží zdaleka pouze pro telefonování. Chytrý telefon kombinuje funkce mnoha zařízení a umožňuje jejich mobilní využití. Pro představu těchto funkcí lze uvést např. přehrávání hudby, pořizování fotografií, nahrávání videa, hraní her nebo GPS navigaci. Většina chytrých telefonů využívá přístupu k internetu a umožňuje instalovat aplikace dostupné na distribučním kanále dané mobilní platformy.

Tato bakalářská práce se zabývá vývojem mobilní aplikace pro platformu Android. Cílem této práce je představit operační systém Android, popsat možnosti při tvorbě mobilní aplikace pro tuto platformu a jednotlivé etapy jejího vývoje při využití programovacího jazyka Java a oficiálního vývojového prostředí této platformy, Android Studia. Dále je cílem rozebrat návrh architektury aplikace a prozkoumat řešení problémů spojené s jejím vývojem.

Práce je součástí týmového projektu, jehož cílem je vytvoření komplexního Task systému s potenciálem ke komercializaci nebo skutečnému praktickému využití. Mobilní aplikace je implementovaná tak, aby spolupracovala s webovou aplikací, se kterou komunikuje prostřednictvím technologie JSON a získává data nutná k fungování Task systému. Tato data jsou následně ukládána do interní databáze v mobilním telefonu, čímž je umožněno používání aplikace i v době, kdy nemůže komunikovat s rozhraním webové aplikace (např. z důvodu nedostupnosti internetového připojení či výpadku webového serveru).

Struktura práce

V první kapitole jsou stručně vysvětleny základní pojmy, které se v této práci vyskytují, jejichž znalost je důležitá pro správné pochopení dalších částí této práce.

Ve druhé kapitole jsou popsány principy fungování operačního systému Android a jednotlivé vrstvy jeho architektury. Na konci této kapitoly je čtenář seznámen s jednotlivými částmi samotné Android aplikace, tvorbou těchto částí, jejich účelem a fungováním.

Třetí kapitola nejprve nastiňuje možnosti při vývoji pro Android. Dále se zabývá vývojovými nástroji používanými při vývoji s využitím programovacího jazyku Java a jejich následným použitím.

Čtvrtá kapitola rozebírá možnosti při návrhu Android aplikace, popisuje výhody a nevýhody jednotlivých architektur a seznamuje s implementací MVP architektury.

V páté kapitole je prezentována aplikační vrstva praktické části.

Šestá kapitola pojednává o problémech při vývoji aplikace pro Android. Jsou zde postupně nastíněny nedostatky vývojového prostředí, dále problémy Android Frameworku spolu s možnostmi řešení prostřednictvím externích knihoven, a nakonec funkční problémy, možnosti jejich řešení a představení řešení použitých v praktické části této práce.

Typografické konvence

V práci jsou dodržovány obvyklé typografické konvence. Kromě nich jsou ještě kurzívou vyznačeny názvy tříd a objektů, anotace, texty na prvcích GUI z praktické části práce a názvy souborů a složek v obsahu volně loženého CD (zde je navíc pro větší přehlednost zmenšeno písmo na velikost 11 bodů). Z důvodu úspory místa jsou některé obrázky umístěny vedle sebe v rámci jednoho obrázku a označeny a) a b). Ukázky zdrojového kódu jsou uváděny v rámečku ve fontu Courier New velikosti 10 bodů s barvou a tučností písma dle syntaxe daného programovacího jazyku.

1 Základní pojmy

V této kapitole je vysvětleno několik základních pojmů, které jsou používány v dalších kapitolách této práce.

Android

Základ mobilní platformy Android tvoří OS pro chytré telefony, který od roku 2005 vlastní společnost Google a vyvíjí konsorcium Open Handset Alliance tvořené mnoha technologickými a mobilními společnostmi. Systém je postavený na linuxovém jádře a je navržený tak, aby mohl běžet na různém HW. Další nedílné součásti této platformy jsou

- distribuční kanál, který se v současnosti nazývá Google Play Store,
- výrobci mobilního HW s OS Android,
- vývojáři vytvářející mobilní aplikace pro OS Android
- a hlavně uživatelé mobilních zařízení této platformy. [2]

ART

ART je nový virtuální stroj, který od roku 2013 postupně nahradil Dalvik v nových verzích Androidu. Chová se jako Dalvik, ale bajtkód neinterpretuje. Místo toho ho už při instalace přeloží na strojový kód, který poté pouze spouští při každém spuštění aplikace. To má za následek rychlejší běh aplikací a delší výdrž baterie. Nevýhodami jsou naopak větší velikost aplikací (o 10-20%) a delší instalace. [3]

Autentizace

Testování autentizace je obšírně vysvětleno v [4]. *„Pod pojmem autentizace rozumíme ověření totožnosti uživatele dat, tzn. ověření toho, že je uživatel skutečně tím, za kterého se vydává.*

Metod autentizace je mnoho, můžeme je však rozdělit do tří základních skupin:

- *autentizace založená na určité znalosti (např. hesla),*
- *autentizace biometrická, jenž spočívá v tom, že každý jedinec má určité biologické znaky jedinečné (např. snímání sítnice oka)*
- *a autentizace prostřednictvím autentizačního předmětu, kdy se identita prokáže vlastnictvím jedinečného předmětu (např. platební karty).“*

Autorizace

„Autorizace je proces, který navazuje na autentizaci a můžeme ho chápat jako přiřazení určitých práv (co uživatel může a co ne) pro práci a využívání příslušného systému.“ [5]

Vlastní autorizace pak může probíhat např. metodou vyhledání v seznamu oprávněných subjektů, jejich rolí nebo práv.

Backporting

Backporting je proces, při které jsou funkce z nové verze softwaru přenášeny na starší verze stejného softwaru. Tento proces je nejčastěji používán při opravě bezpečnostních chyb ve starých verzích softwaru. [6]

Bajtkód

Bajtkód je mezikód, do kterého je kompilovaný Java kód, který je následně interpretován JVM, jak uvádí také literatura: *„Při spuštění je pak buď interpretovaný prostřednictvím JVM, nebo (za účelem dosažení vyššího výkonu) kompilovaný tzv. JIT (Just-in-time) kompilátorem, tedy za běhu programu do nativního kódu. Z principu nemůže dosahovat rychlosti čistě nativního kódu.“* [7]

BIOS

„BIOS je základní programem PC, který řídí komunikaci s HW na nejnižší úrovni.“ Ihned po startu PC se používá pro oživení a konfiguraci připojených HW zařízení. [8]

Dalvik

Dalvik byl virtuální stroj vytvořený speciálně pro platformu Android. Při vývoji převáděl bajtkód JVM na bajtkód pro Dalvik (soubory s koncovkou .dex). Tyto soubory byly daleko menší než soubory .class a do JAR souborů se pouze ukládaly (nebylo potřeba je komprimovat). Načítání pak bylo daleko rychlejší. [9]

Kromě převodu kódu sloužil Dalvik i jako samotné běhové prostředí pro mobilní aplikace v zařízeních s OS Android. Při spuštění aplikace načetl .dex soubory a ty poté interpretoval (prováděl) přímo v jejich zdrojovém kódu (nepřekládal je tedy do strojového kódu). Na rozdíl od JVM byl optimalizován pro mobilní zařízení, která mají obvykle menší paměť a méně výkonná CPU. [10]

Dependency Injection

Dependency Injection (dále DI) je návrhový vzor pro vkládání závislostí. *„Podstatou DI je odebrat třídám zodpovědnost za získávání objektů, které potřebují ke své činnosti (tzv. služeb) a místo toho jim služby předávat při vytváření.“* [11] To znamená, že třída se nemusí starat o to, odkud má závislosti získat a počká, až jí je předá někdo jiný.

Groovy

„Groovy je dynamicky typovaný skriptovací jazyk, který se syntaxí nejvíce podobá jazyku Java. Je kompilován do byte kódu Javy a s tím souvisí i jeho schopnost využívat knihovny napsané v Javě, naopak v Javě lze využívat knihovny napsané v Groovy. Toto propojení dává jazyku Groovy příležitost se ihned prosadit všude, kde je doposud používaná jen Java.“ [12]

HW virtualizace

HW virtualizace umožňuje na jednom fyzickém počítači spustit více operačních systémů pomocí několika virtuálních strojů. Díky virtualizaci je také možné testovat a ladit programy bez nutnosti instalace další testovací verze operačního systému. [13]

Výrobci procesorů označují HW virtualizaci různými jmény, jde však o podobné technické řešení. Technologie Intelu se nazývá Intel VT-x a technologie AMD se zase označuje jako AMD-V. [14]

I když procesor HW virtualizaci podporuje, je často ve výchozím stavu vypnutá a je třeba ji povolit v programu SETUP systému BIOS.

Chytrý telefon

Mobilní telefon obsahující speciální OS kombinující výhody počítačového OS se specifickými funkcemi užitečnými pro mobilní zařízení. Hlavními znaky moderního chytrého telefonu je ovládání přes dotykový displej a možnost instalovat nejrůznější mobilní aplikace určené pro danou mobilní platformu, které nemusí souviset s telefonováním. [15]

Instalace SW

„Instalace SW je takový proces, při kterém je SW upraven do stavu, ve kterém může být spuštěn se všemi funkcemi, které SW poskytuje.“ [16]

Obvykle je použit komprimovaný soubor, který je při instalaci dekomprimován, soubory z něj jsou umístěny na svá místa a provedou se potřebná nastavení (např.: nastavení firewallu a úprava registrů).

Java Development Kit

Java Development Kit (dále JDK) je soubor nástrojů pro vývoj aplikací v programovacím jazyce Java. JDK obsahuje také běhové prostředí JRE pro programy napsané v Javě, jehož součásti jsou JVM (virtuální stroj) a sada knihoven (místo nich Android používá knihovny od Googlu). [17]

JSON

JSON je jednoduchý textový formát, který je nezávislý na použité platformě. Kromě toho vyznačuje také tím, že je pro člověka snadno čitelný a využívá konvence, které jsou dobře známé programátorům jazyků rodiny C (např. C++, Java a PHP). [18]

„JSON je založen na dvou strukturách:

- *Kolekce párů název/hodnota. Ta bývá v rozličných jazycích realizována jako objekt, záznam (record), struktura (struct), slovník (dictionary), hash tabulka, klíčový seznam (keyed list) nebo asociativní pole.*
- *Seřazený seznam hodnot. Ten je ve většině jazyků realizován jako pole, vektor, seznam (list) nebo posloupnost (sequence).“* [18]

Java Virtual Machine

Java Virtual Machine (dále JVM) je virtuální stroj, který převádí zdrojové kódy programů napsaných v programovacím jazyce Java na bajtkód (soubory s koncovkou .class). Na rozdíl od běžných Java aplikací Android nevyužívá JVM pro běh samotných aplikací a tento virtuální stroj je tedy vyžíván pouze při vývoji. [19]

Linuxové jádro

Jádro operačního systému vytvořil finský student Linus Torvalds, který v roce 1991 uvolnil jeho zdrojové kódy pod licencí zakazující komerční distribuci. Následně se k vývoji začali přidávat programátoři z celého světa.

Dnes linuxové jádro tvoří několik milionů řádků zdrojového kódu a nové verze jsou uvolňovány pod licencí GNU GPL, což je licence pro svobodný software, která vyžaduje, aby byla odvozená díla dostupná pod toutéž licencí. [20]

MAC adresa

„MAC adresa je 48 bitů dlouhý unikátní identifikátor, kterým je označeno každé síťové zařízení, někdy také označována jako fyzická adresa nebo Ethernet adresa. Je zadávána výrobcem a měla by být hardwarově neměnná, neboť je vypálena na čipu paměti ROM (Read Only Memory), ovšem při spuštění zařízení je nahrávána do paměti RAM (Random Access Memory), kde je možné ji změnit softwarově.

MAC adresa je zapisována 12 hexadecimálními číslicemi a je možné ji zapsat třemi možnými způsoby např. 1:23:45:67:89:AB, 01-23-45-67-89-AB nebo 0123.4567.89AB. Uspořádání MAC adresy je dáno standardizací IEEE. Počátečních 24 bitů, to jest prvních 6

hexadecimálních čísel, slouží pro označení výrobce podle OUI (Organizationally Unique Identifier) a zbylých 6 je unikátní označení daného zařízení.“ [21]

Mobilní aplikace

Mobilní aplikace je SW program určený speciálně pro chytré telefony, případně tablety, využívající funkce mobilního OS, pro který je určen, a možností jeho HW. Do chytrého telefonu bývá zpravidla stahována přes distribuční kanál dané mobilní platformy. [22]

Mobilní platforma

Mobilní platforma označuje soubor zásad, na kterých je postaven chytrý telefon. Každá mobilní platforma má většinou vlastní operační systém, vývojové nástroje, aplikační rozhrání, HW požadavky a jejím vlastníkem bývá zpravidla velká ICT korporace. [23]

Nejznámějšími mobilními platformami jsou Apple iOS a Google Android.

Open Source

Open Source je licence, pod kterou je vyvíjen software, který obdrží certifikaci od společnosti Open Source Initiative. Tento software musí být volně šiřitelný a poskytovat své zdrojové kódy, které je možné prohlížet, upravovat a distribuovat dále pod stejnou licencí. [24]

Objektově relační mapování

Relační databáze (dále RDB) a objektově orientované programovací jazyky (dále OOPL) jsou založené na odlišných paradigmatech obsahujících technické a koncepční neslučitelnosti. Objektově relační mapování (dále ORM) zahrnuje řešení pro automatizované mapování objektů z aplikační vrstvy do relačních dat. [25]

Díky použití ORM může být vývojář, který v aplikaci pracuje s daty reprezentovanými objekty, odstíněn od nutnosti pracovat s konkrétními SQL dotazy. ORM tedy v aplikaci zjednodušuje provádění běžných databázových operací jako je čtení, zápis, úprava a mazání dat.

POJO

POJO je třída, která se vyznačuje jednoduchostí a není omezena žádnými speciálními požadavky. Takováto třída by neměla implementovat předem specifikované rozhraní a dědit od předem specifikované třídy. Obvykle obsahuje bezparametrický konstruktor a metody pro nastavení a čtení jejích vlastností. [26]

Programovací jazyk

Programovací jazyk je prostředek k zápisu algoritmů počítačového programu, jež jsou přeložitelné do podoby spustitelné počítačem. [27]

Relační databáze

„Relační databáze je sada nástrojů pro efektivní a spolehlivé ukládání dat a pro manipulaci s nimi, data jsou uchovávaná v tabulkách, kde řádek odpovídá záznamu a sloupec atributům určitého datového typu. Vazby mezi tabulkami se nazývají relace a ty v podstatě popisují vztahy mezi objekty v reálném světě.“ [28]

Režim offline

Režim offline označuje stav, kdy není zařízení, se kterým uživatel pracuje, připojené k internetu. [29]

Když tedy uživatel pracuje na zařízení, které se zrovna nachází v tomto stavu, aplikace nemůže stáhnout ze serveru aktuální data, ty se tak stáhnou až ve chvíli, kdy se zařízení může znova připojit na server (dostane se do režimu online). V tu chvíli se také nahrají na server změny, které uživatel v aplikaci provedl během doby, kdy bylo zařízení v režimu offline.

Singleton

Singleton je návrhový vzor, který zaručuje, že daná třída bude mít právě jednu instanci, která bude globálně přístupná. Tento vzor by měl být využíván pouze v případech, kdy je jisté, že daná třída nebude nikdy v budoucnu potřebovat více instancí. Pro umožnění přístupu k instanci pro více tříd by měla být využívána jiná řešení. [30]

SQL

SQL je standardizovaný počítačový jazyk navržený pro získávání informací z databází, který se postupně stal nejpopulárnějším dotazovacím jazykem vůbec. [31]

SQLite

SQLite je relační databázový systém, který je soběstačný, funguje bez serveru a s nulovou konfigurací. Kód pro SQLite je zdarma veřejně dostupný pro jakékoliv použití (komerční nebo osobní). [32]

Android nabízí plnou podporu pro SQLite databáze. Všechny databáze v ní vytvořené budou dostupné podle jména v jakékoliv části aplikace ale ne mimo ni. [33]

Špagetový kód

Špagetový kód je způsob programování, který vede k nesrozumitelnému a neudržovatelnému kódu. Špagetový kód může každý vývojář vnímat jinak podle toho, na jakém levelu jsou jeho programovací schopnosti. [34]

Vývojové prostředí

Vývojové prostředí (anglická zkratka IDE) je sada nástrojů, které pomáhají vývojářům např. s laděním programů, hledáním chyb, udržováním přehledu ve zdrojovém kódu nebo nápovědou k funkcím. Jeho hlavní součástí je však integrovaný překladač, který dokáže přeložit zdrojový kód do kódu spustitelného na daném zařízení. [35]

Podle statistik vyhledávače Google jsou nejpopulárnějšími vývojovými prostředími Microsoft Visual Studio (používané hlavně pro programování v jazycích C/C++ a C#) a Eclipse (slouží hlavně pro programování v jazyce Java). [36] Pro programování pro mobilní platformu Android se používá hlavně Android Studio a rozšířené Eclipse. [37]

Testování softwaru

Testování softwaru je provádění programu s úmyslem nalézt chyby. V každém softwaru mohou být různé typy chyb jako jsou např. chybná specifikace, chybný návrh, chybný příkaz, chybný vstup nebo chybný test. Aby mohly být všechny možné chyby nalezeny, existuje několik fází testování:

- Jednotkové testování – testování jednotek u OOP znamená testování jednotlivých tříd a metod. *„Testy jednotek se velmi špatně aplikují na již zaběhlých projektech. Proto je vhodné zabývat se těmito testy již v etapě návrhu aplikace a v té době se rozhodnout, zda tyto testy budeme využívat.“* [39] Testy by měly izolovány (aby se vzájemně neovlivňovaly) a fungovat za každých okolností (např. při spuštění v jakémkoli pořadí nebo bez přístupu k internetu).
- Integrační testování – ověření bezchybné komunikace mezi jednotlivými komponentami uvnitř aplikace. Tyto testy lze u menších projektů vynechat, pokud jsou provedeny následující fáze testování.
- Systémové testování – ověření aplikace jako funkčního celku. *„Podle připravených scénářů se simulují různé kroky, které v praxi mohou nastat. Obvykle probíhají v několika kolech. Nalezené chyby jsou opraveny a v dalších kolech jsou tyto opravy opět otestovány.“* [39] Tato úroveň testování je poslední před předáním softwaru zákazníkovi.
- Akceptační testování – testování probíhající na straně zákazníka. *„Testy jsou prováděny podle připravených scénářů a nalezené nesrovnalosti mezi aplikací a specifikací jsou reportovány zpět vývojovému týmu. Opravené chyby jsou nasazeny na prostředí u zákazníka.“* [39]

Zároveň platí, že v čím dřívější fázi jsou chyby nalezeny, tím méně prostředků stojí jejich oprava. Testování může být jak manuální, tak automatizované. Čerpáno z [38, 39].

Zdrojový kód

Zdrojovým kódem je popis programu pomocí programovacího jazyka, který lze přeložit do strojového kódu počítače a spustit. [40]

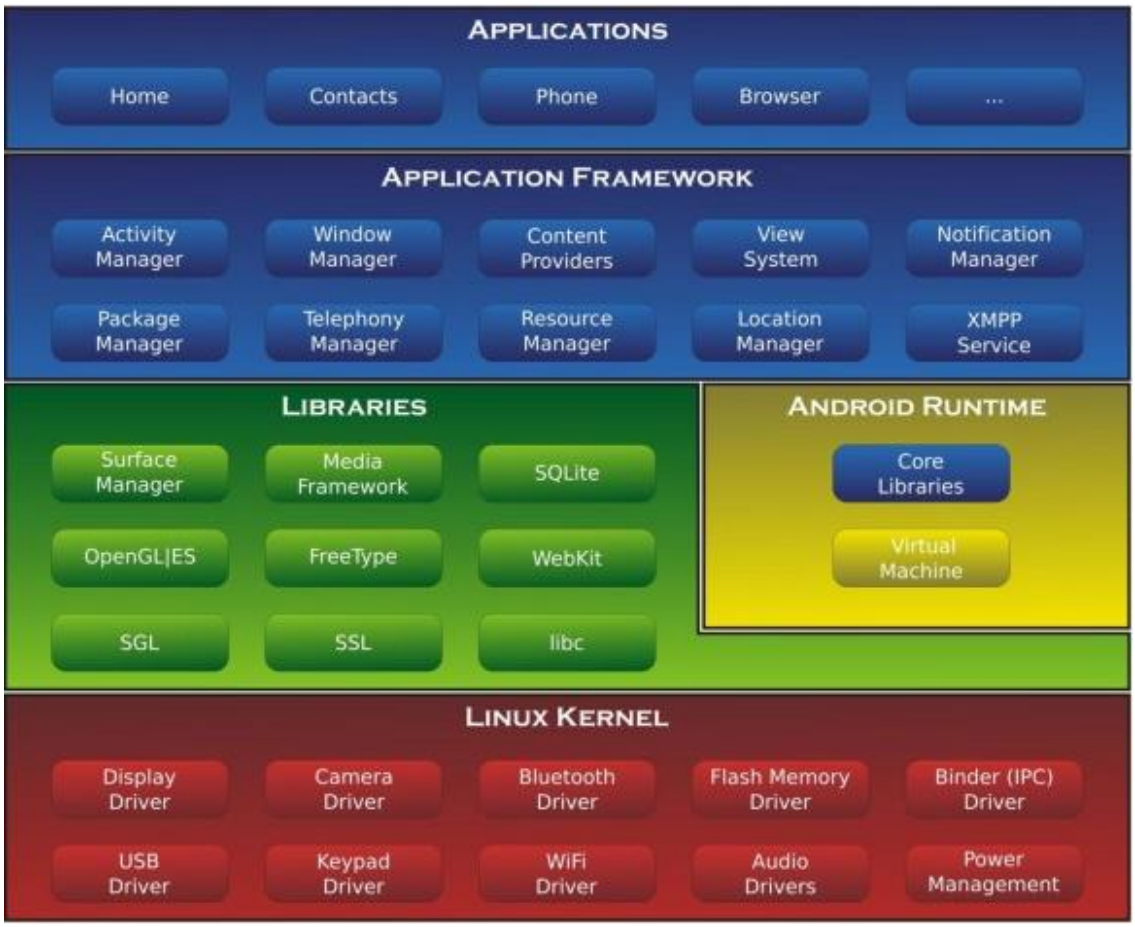
2 Android

„Android je rozsáhlý operační systém vytvořený společností Google, založený na open source platformě.“ Představen byl v roce 2007 sdružením firem „Open Handset Alliance“, jehož členy byly společnosti jako Google, Intel, HTC, Samsung a další. [10, s. 15-16]

Cílem této kapitoly je poskytnout čtenáři širší přehled o tom, jak funguje operační systém Android a jak probíhá programování mobilní aplikace určené pro tento systém, což umožní snáze pochopit další kapitoly této práce.

2.1 Architektura OS

Architekturu operačního systému Android tvoří pět vrstev, které spolu spolupracují (zobrazeno na obrázku 1). Poslední vrstvou jsou samotné aplikace.



Obrázek 1: Architektura operačního systému Android[1]

[1] Zdroj: [41]

Linuxové jádro

„Nejnižší vrstvu architektury představuje linuxové jádro, jehož základní funkcí je implementace abstrakce mezi použitým hardwarem a softwarem ve vyšších vrstvách. Při startu zařízení je jádro zavedeno do operační paměti a poté je mu předáno řízení. To představuje neustálou kontrolu nad systémem, koordinaci činnosti všech běžících procesů, podpora správy paměti, správa sítě atd.“ [10, s. 19]

Knihovny

Další vrstvu tvoří Android API a pokročilé C/C++ knihovny.

Knihovny patřící do Android API jsou napsané v Javě a jsou určeny pro programování Android aplikací. Usnadňují vývojářům např. kreslení grafiky a přístup k databázi.

Tyto knihovny ale mnoho práce nezastanou a spíše to jsou takové obálky kolem pokročilých C/C++ knihoven. Tyto knihovny zastanou většinu výpočetně složitějších úkolů, jako je vykreslování grafiky a písma, správu relační databáze nebo SSL šifrování. [42, s. 86]

ART

Vrstva ART má podobně jako ta předchozí 2 části. Obsahuje virtuální stroj (dříve Dalvik Virtual Machine, který je postupně nahrazován ART) a základní Java knihovny. Tento virtuální stroj na rozdíl od JVM je optimalizován pro mobilní zařízení, která mají většinou menší operační paměť a výkon.

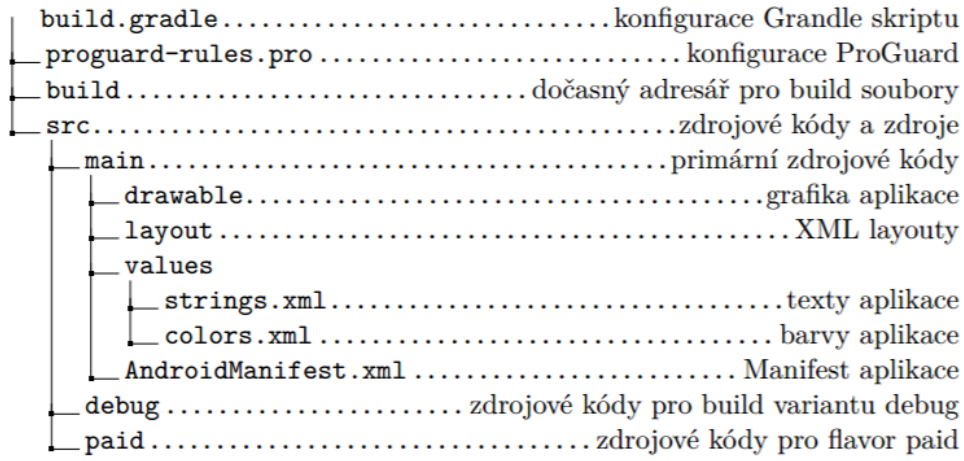
Obsah knihoven lze srovnat s platformou Java SE, ale na rozdíl od knihoven této platformy neobsahují knihovny pro vytváření GUI desktopové aplikace (AWT, Swing), které byly nahrazeny knihovnami pro tvorbu GUI pro Android. Navíc jsou přidány knihovny Apache pro práci se sítí. [10, s. 21]

Application Framework

Jedná se o nejdůležitější vrstvu, která umožňuje tvorbu bohatých a inovativních aplikací. Prostřednictvím této vrstvy může vývojář přistupovat k nejrůznějším službám a využívat je ve svých aplikacích. Tyto služby mohou zpřístupňovat data v jiných aplikacích, prvky uživatelského rozhraní, hardware používaného zařízení a řadu dalších funkcí. [10, s. 21]

2.2 Struktura projektu

Strukturu Android projektu je možné vidět na obrázku 2.



Obrázek 2: Struktura Android projektu[2]

Android Manifest

Základem jakékoliv aplikace pro Android je XML soubor AndroidManifest.xml, který je uložen v kořenovém adresáři a specifikuje jednotlivé parametry aplikace. Při vytvoření aplikace je automaticky vygenerován výchozí manifest, ale ten je třeba s rozvíjející se aplikací upravovat. Obsahuje např.

- jméno Java balíčku aplikace (reprezentuje unikátní identifikátor pro aplikaci),
- deklaraci systémových oprávnění aplikace (např. přístup k internetu a čtení kontaktů),
- základní atributy aplikace (např. název potomka třídy Application implementované pro aplikaci, téma aplikace a ikona aplikace)
- a jednotlivé komponenty (např. aktivity, služby a poskytovatelé obsahu). [10, s. 45]

Grafické objekty

Ikony a další grafické objekty jsou ukládány do adresáře drawable. Tento adresář tedy obsahuje buď vektorové XML soubory nebo soubory obrázkových formátů (JPG, PNG, …) v různých rozlišeních.

V implementační části této práce byly použity 2 druhy ikon. První druh byly klasické Android ikony importované prostřednictvím Android Drawable Importer pluginu. Další skupina ikon byla stažena z databáze volně dostupných vektorových ikon http://www.flaticon.com ve formátu SVG. Z tohoto formátu byly ikony převedeny do XML formátu prostřednictvím online konvertoru http://inloop.github.io/svg2android.

[2] Zdroj: [43]

2.3 Activity

Activity (dále aktivita) je hlavní část programu, bez které by aplikace nešla spustit. Každá aktivita je potomkem třídy Activity a reprezentuje jedno okno aplikace.

„Okno typicky vyplňuje celou obrazovku, nebo představuje plovoucí dialog. Aplikace se obvykle skládá z několika aktivit, které jsou volně vázány mezi sebou. Typicky je v aplikaci jedna aktivita určená jako hlavní (provádí hlavní funkcionalitu) a je zobrazena uživateli ihned po spuštění. Každá aktivita může spouštět další aktivity s cílem provádět různé další funkce.“ [10, s. 39]

Aktivity v systému jsou řízeny jako zásobník aktivit. Když je vytvořena nová aktivita, automaticky je umístěna na vrchol tohoto zásobníku a stává se běžící. Předchozí aktivita zůstává v zásobníku vždy níže a nepřejde znovu na popředí, dokud nová aktivita existuje. [44]

Běžící (Running)

Aktivita je na popředí obrazovky (na vrcholu zásobníku) a dostává informace o uživatelském vstupu. [43]

Pozastavená (Paused)

V tomto stavu se aktivita nachází, pokud je částečně překryta jinou aktivitou (nikoli svým vlastním dialogem) nebo je nad ní jiná aktivita, která je však průhledná. Aktivita si stále pamatuje svůj stav před pozastavením, ale může být zničena systémem v situacích, kdy je volné extrémně nízké množství pamětí. [44]

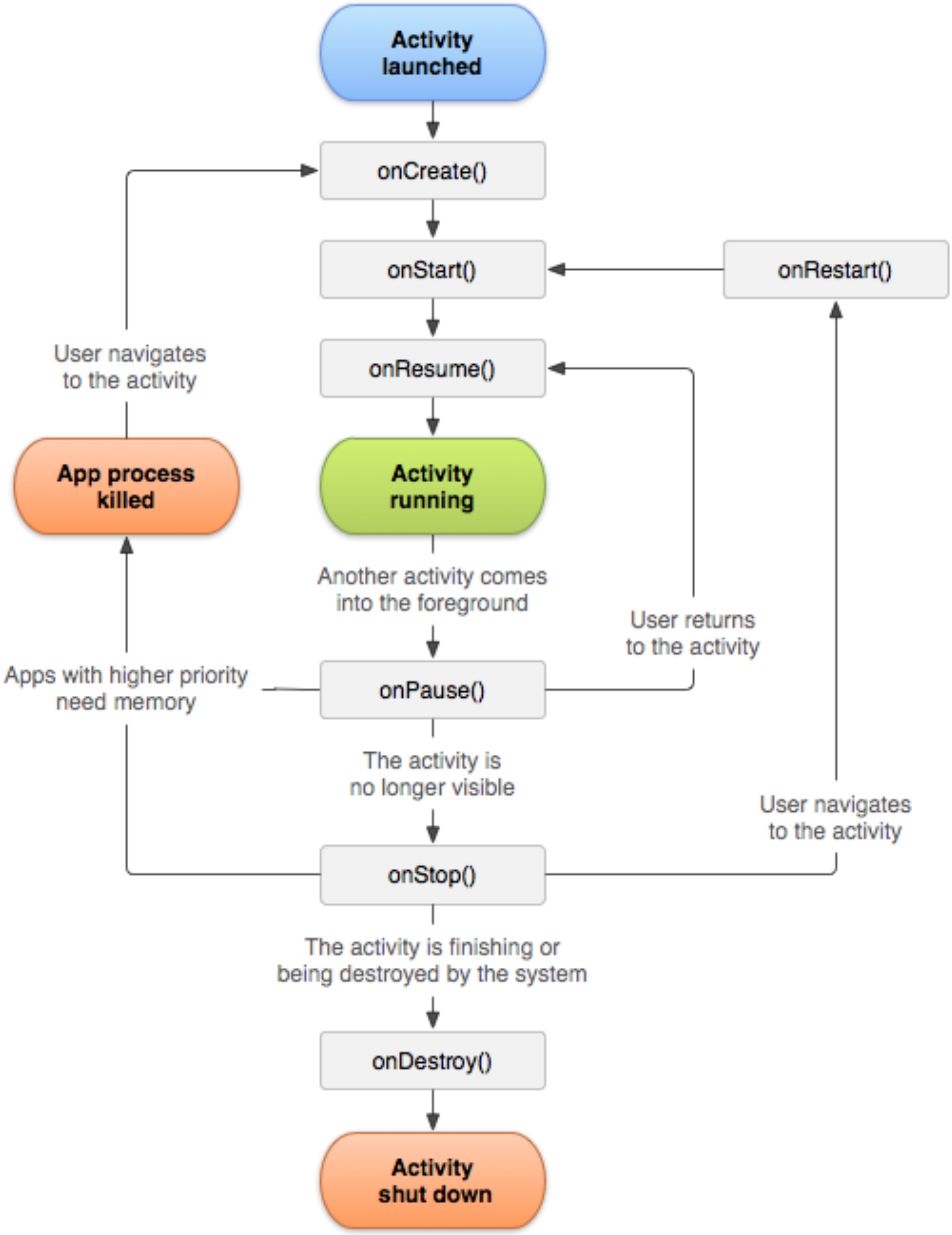
Zastavená (Stopped)

Pokud je aktivita kompletně překryta jinou aktivitou, je zastavena. Stále si pamatuje svůj stav před zastavením, nicméně pokud není delší čas zobrazená uživatelem, tak je okno skryté a může být zničeno systémem, pokud je paměť potřeba někde jinde. [44]

Uvolněná z paměti (Dropped from memory)

Když je aktivita zastavená nebo pozastavená, systém ji může uvolnit z paměti tak, že se zeptá na ukončení nebo prostě zabije její proces. Pokud je uživatelem znovu zobrazena, musí se kompletně restartovat a obnovit svůj předchozí stav.

Na obrázku 3 lze vidět nejdůležitější stavové cesty aktivity. Obdélníky reprezentují zpětná volání (callback method), která může vývojář ve své aktivitě, která je potomkem třídy Activity, implementovat a provést tak požadované operace, když aktivita změní svůj stav. Barevné ovály reprezentují hlavní stavy, ve kterých aktivita může být. [43]

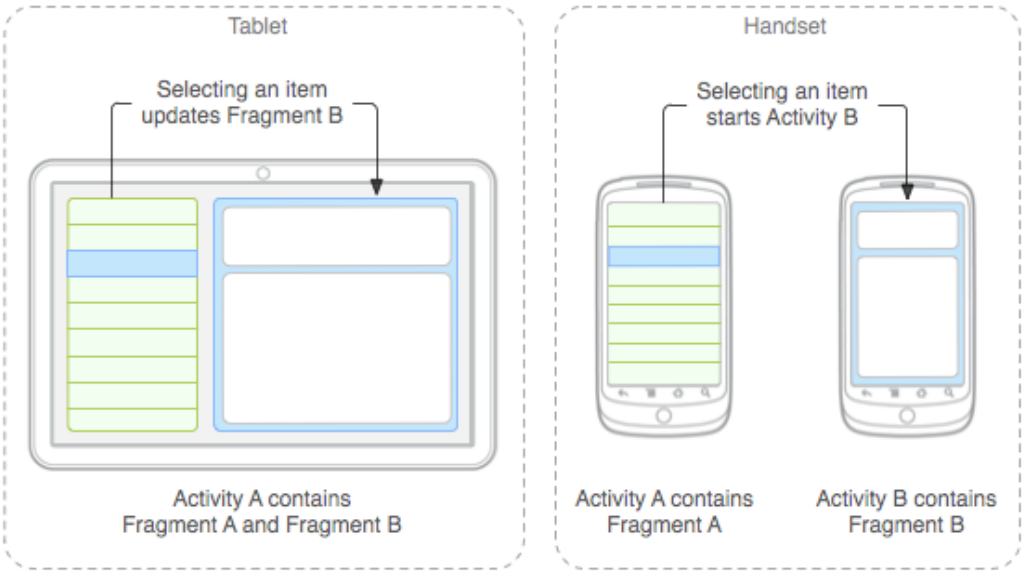


Obrázek 3: Životní cyklus aktivity[3]

2.4 Fragment

Fragmenty byly původně primárně navrženy pro snadnější tvorbu tabletové verze aplikace (zobrazeno na obrázku 4). Díky vhodným kontejnerům TabLayout a NavigationDrawer se však začaly využívat i v případě, kdy vývojář vytváří pouze mobilní verzi aplikace. Fragmenty zapouzdřují část uživatelského rozhraní aktivity a jejich největší výhodou je, že se mohou dynamicky měnit, díky čemuž je jejich načítání značně rychlejší než načítání celých aktivit.

[3] Zdroj: [44]



Obrázek 4: Použití fragmentů při tvorbě tabletové a mobilní verze aplikace[4]

Životní cyklus fragmentu se značně podobá životnímu cyklu aktivity (hlavně z důvodu, že je její součástí). Fragmenty by nikdy neměly komunikovat přímo mezi sebou ale prostřednictvím aktivity. To je možné pomocí zpětných volání, kdy fragment má atribut, jehož typem je rozhraní, které implementuje aktivita. Při připojování fragmentu k aktivitě je tento atribut inicializován a jeho prostřednictví se poté mohou provádět zmíněná zpětná volání.

2.5 Layout

Layout je definice struktury uživatelského rozhraní ve formátu XML. V Android Studiu je možné vytvářet GUI 2 způsoby. Buď se použije vestavěný Designer a prostým přetahování komponent na plátno se vytvoří uživatelské rozhraní anebo se k jeho definici použije přímo jazyk XML.

Výhodné je vytvářet části GUI v oddělených layoutech a poté je pomocí elementu include zahrnout do hlavního layoutu. Layouty jsou poté přehlednější a navíc znovupoužitelné (je možné je současně zahrnout do více hlavních layoutů).

2.6 UI komponenty

UI komponenty jsou objekty tříd, které jsou potomkem třídy View, která obsahuje základní metody (např. pro nastavení rozměrů a pozadí) Tyto komponenty jsou tvořeny buď programově (zřídka) nebo jsou definovány v layoutu spolu s atributem id, pomocí něhož jsou poté asociovány s danými View objekty.

[4] Zdroj: [45]

Layout

Layout je objekt třídy, která je potomkem třídy ViewGroup. Slouží k zapouzdření dalších komponent a určení jejich rozložení.

Rozložení jejich prvků může být různé. Nejjednodušší je LinearLayout, který řadí prvky lineárně za sebe (ať už horizontálně nebo vertikálně). Dále je možné použít RelativeLayout řadící prvky relativně k ostatním prvkům nebo k sobě, TableLayout řadící prvky jako buňky v tabulce a FrameLayout sloužící k zobrazení vždy jen jedné komponenty.

2.7 Level Android API

Jak se Android Framework vyvíjí, postupně vznikají jeho nové verze, které je třeba odlišit. Toto odlišení je zajištěno pomocí tzv. levelu API.

Na začátku vývoje je třeba určit, jakou nejnižší verzi API bude aplikace podporovat. Nízký level znamená vysokou kompatibilitu ale méně funkcí, vysoký level naopak znamená velkou škálu funkcí ale nízkou kompatibilitu. Pro aplikaci v této práci bylo zvoleno jako nejnižší podporované API 19, které odpovídá verzi Androidu 4.4 KitKat a v době začátku tvorby aplikace ho podle Android Studia podporovalo 73,9 % zařízení.

2.8 Support Library

Nabízí možnost používat funkce z novějších verzí API. Poskytuje UI komponenty, které nejsou zahrnuty ve Frameworku, a celou řadu nástrojů, ze kterých mohou aplikace čerpat.

Často se stává že funkce, která může být vývojáři užitečná, není zahrnuta ve verzi SDK, kterou vývojář používá. Tuto situaci řeší Android SDK, které v sobě zahrnuje několik knihoven, které jsou kolektivně nazývané Android Support Library a vývojáři je mohou použít. [46]

3 Vývoj pro Android

V této kapitole jsou popsány možnosti, které jsou k dispozici při vývoji Android aplikací. Dále bude popsáno vývojové prostředí a možnosti testování aplikace

3.1 Programovací jazyk

Kód mobilní aplikace pro Android lze psát v mnoha programovacích jazycích. To je výhoda jak pro začínající programátory (mohou si vybrat jazyk, který se naučí) tak pro programátory s určitými znalostmi (většinou naleznou jazyk, který znají nebo takový, který je tomuto jazyku podobný). V této podkapitole bude popsáno, jaké možnosti vývojář má.

Java

Nejvíce používaný jazyk využívající Android SDK. Tento jazyk je používán v implementační části této práce, jeho použití tedy bude více popsáno v dalších kapitolách.

C, C++

Pro lepší výkon je možné psát aplikace v nativním kódu. Jeho výhodou kromě výkonu je i to, že mnoho kódu je možné sdílet napříč platformami. To znamená, že kód obsahující základní logiku aplikace dostupné pro Android i iOS se nemusí psát dvakrát. Využíván je tentokrát vývojářský balík Android NDK.

C#

Android aplikace lze vyvíjet také ve velmi rozšířeném programovacím jazyce C#. Ten používá nástroj pro tvorbu multiplatformních aplikací Xamarin, který nabízí ještě větší možnosti při tvorbě společného kódu pro více platforem (lze v něm např. vytvořit i společné GUI). [47]

HTML 5, Javascript, CSS

Mobilní aplikace pro Android lze psát také pomocí webových technologií. K tomu slouží nástroj Phonegap založený na projektu Apache Cordova. V podstatě se pomocí HTML 5 a CSS vytváří GUI, podobně jako by se jednalo o webovou stránku, a to je obsluhováno pomocí Javascriptu. Tento nástroj může také využívat některé funkce nativních aplikací jako např. přístup k akcelerometru, fotoaparátu, aktuální poloze nebo lokálnímu uložišti. [48]

Lua

Kdo nechce psát mobilní aplikace v Javě a jejich GUI v XML, ten může využít skriptovací jazyk Lua. Tento skriptovací jazyk je na naučení lehčí než Java a jeho vývojářský balík Corona SDK řeší některé tradiční neduhy Android programování. Corona obsahuje sofistikovaný emulátor, který vývojáři umožní spustit jeho aplikaci okamžitě bez kompilace jejího kódu. Při sestavování konečného APK souboru je využíván online Corona kompilátor. Corona byla

vytvořena primárně pro vývoj her, takže obsahuje hlavně knihovny, které jsou při něm potřebné. [48]

3.2 Android Studio

Android Studio je oficiální vývojové prostředí pro Android, které je založené na vývojovém prostředí IntelliJ IDEA. Jedná se o přetvořenou verzi tohoto IDE, do které jsou přidány funkce určené speciálně k vývoji pro Android. Vývoj tohoto programu navíc vede přímo společnost Google a je tedy ideální pro tvorbu mobilních aplikací pro platformu Android. [49]

Kromě Android Studia lze pro vývoj v programovacím jazyku Java použít také vývojové prostředí Eclipse, ale vzhledem k tomu, že Google nedávno oficiálně ukončil podporu a vývoj nástrojů pro toto IDE [50], lze si již jen těžko představit použití tohoto vývojového prostředí pro vývoj.

Gradle

Součástí Android Studia je také Gradle, což je systém pro sestavení APK souboru. Tento systém na všechny soubory z adresáře projektu (.java nebo .xml) aplikuje vhodné nástroje (např. vezme .class soubory Javy a převede je na .dex soubory) a sdruží je do jednoho komprimovaného APK souboru. Gradle využívá samostatný jazyk DSL založený na Groovy, pomocí něhož je možné specifikovat konvenci využívanou při sestavování. [51]

Kromě toho Gradle nabízí mnoho dalších funkcí. Největší plus tohoto systému je určitě v tom, že je postaven na pluginech. Díky tomu může vývojář rozšířit schopnosti svého projektu použitím nekonečného množství pluginů napsaných v Javě nebo jazyku Groovy. Gradle nabízí kompletní podporu pro vytváření a získávání těchto závislostí z repozitářů ANT, Maven a Ivy. [52]

Java

Android nabízí oficiálně kompletní podporu Javy 1.6 a 1.7. Nejnovější verze 1.8 je však podporována jen částečně (chybí např. java.time API) pomocí kompilátoru nazývaného Jack, který podporuje Android Studio 2.1 a novější. [53] O řešení problémů s nejnovější verzí Javy pojednává část šesté kapitoly.

Pluginy

Kromě Gradle pluginů existují i pluginy pro samotné Android Studio, které díky nim může získat novou funkcionalitu.

V implementační části práce byly použity dva pluginy:

- Genymotion plugin slouží k propojení Android Studia s emulátorem Genymotion (více o něm v následující podkapitole). Po tomto propojení může vývojář vytvořit nebo zapnout Genymotion virtuální zařízení přímo z Android Studia a testovat na něm své aplikace.
- Android Drawable Importer přidá několik možností importu grafických objektů a možnost jejich úpravy do správného rozlišení.

3.3 Emulátor

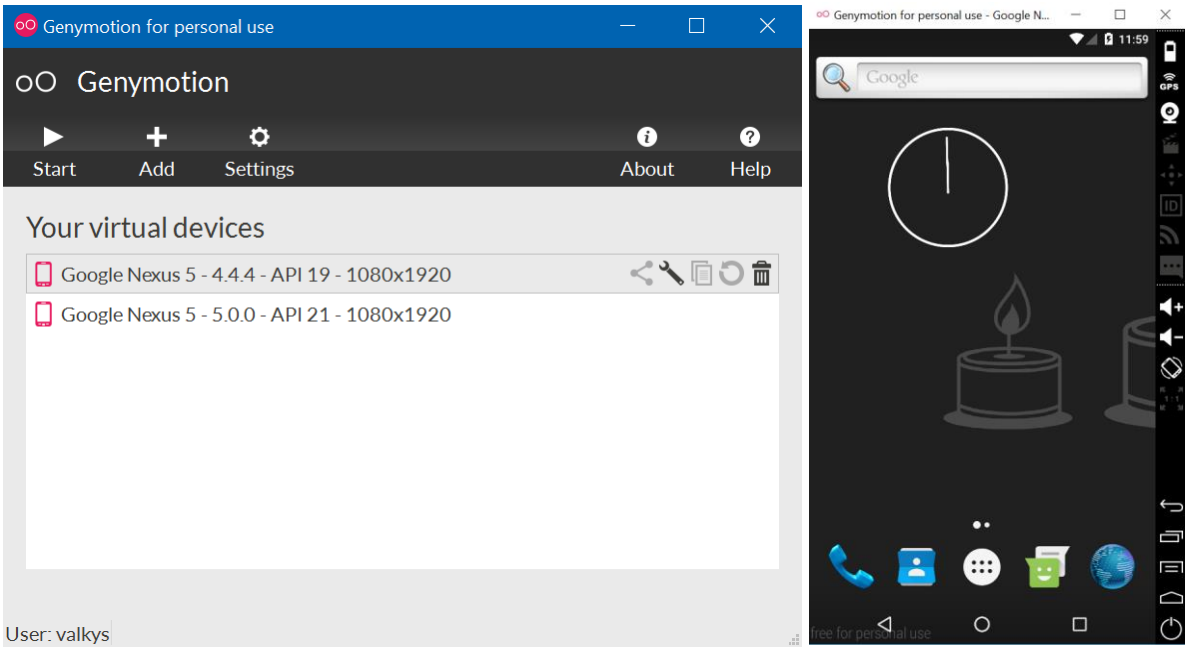
Android emulátor (dále emulátor) je program, ve kterém lze spustit virtualizované mobilní zařízení s OS Android. Vývojář v tomto programu může velmi jednoduše testovat své mobilní aplikace.

Vestavěný emulátor

Emulátor, který je součástí Android SDK, trpí některými nedostatky a omezeními. Prvním z nich je, že při jeho spuštění je vhodné, aby PC vývojáře podporoval HW virtualizaci, jinak je běh emulátoru nesmírně pomalý. Také s podporou HW virtualizace ale není výkon emulátoru zcela ideální, a proto v této práci nebyl použit.

Genymotion

Místo vestavěného emulátoru je vhodnější použít oblíbený Android emulátor Genymotion (zobrazen na obrázku 5), který je založený na emulaci skrze virtualizační program VirtualBox.



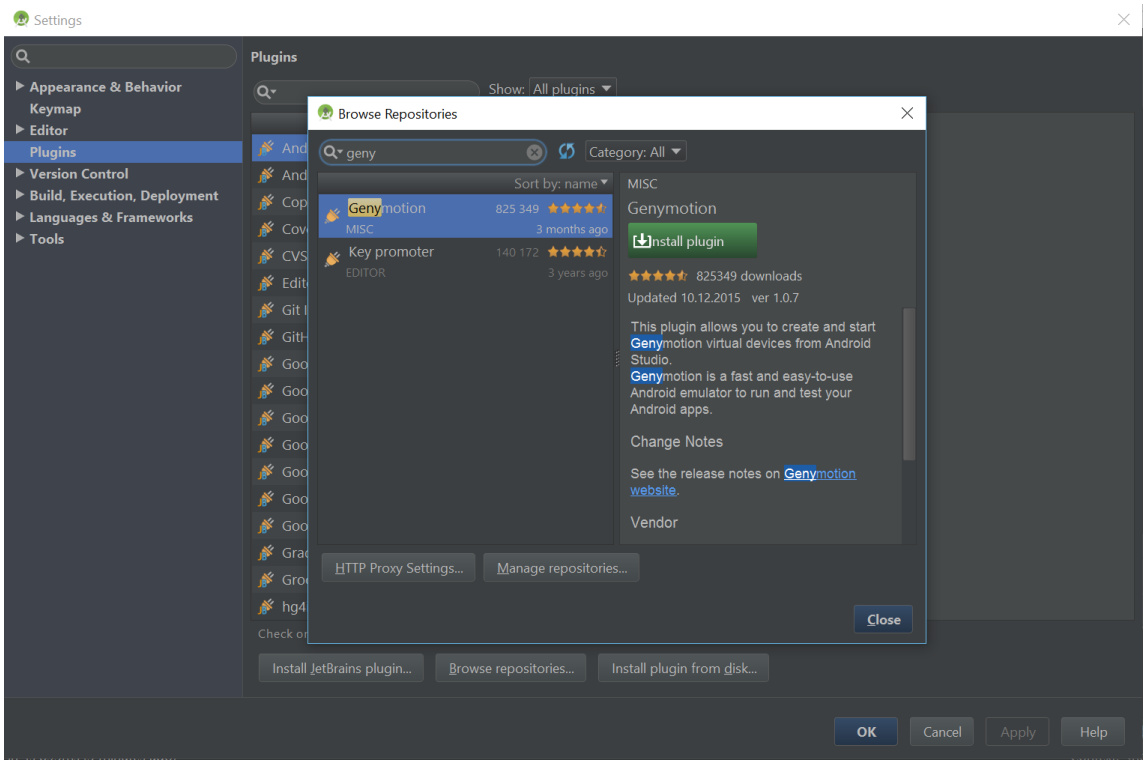
Obrázek 5: Prostředí programu Genymotion a zapnuté virtuální zařízení

Genymotion je o něco výkonnější než vestavěný emulátor, navíc plně podporuje vstupní zařízení, která jsou připojena k vývojářskému PC, což značně usnadňuje ovládání emulátoru (např. možnosti rolování v menu pomocí kolečka myši a psaní na HW klávesnici).

Užitečná je funkce drag and drop, pomocí které lze do zařízení přesouvat kromě běžných souborů také APK soubory aplikací, které se na virtuální zařízení ihned nainstalují.

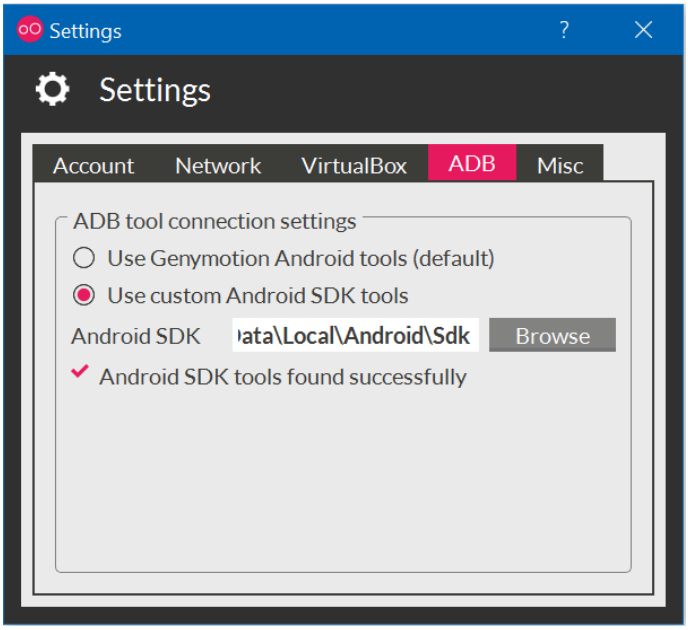
Po registraci na https://www.genymotion.com/ si lze vybrat verzi programu (k dispozici je i bezplatná verze Basic, která obsahuje vše potřebné a byla použita během testování implementační části této BP) a tu stáhnout.

Po stažení programu a jednoduché instalaci, při které se nainstaluje program VirtualBox (pokud již není na PC nainstalován) a v něm se vytvoří virtuální mobilní zařízení, je ještě potřeba propojit Genymotion s Android Studiem pomocí pluginu (zobrazeno na obrázku 6) a restartovat Android Studio. Poté je ještě třeba v nastavení Genymotion zvolit správnou cestu k Android SDK (zobrazeno na obrázku 7) a vše funguje, jak má.



Obrázek 6: Instalace pluginu Genymotion do Android Studia

Někdy se objevují problémy při stažení verze Genymotion, jejíž součástí je i VirtualBox, řešením je stáhnout oba programy samostatně, nainstalovat VirtualBox a poté Genymotion. Virtuální zařízení lze vytvářet a spouštět jak skrze samotný program, tak skrze plugin v Android Studiu.



Obrázek 7: Nastavení cesty k Android SDK v programu Genymotion

3.4 Fyzické zařízení

Kromě emulátoru je možné testovat aplikaci také přímo na některém fyzickém zařízení, které obsahuje verzi OS Android, kterou aplikace podporuje. K tomu je potřeba nainstalovat ovládače daného zařízení na PC vývojáře a povolit v nastavení daného zařízení možnost ladění USB. Poté už stačí pouze připojit zařízení k PC a při spuštění projektu v Android Studiu se zařízení objeví v seznamu připojených zařízení. Po vybrání tohoto zařízení a sestavení APK souboru se aplikace na toto zařízení nainstaluje a spustí.

Testování na fyzickém zařízení je hojně využíváno, protože je většinou podstatně rychlejší a pohodlnější než testování pomocí emulátoru. Při vývoji mobilní aplikace v této práci byla k testování použita zařízení Honor 4C a Lenovo Tab3 7 Essential.

4 Architektura Android aplikace

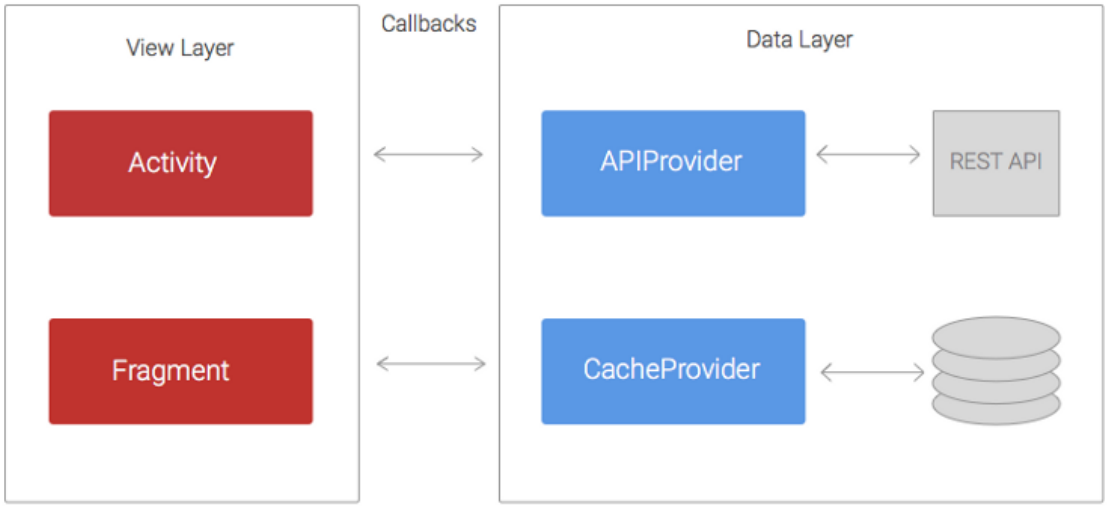
V této kapitole je nastíněno použití jednotlivých architektur při tvorbě Android aplikace. Text se soustředí na pozitiva a negativa, která souvisí s jejich návrhem. Podrobněji je pak zaměřena pozornost na architekturu MVP, která byla použita při návrhu aplikace v implementační části této práce. Na závěr je zhodnocena vhodnost použití jednotlivých architektur při návrhu Android aplikace.

4.1 Naivní architektura

Naivní architektura je obecně ta nejméně vhodná, kterou lze použít při návrhu softwaru. V případě Android aplikací vypadá tato architektura tak, že veškerá dění probíhá v aktivitách nebo fragmentech a dochází k minimální dekompozici kódu. Jedna třída se tedy musí starat o zobrazování dat, komunikaci se serverem, práci s interní SQLite databází a všechny potřebné výpočty. SW vytvářený dle této architektury je těžko modifikovatelný a většinou obsahuje chyby. U větších projektů je použití naivní architektury nemožné.

4.2 VM architektura

Řešením problémů s naivní architekturou je rozložení odpovědnosti do více vrstev aplikace. Nejčastěji používanou architekturou v Android aplikacích je dvouvrstvá View Model architektura (zobrazena na obrázku 8).



Obrázek 8: View Model architektura[5]

[5] Zdroj: [54]

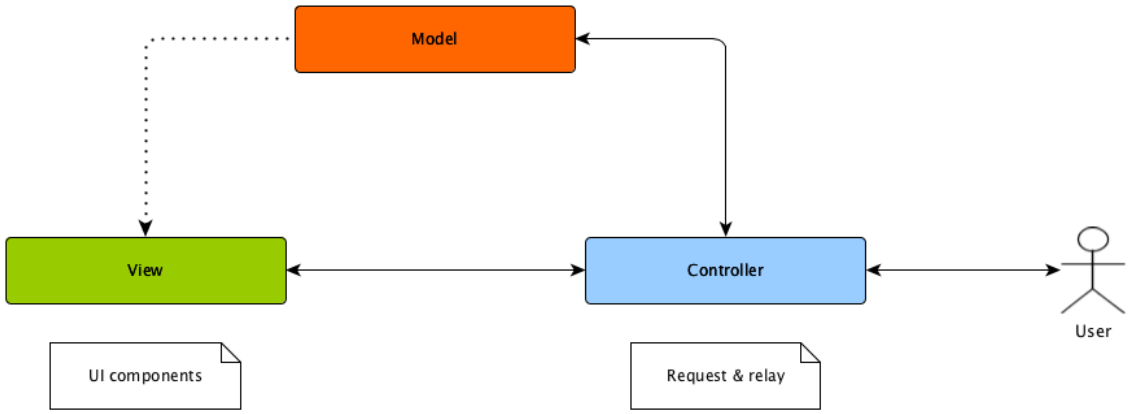
Model je datová vrstva, která se stará např. o komunikaci se serverem (načítání dat získaných z této komunikace), práci s interní SQLite databází, a navíc obsahuje logiku celé aplikace.

View je zobrazovací vrstva, která se stará o zachytávání událostí. Zároveň při zachycení události musí tato vrstva komunikovat s datovou vrstvou a následně zobrazit obdržená data na požadované UI komponenty.

Zobrazovací vrstva je v Androidu reprezentována aktivitou nebo fragmentem. Jak již bylo uvedeno ve druhé kapitole, tyto části Android aplikace mají svůj životní cyklus, se kterým musí počítat zejména dlouhotrvající operace, protože některé objekty, se kterými pracují, nemusí po ukončení této operace již vůbec existovat. Z tohoto důvodu se postupně zvyšuje složitost této vrstvy a počet závislostí mezi objekty v ní, což znovu vede k její obtížné udržovatelnosti.

4.3 MVC architektura

Vylepšení MV architektury se naskýtá v přidání další vrstvy. Toho lze dosáhnout použitím architektury Model View Controller, která mezi datovou a zobrazovací vrstvu přidává tzv. Controller (zobrazeno na obrázku 9), čímž dojde k zjednodušení zobrazovací vrstvy. Controller může zároveň obsluhovat více View najednou.



Obrázek 9: Model View Controller architektura[6]

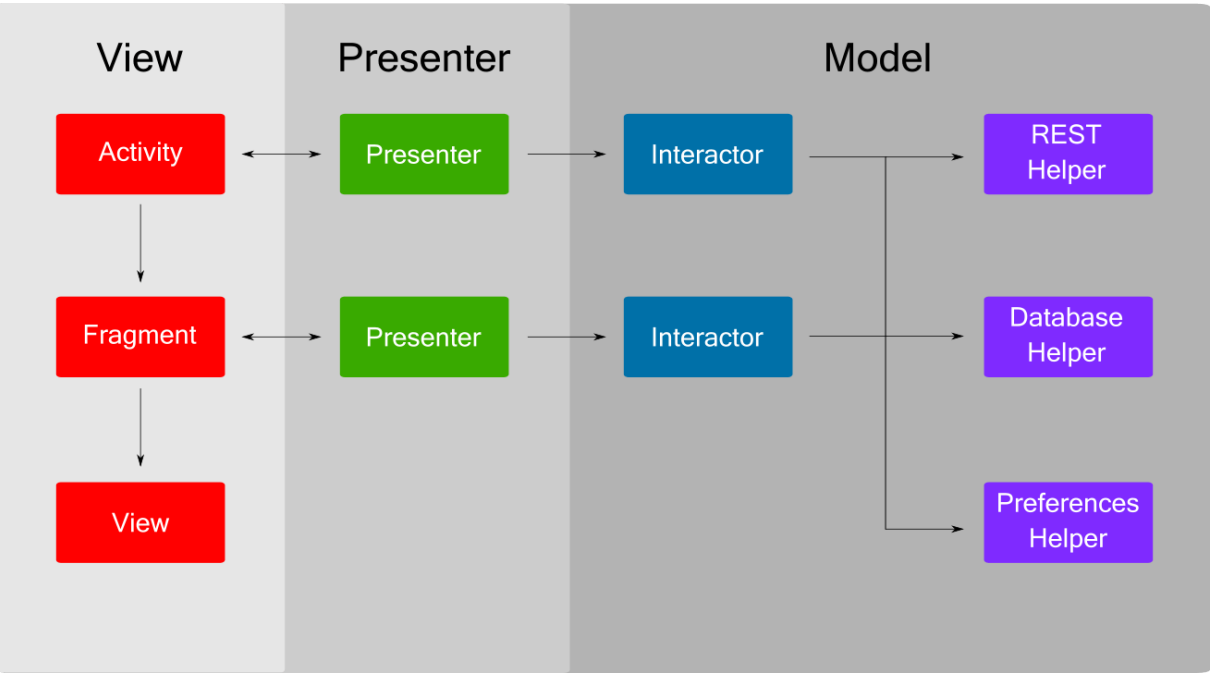
MVC není pro Android úplně vhodný. Aktivita (či fragment) je totiž na pomezí mezi View a Controllerem. Jako taková totiž není potomkem základní třídy View, ale skládá se z komponent, které potomkem View jsou. Zároveň také složí k zachytávání událostí komponent, ze kterých se skládá. MVC lze tedy implementovat pouze tehdy, když bude aktivita

[6] Zdroj: [55]

brána jako pseudo Controller, což ale povede k tomu, že kód bude stále rozdělen pouze do dvou vrstev.

4.4 MVP architektura

Rozšíření na třívrstvý model lze dosáhnout, pokud Controller bude nahrazen Presenterem (zobrazeno na obrázku 10). Ten na rozdíl od Controlleru neslouží pro uživatelský vstup, díky čemuž je při tvorbě Android aplikace skutečně možné rozdělit kód do tří vrstev.



Obrázek 10: Model View Presenter architektura[7]

Presenter představuje prezentační vrstvu. Má za úkol reagovat na události zasílané zobrazovací vrstvou, předat zprávu datové vrstvě, která provede danou operaci a pomocí zpětného volání informuje Presenter o úspěšnosti operace, případně také zašle data, která Presenter předá zobrazovací vrstvě k zobrazení.

Protože veškerá data, která View zobrazí, získává z prezentační vrstvy, je v ní uložen i jeho stav. Z tohoto důvodu má každý View svůj vlastní Presenter, přičemž jeden Presenter nemůže sdílet více View. Probíhá to tak, že když je View uvolněn z paměti, Presenter uloží jeho stav, který poté může restartovaný View znova načíst.

Tímto se z View stává zcela pasivní vrstva, která neobsahuje žádnou logiku aplikace. Je tedy dovršeno oddělení podoby GUI od logiky kódu, díky čemuž je změna vzhledu aplikace velmi snadná a vývojář se při ní nemusí bát, že by se aplikace po změně vzhledu mohla začít chovat

[7] Zdroj: [56]

jinak. Další výhodou je samozřejmě snazší tvorba testů díky odstranění závislosti View na Modelu.

Datová vrstva má více různých částí než předchozí dvě vrstvy a jak lze vidět na obrázku 10, prezentační vrstva s ní komunikuje pomocí objektů nazvaných Interactory. Interactor je obalovací třída, která obsahuje logiku aplikace a zároveň přímo komunikuje s objekty spravujícími nejrůznější služby, jako je např. obsluha interní SQLite databáze nebo komunikace se vzdálenými servery. Tyto objekty Interactor získá pomocí DI (více o používání DI při vývoji Android aplikace v další kapitole).

Implementace prezentační a zobrazovací vrstvy

Řešení, jak implementovat MVP architekturu při tvorbě Android aplikace, je mnoho. V této práci byla částečně použita externí knihovna Nucleus [57], která kromě jiného nabízí rozhraní pro tvorbu Presenteru a jeho spojení s View.

V prezentační vrstvě musí být každý Presenter potomkem generické třídy Presenter (typový parametr označuje dané View), která má definované jednoduché metody pro vytvoření Presenteru, připojení/odpojení View a uložení stavu.

Vytvoření zobrazovací vrstvy probíhá tak, že je zvolen View, který se použije (aktivita nebo fragment), a vytvoří se jeho abstraktní generický potomek (typový parametr označuje daný Presenter), který řádně implementuje generické rozhraní ViewWithPresenter a je poté používán jako rodič vytvářených View. Každý nově vytvořený View pak navíc musí před definicí své třídy označit třídu svého Presenteru pomocí anotace *@RequiresPresenter*.

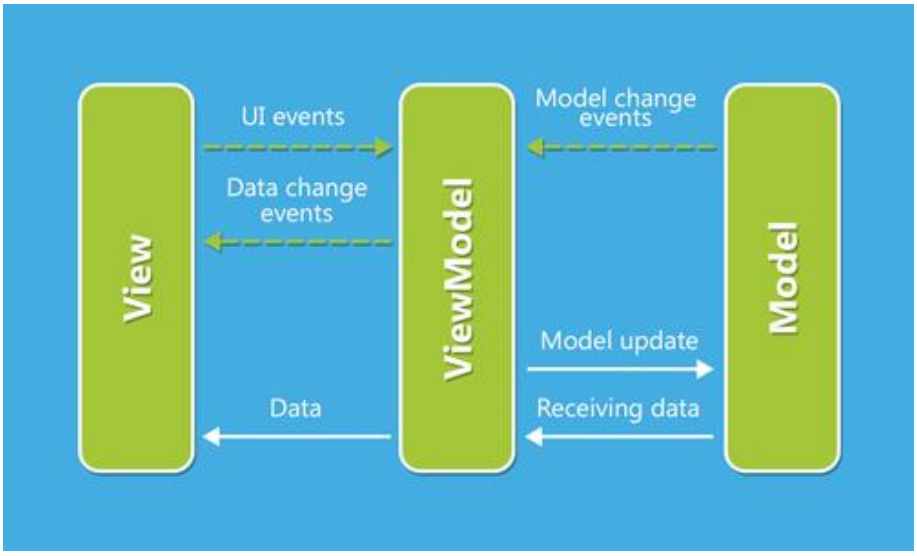
Pro lepší představu, jak probíhá vytvoření těchto dvou vrstev, slouží vývojový diagram v příloze A, který abstrahuje od některých detailů a zaměřuje se na propojení těchto dvou vrstev.

4.5 MVVM architektura

Architektura nazývaná Model View ViewModel (zobrazena na obrázku 11) se začala používat ve světě Android aplikací po představení knihovny Data Binding od Googlu. Tato knihovna přinesla řešení podobné řešením z jiných platforem, kdy je možné svázat hodnoty dat aplikace přímo s komponentami definovanými v XML layoutu.

V této architektuře je prostřední vrstvou tzv. ViewModel. Tato vrstva, stejně jako v Presenter v MVP architektuře, představuje prostředníka mezi zobrazovací a datovou vrstvou, je však spíše blíže View než Modelu. ViewModel má za úkol veřejně vystavit příslušná data (tyto data

přijímá z modelu), která vyžaduje View. Ten svůj obsah čerpá právě z ViewModelu, se kterým jsou pomocí posluchačů (pozorovatelů)[8] propojeny definice jeho UI komponent obsažené v layoutu. Po tomto propojení probíhá mezi View a ViewModelem obousměrná aktualizace dat, což je výhoda oproti ostatním architekturám, kde může probíhat pouze jednosměrná aktualizace. [58]



Obrázek 11: Model View ViewModel architektura[9]

Přes množství výhod však tato architektura obsahuje také poměrně velké množství nevýhod. Hlavní nevýhodou je zvyšující se složitost layoutů. Do jejich struktury je např. přidávána prezentační logika, lambda výrazy na posluchače a import tříd, které posluchači potřebují. To vede ke špagetovému kódu. Další nevýhody se pak např. objevují v situaci, když aplikace vyžaduje rozdílné layouty pro rozdílné úhlopříčky. Vývojář pak musí duplikovat svazovací logiku do každého layoutu a držet oba layouty aktualizované, což je zbytečné. [60]

4.6 Závěr

Jak již bylo uvedeno na začátku této kapitoly, použití naivní architektury není u netriviálních aplikací možné. Vylepšení této architektury představuje dvouvrstvá VM architektura, která je již sice aplikovatelná, ale se zvyšující složitostí aplikace se její kód stane postupem času obtížně udržovatelným. Vzhledem k povaze Android Frameworku nepomůže k dekompozici na tři vrstvy ani MVC architektura. Nejvýhodnější je tedy při návrhu Android aplikace použit MVP nebo MVVM architekturu. Výběr mezi těmito dvěmi architekturami pak závisí hlavně na druhu aplikace a subjektivních preferencích vývojáře.

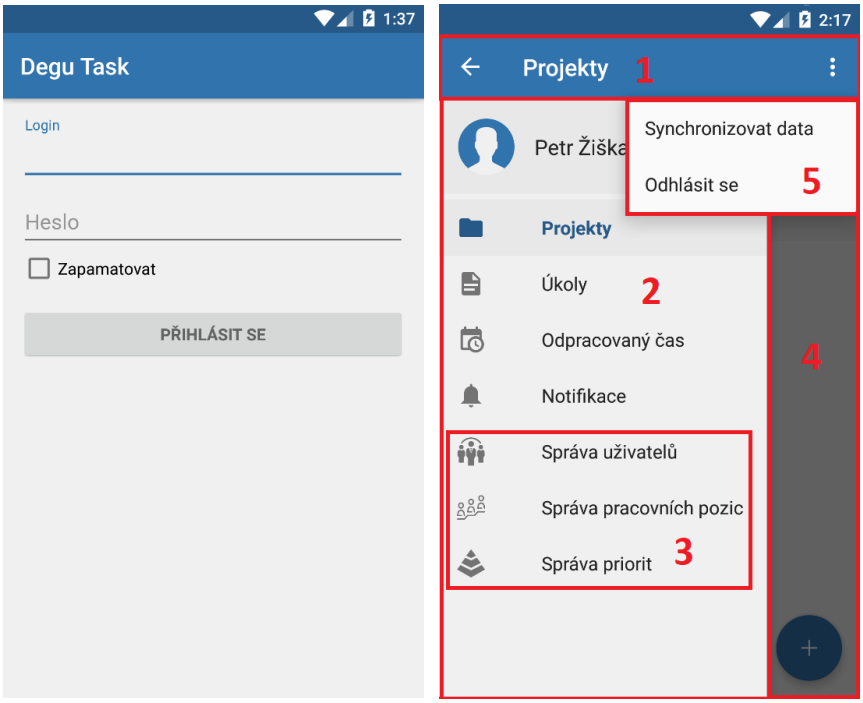
[8] Návrhový vzor Listener, resp. Observer.
[9] Zdroj: [59]

5 Představení aplikační části práce

V této kapitole je představeno GUI mobilní aplikace vyvíjené v rámci praktické části této práce. Toto představení probíhá formou popisu jednotlivých obrazovek aplikace.

Přihlašovací aktivita

Na obrázku 12a je možné vidět přihlašovací obrazovku. Uživatel může po vyplnění údajů zaškrtnout možnost *Zapamatovat*, díky čemuž nemusí při dalším přihlašování znova vyplňovat údaje. Tato obrazovka je přeskočena v případě, že se uživatel naposledy neodhlásil a má v interní databázi uložený platný autentizační klíč.



a) Login b) Základní GUI

Obrázek 12: Vzhled uživatelského rozhraní pro přihlašování a správu

Základní layout

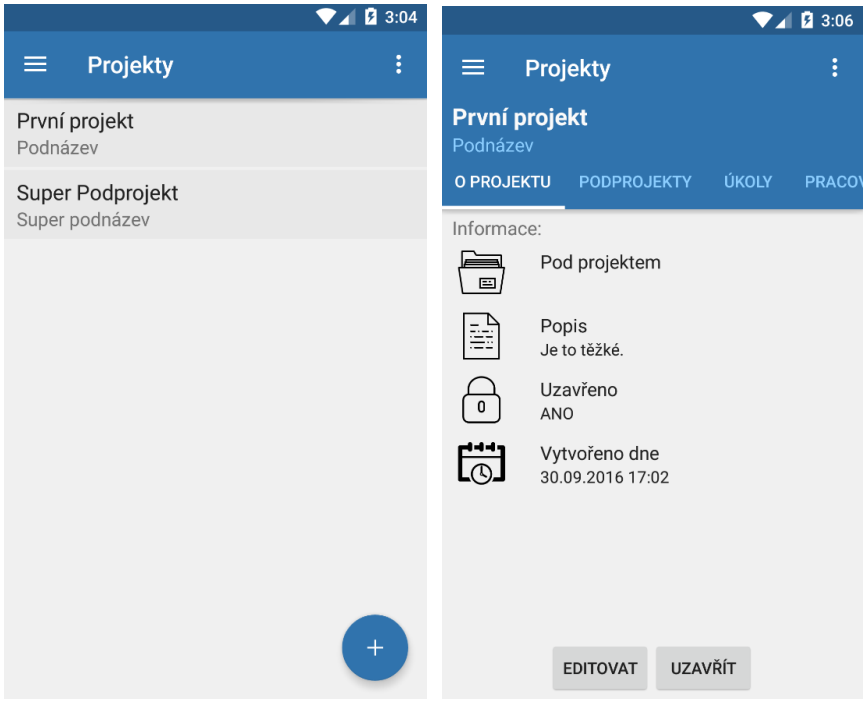
Po úspěšném přihlášení se uživatel přesune na obrazovku hlavní aktivity. Obrázek 12b ukazuje rozložení jejího základního layoutu:

1. Toolbar obsahuje tlačítko pro zobrazení bočního menu, název právě procházené části aplikace a tlačítko pro zobrazení rolovacího menu.
2. Boční menu obsahuje navigaci v rámci aplikace
3. Součástí bočního menu je skupina položek, která se zobrazuje pouze uživatelům, kteří mají potřebná oprávnění.

4. Kontejner obsahující fragment zobrazující obsah právě procházené části aplikace.
5. Rolovací menu obsahuje položky, s jejichž pomocí může uživatel synchronizovat data nebo se odhlásit.

Projekty

Na obrázku 13a lze vidět seznam projektů, který obsahuje položky tvořené názvem a podnázvem kořenových projektů. Po klepnutí na položku je uživatel přesunut na obrazovku projektu.



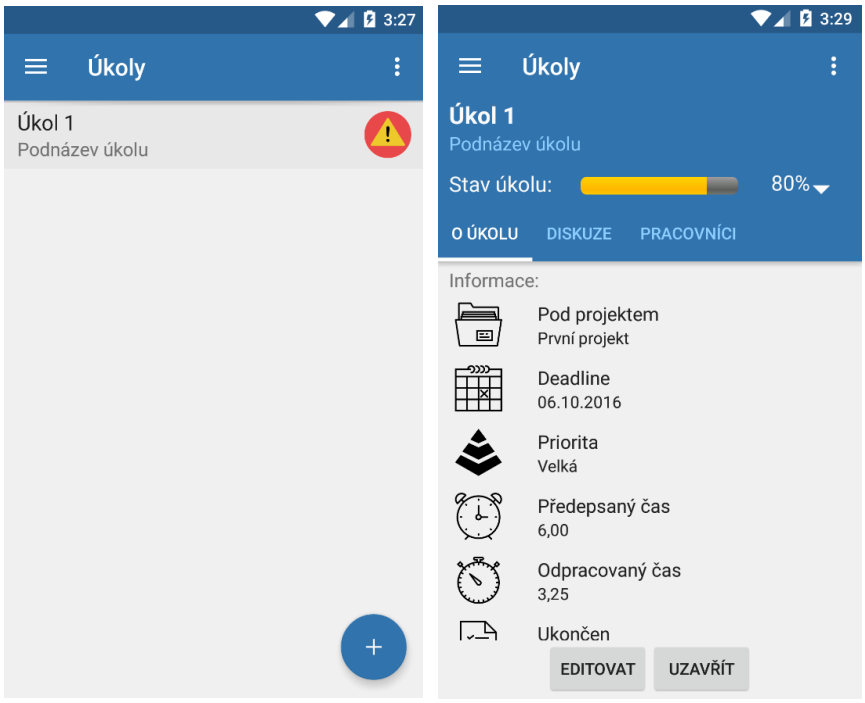
a) Projekty b) Detail projektu

Obrázek 13: Uživatelské rozhraní pro projekty

Detail projektu na obrázku 13b je tvořen nejprve názvem a podnázvem projektu. Pod nimi se poté nachází taby, do kterých jsou rozděleny jednotlivé části projektu. První tab ukazuje základní informace o projektu. Pokud má uživatel oprávnění editovat a uzavírat projekty, zobrazí se mu i příslušná tlačítka. V příloze B lze pak nalézt také podobu tabů se seznamem podprojektů a úkolů. Po klepnutí na požadovanou položku je uživatel přenesen na její obrazovku.

Úkoly

Obrázek 14a zobrazuje seznam úkolů seřazených podle priority tvořených jejich názvem a podnázvem. Úkoly, jejichž deadline již vypršel a nebo za nedlouho vyprší, jsou označeny ikonou vykřičníku. Po klepnutí na požadovanou položku je uživatel přenesen na obrazovku daného úkolu.



a) Úkoly b) Detail úkolu

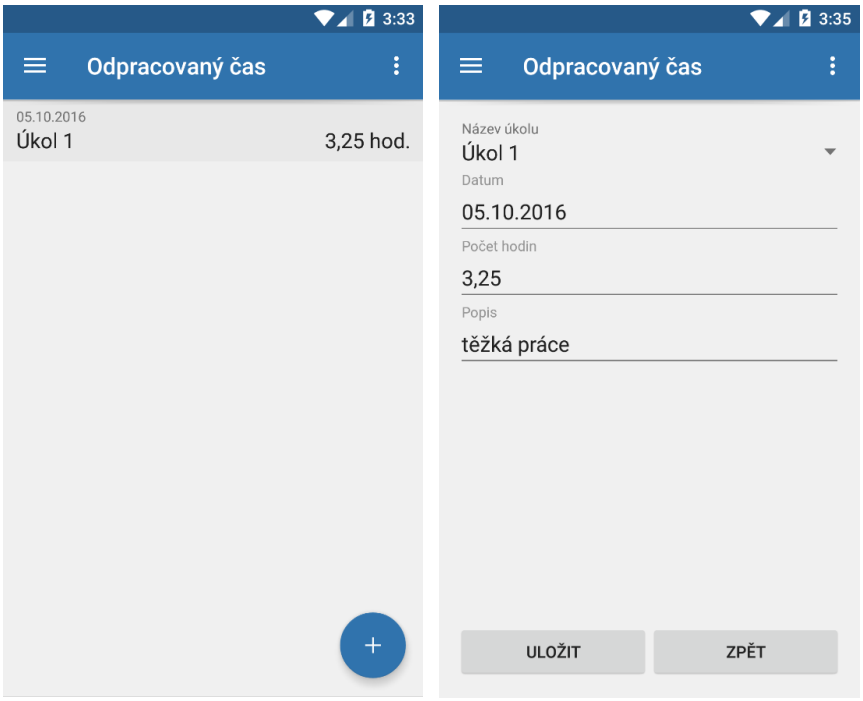
Obrázek 14: Uživatelské rozhraní pro úkoly

Detail úkolu na obrázku 14b je nejprve tvořen názvem a podnázvem úkolu. Pod nimi se nachází progress bar zobrazující informaci o stavu úkolu, kterou lze upravovat. Pod ním jsou taby, do nichž jsou rozděleny zbývající části úkolu. V příloze B lze nalézt podobu tabů zobrazujících diskuzi k úkolu a seznam lidí pracujících na úkolu.

Odpracovaný čas

Na obrázku 15a lze vidět seznam odpracovaných časů. Položky jsou tvořeny datem času, názvem úkolu a počtem odpracovaných hodin v daném datu. Po klepnutí na některou položku je uživatel přenesen obrazovku, kde může odpracovaný čas upravit.

Obrázek 15b zobrazuje editaci odpracovaného času. Nejprve lze vybrat úkol, ke kterému je odpracovaný čas přidáván. Poté lze zvolit datum, kdy byl čas odpracován, počet hodin a napsat krátký popis.

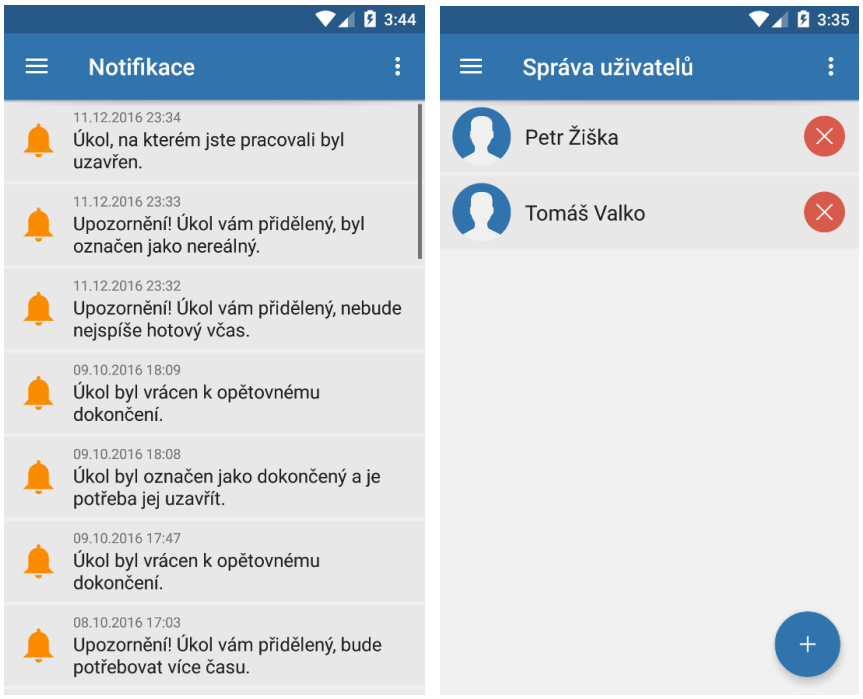


a) Odpracovaný čas b) Editace odpracovaného času

Obrázek 15: Uživatelské rozhraní pro odpracovaný čas

Notifikace

Na obrázku 16a je seznam nepřečtených notifikacích, klepnutím na notifikaci se daná položka označí jako přečtená a uživatel je přenesen na obrazovku úkolu, k němuž notifikace náleží.



a) b)

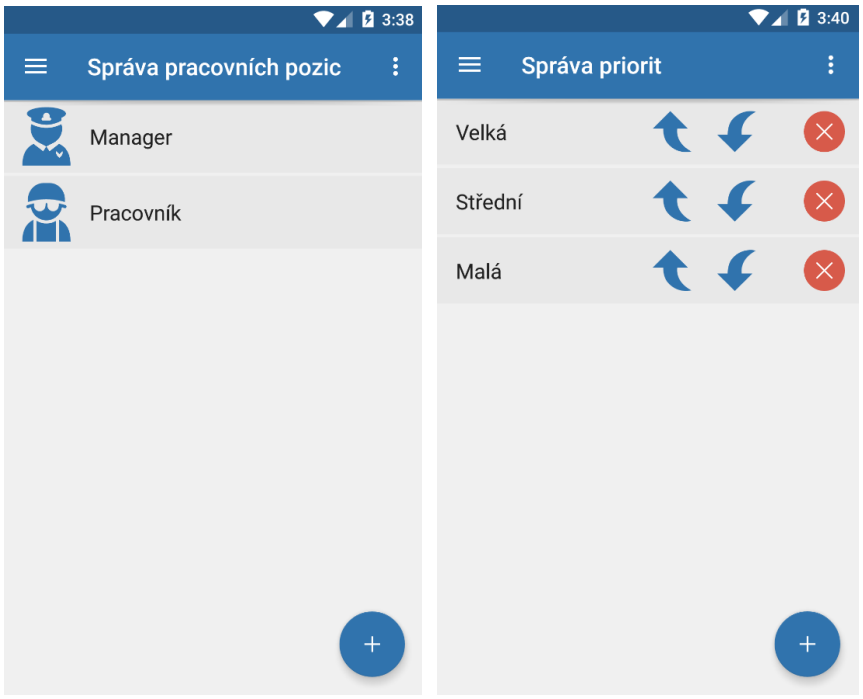
Obrázek 16: Uživatelské rozhraní pro notifikace a správu uživatelů

Správa uživatelů

Obrázek 16b ukazuje seznam uživatelů, které lze spravovat. Uživatelé lze zablokovat nebo přidat nové. Správa uživatelů je umožněna pouze uživateli, který má potřebné oprávnění.

Správa pracovních pozic

Obrázek 17a zobrazuje obrazovku, kde lze spravovat pracovní pozice. Spravovat pracovní pozice může pouze uživatel s potřebným oprávněním.



a) b)

Obrázek 17: Uživatelské rozhraní pro správu pracovních pozic a priorit

Správa priorit

Na obrázku 17b lze vidět seznam priorit. Ty lze přidávat a měnit jejich pořadí. Priority může spravovat pouze uživatel s oprávněním k tomuto úkonu.

6 Komplikace při vývoji aplikace a jejich řešení

Následující kapitola pojednává o problémech spojených s vývojem Android aplikací.

6.1 Problémy vývojového prostředí

Android Studio je pokročilým vývojovým prostředím umožňujícím tvorbu rozsáhlých aplikací. Obsahuje ale také některé problémy, které každý den komplikují život velkému množství vývojářů. [61]

Gradle a konflikty závislostí

Jak již bylo uvedeno ve třetí kapitole, Gradle je automatizovaný systém, který se stará o sestavení projektu a je součástí Android Studia. Častým problémem při sestavování projektu pomocí tohoto systému jsou konflikty závislostí. Při řešení tohoto problému bylo čerpáno především z [62].

Gradle téměř nezná automatizaci řešení závislostí, což způsobuje velké množství problémů především rozsáhlejších projektech. K problému ale stačí situace, kdy dvě závislosti mají závislost na rozdílné verze stejné knihovny.

Jakmile existuje konflikt závislostí v sestavení, vývojář musí rozhodnout, jaká verze knihovny bude nakonec do sestavení zahrnuta. Existuje několik způsobů, jak takový konflikt vyřešit:

- První možností je vyloučit konfliktní modul/knihovnu z jedné ze závislostí. To je možné udělat během deklarace závislosti (zobrazuje ukázka níže). Záporem tohoto řešení je, že v rozsáhlém projektu se konflikt může týkat mnoha závislostí a u každé je pak potřeba tímto způsobem vyloučit konfliktní modul.

```
androidTestCompile('com.android.support.test:runner:0.5')
androidTestCompile('com.android.support.test.espresso:espresso-
core:2.2.2', {
    exclude group: 'com.android.support', module: 'support-
annotations'
})
```

- Další možností je explicitní specifikace konfliktní knihovny. V tomto případě je třeba zmínit verzi knihovny, která má být zahrnuta do sestavení pro některou z konfigurací (zobrazeno na ukázce níže). Pokud tedy projekt obsahuje více závislostí, které jsou v konfliktu, je možné pouze jednou explicitně specifikovat verzi konfliktní knihovny, čímž lze získat čistší řešení. Nevýhodou ale je že vývojář při aktualizaci závislostí potřebuje aktualizovat také verzi explicitně specifikované konfliktní knihovny.

```
androidTestCompile('com.android.support.test.espresso:espresso-
core:2.2.2')
androidTestCompile('com.android.support.test:runner:0.5')
androidTestCompile('com.android.support-annotations:25.3.1')
```

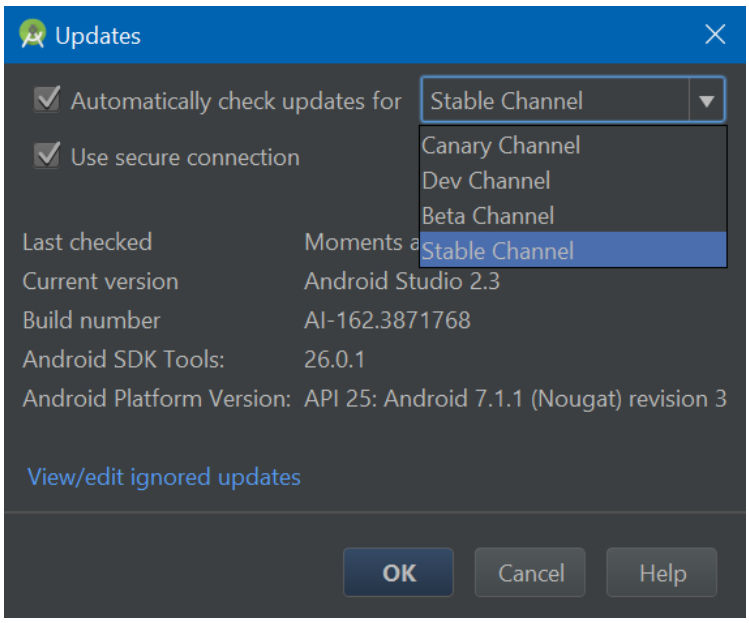
- Konflikt lze vyřešit také vnucením konkrétní verze knihovny všem závislostem, kterých se týká (zobrazeno na ukázce níže). Tento přístup při řešení konfliktu by však měl být používán s extrémní opatrností, protože jsou s jeho použitím spojené některé vážné důsledky. V případě, že závislosti aktualizovaly verzi konfliktní knihovny, bude totiž stále vnucená zastaralá verze. Ačkoli je tento scénář podobný předchozí možnosti, tak při tomto přístupu bude vnucena verze závislé knihovny všem konfiguracím namísto jedné konfigurace, která může být chybná.

```
android {
   configurations.all {
      resolutionsStrategy.force 'com.android.support-
annotations:25.3.1'
   }
}
```

Aktualizace

Google se snaží své IDE rychle zlepšovat, což se mu poměrně daří. Na druhou stranou není výjimkou, že se po aktualizaci nepovede zkompilovat dříve funkční projekt.

Chyby IDE jsou poměrně časté a jsou důsledkem rychlého vydávání nových verzí IDE. Základní prevencí proti nim je přepnutí v nastavení Android Studia na stabilní kanál aktualizací (zobrazeno na obrázku 18). Bohužel však ani stabilní kanál není zdaleka bez chyb. [63]



Obrázek 18: Změna aktualizačního kanálu Android Studia

Spolu s Android Studiem se také aktualizuje vývojářský balík Android SDK. Při jeho vývoji Google porušuje některé principy (např. jsou občas změněna některá již dříve zveřejněná rozhraní), které vedou k tomu, že se v kódu může najednou objevit množství chyb.

U Android Studia se tedy určitě vyplatí s aktualizacemi počkat a sledovat nejprve na internetových fórech množství chyb, které je s konkrétní aktualizací spjato. Vzhledem k velkému rozšíření tohoto IDE je navíc většina chyb známá do několika dní.

6.2 Vázání UI komponent součástí jejich inicializace

Jednou z nejnepříjemnějších věcí při vývoji Android aplikací je svazování UI komponent s jejich definicemi v XML layoutu. V aktivitě to totiž probíhá jako součást inicializace, která je tak potřeba provést pro každou komponentu zvlášť.

Inicializace proběhne tak, že se nejprve nastaví layout, který aktivita používá, a následně se pro každou používanou komponentu najde podle id konkrétní View, které je v daném layoutu definované, a přetypuje se na požadovanou komponentu. Kód na začátku aktivity v běžné aplikaci tedy vypadá přibližně tak jako na ukázce níže.

```
public class LoginActivity {

    TextView textViewUsername;
    TextView textViewPassword;
    Button buttonSignIn;

    @Override
    public void onCreate(Bundle bundle) {
        setContentView(R.layout.activity_login);
        textViewUsername = (TextView) findById(R.id.textViewUsername);
        textViewPassword = (TextView) findById(R.id.textViewPassword);
        buttonSignIn = (Button) findById(R.id.buttonSignIn);
    }
}
```

Butter Knife

Řešení nabízí externí knihovna Butter Knife [64], která odděluje svazování komponent od jejich inicializace a zjednodušuje tímto kód.

Tato knihovna je založena na anotacích, pomocí nichž řeší svázání komponent s jejich definicemi v XML layoutu. Následná inicializace proběhne pro všechny komponenty zároveň zavoláním jediné statické metody (zobrazuje ukázka níže).

```
public class LoginActivity {

    @BindView(R.id.textViewUsername) TextView textViewUsername;
    @BindView(R.id.textViewPassword) TextView textViewPassword;
    @BindView(R.id.buttonSignIn) Button buttonSignIn;

    @Override
    public void onCreate(Bundle bundle) {
        setContentView(R.layout.activity_login);
        ButterKnife.bind(this);
    }
}
```

Na rozdíl od poměrně nedávno (2016) představené interní knihovny Data Binding, která nabízí v tomto směru podobné možnosti, lze takto také svázat UI komponenty definované v layoutech, jež jsou zahrnuté do jiného layoutu pomocí elementu *include*, což je velká výhoda při použití složitých layoutů skládajících se z mnoha částí.

Podobné možnosti jako u aktivit jsou dostupné také v případě fragmentů a adaptérů (které slouží k specifikaci podoby jednoho řádku seznamu). Kromě toho tato knihovna nabízí také např. svazování předdefinovaných zdrojů (barev, rozměrů, hodnot) a posluchačů komponent.

Používáním této knihovny lze tedy dosáhnout toho, že kód aplikace je srozumitelnější, kratší a jeho modifikace se stává jednodušší.

6.3 Předávání závislostí

Jedním z největších problémů, který provází vývoj softwaru, je předávání závislostí různým částem aplikace. Velká část vývojářů se toto snaží obejít používáním návrhového vzoru Singleton, díky kterému jsou nejpoužívanější závislosti globálně přístupné v rámci celé aplikace. Tento přístup ale není vždy ideální, jak je široce diskutováno např. v [30].

Dependency Injection

Řešení nabízí Dependency Injection (dále DI), které navíc usnadňuje modifikaci kódu v momentě, kdy je potřeba změnit způsob vytváření některé závislosti (stačí modifikovat pouze třídu, která se o toto vytváření stará). Android Framework sám o sobě DI nepodporuje a je tedy vhodné použít některou z externích knihoven, které použití DI usnadňují

Dagger 2

Nejlepší volbu představuje knihovna Dagger 2 [65]. Tato knihovna je tvořena několika částmi. Moduly jsou třídy, jejichž metody nabízejí závislosti. Jsou označeny anotací *@Module* a každá jejich metoda je označena anotací *@Provides*, část modulu z praktické části práce zobrazena v ukázce níže.

```
@Module
public class ApplicationModule {

    private final DeguTaskApplication app;

    public ApplicationModule(DeguTaskApplication app) {
        this.app = app;
    }

    @Provides @PerApplication
    public SharedPreferences providesSharedPreferences() {
        return app.getSharedPreferences("settings", Context.MODE_PRIVATE);
    }

    @Provides @PerApplication
    public Context providesApplicationContext() {
        return app.getApplicationContext();
    }

    @Provides @PerApplication
    public MicroOrm providesMicroOrm() {
        return new MicroOrm.Builder()
                .registerTypeAdapter(LocalDate.class, new
                        TypeAdapters.LocalDateAdapter())
                .registerTypeAdapter(LocalDateTime.class, new
                        TypeAdapters.LocalDateTimeAdapter())
                .build();
    }
}
```

Konzumenti jsou třídy, které očekávají předání závislosti. Ty se dají předat konstruktorem (nejlepší řešení), metodou (připadá v úvahu pouze, když objekt předává sám sebe do závislosti) nebo vlastností (která však v tom případě musí zůstat veřejná). V níže zobrazené ukázce z praktické části práce je anotací *@Inject* označen konstruktor takové třídy.

```
@PerApplication
public class ServerActions {

    private final Context context;
    private final Deserializer deserializer;
    private final Serializer serializer;

    @Inject
    public ServerActions(Context context, Deserializer deserializer,
                Serializer serializer) {
        this.context = context;
        this.deserializer = deserializer;
        this.serializer = serializer;
    }
}
```

Důležitou částí je komponenta, která vytváří most mezi moduly a konzumentem. Komponenta je rozhraní označené anotací *@Component*. Pomocí parametrů vstřikovacích metod se zde definují třídy, které dané závislosti potřebují (níže zobrazeno na ukázce z praktické části práce).

Dále je zde možnost definovat metody, které závislosti komponenty případně předávají další komponentě, která tuto komponentu využívá jako vlastní závislost.

```
@PerActivity
@Component(dependencies = ApplicationComponent.class, modules =
        LoginBindModule.class)
public interface LoginComponent {
    void inject(LoginPresenter presenter);
}
```

Samotné předání závislosti poté probíhá inicializováním komponenty pomocí vygenerovaného stavitele[10] a vstříknutím poskytovaných závislostí do dané třídy (níže zobrazeno na zkrácené ukázce z praktické části práce). Komponenty a moduly využívající bezparametrický konstruktor se doplní automaticky a nemusí se uvádět.

```
@RequiresPresenter(LoginPresenter.class)
public class LoginActivity extends NucleusAppCompatActivity<LoginPresenter>
        implements NetCheck.OnNetCheckListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setInjector();
    }

    private void setInjector() {
        ApplicationComponent appComponent =
((DeguTaskApplication)getApplication()).getApplicationComponent();
        LoginComponent loginComponent = DaggerLoginComponent.builder()
                .applicationComponent(appComponent).build();
        loginComponent.inject(getPresenter());
    }
}
```

6.4 Podpora Javy 1.8

Java ve verzi 1.8 obsahuje mnoho užitečných funkcí, jako jsou např. lambda výrazy nebo java.time API. Podpora této verze ze strany Googlu je ovšem značně problematická. Jak již bylo zmíněno ve čtvrté kapitole, částečná podpora nejnovější verze Javy je poskytnuta prostřednictvím kompilátoru Jack.

Jack ale obsahuje poměrně velké množství chyb a současně neumožňuje použití externích knihoven Butter Knife a Dagger 2 kvůli konfliktům anotačních procesorů. Navíc byl nedávno označen jako zastaralý.[11] [66]

[10] Návrhový vzor Builder.
[11] Google se rozhodl pro budoucí přidání podpory verze 1.8 do svého současného Java kompilátoru a používání Jacka se doporučuje pouze do uvolnění této podpory

Podporu některých funkcí a API, které poskytuje Java ve verzi 1.8, lze získat použitím externích knihoven založených na backportingu.

Gradle Retrolambda API

Podporu lambda výrazů a některých dalších vylepšení Javy 1.8 lze vyřešit použitím Gradle pluginu Retrolambda. [67] Limitována je pouze možnostmi backportingu (proto nepodporuje např. defaultní metody a statické metody rozhraní).

ThreeTen Android Backport

Java 1.8 výrazně usnadňuje a zefektivňuje práci s datem a časem, která je v předchozích verzích Javy značně problematická.

Použití těchto funkcí umožňuje externí knihovna Threeten Android Backport [68], která implementuje knihovnu java.time, kterou nepodporuje ani kompilátor Jack.

Aby bylo možné využít všechny funkce této knihovny (např. zjištění aktuálního data), je potřeba nejprve zavolat její inicializační statickou metodu (nejlépe v konstruktoru vlastního potomka třídy *Application*).

6.5 JSON serializace a deserializace

Android Framework poskytuje pouze jednoduchou podporu textového formátu JSON, kdy je s ním zacházeno jako s obyčejnou datovou strukturou *HashMap* na principu klíče a hodnoty. Pro automatizovanou serializaci a deserializaci objektů je třeba použít jiné řešení.

Gson

Gson [69] je externí knihovna od Googlu, která umožňuje konvertovat POJO objekty do jejich JSON podoby a naopak.

Knihovna nabízí hned několik možností, jak pracovat s JSON formátem. Na ukázce níže je zobrazeno, jak probíhá základní serializace/deserializace.

```
public class Car {

    @SerializedName("Color")
    private String color;
    @SerializedName("Brand")
    private String brand;
    @SerializedName("Doors")
    private int doors;

    public Car(String color, String brand, int doors) {
        this.color = color;
        this.brand = brand;
        this.doors = doors;
    }

    public static String serialize(Car car) {
        Gson gson = new Gson();
        return gson.toJson(car);
    }

    public static Car deserialize(String json) {
        Gson gson = new Gson();
        return gson.fromJson(json, Car.class);
    }
}
```

Nejprve je tedy třeba vytvořit objekt třídy *Gson*, jehož metody se pro převod používají. Níže lze vidět příklad JSONu, jenž je výstupem deserializace pomocí Gsonu.

```
{"Color":"red","Brand":"Kia","Doors":"5"}
```

Pokud je potřeba, aby JSON obsahoval jiná jména, než jsou názvy atributů POJO objektu, je možné použít anotace s hodnotou obsahující jméno, které má být použito při serializaci/deserializace.

V případě, že je vyžadován nějaký speciální převod (např. použití konkrétního formátu u data), je potřeba vytvořit adaptér nebo alespoň serializér/deserializér, pomocí něhož se upraví způsob převodu. Poté se vytvoří *Gson* objekt prostřednictvím třídy *GsonBuilder*, která tento převodník zaregistruje.

6.6 Autentizace a autorizace na straně serveru

Aby byl přenos dat ze serveru a opačným směrem správně zabezpečený, je třeba, aby byla zajištěná dostatečná autorizace a autentizace, které zajistí, že s rozhraním na serveru opravdu komunikuje subjekt, který k tomu má oprávnění. Vzhledem k rozsahu daného tématu je v této kapitole rozebíráno pouze řešení použité v praktické části této bakalářské práce.

Autentizace
Autentizace probíhá metodou založenou na znalosti hesla. V případě úspěšné autentizace je do serverové databáze uloženo ID zařízení a server odešle zpět autentizační klíč, který má platnost 14 dní. Po tuto dobu není potřeba další autentizace.

Autorizace
Autorizace probíhá metodou vyhledání v seznamu povolených subjektů. Nejprve probíhá jako součást zaslání autentizačních údajů (na úrovni aplikace), kdy je zjišťováno, jestli seznam povolených aplikací obsahuje obdržené ID aplikace (v tomto případě obsahuje seznam pouze jednu položku).

Poté probíhá také v rámci akcí, a to hned na třech úrovních. Na nejvyšší úrovni je autorizace totožná s ověřením během zaslání autentizačních údajů. Dále probíhá autorizace na úrovni uživatele, kdy je ověřována pravost a platnost jeho autentizačního klíče. Nejnižší úroveň autorizace pak spočívá v ověření ID zařízení. Tím je zamezena komunikace se zařízeními, z nichž dosud neproběhla úspěšná autentizace.

ID aplikace
Během autorizace na serveru je potřeba ověřit, jestli webové rozhraní komunikuje s pravou mobilní aplikací. K tomuto účelu je používáno ID aplikace

Za ID aplikace je u Android platformy obecně považován název balíčku aplikace, který je používán také v praktické části této práce. Za unikátní lze toto ID považovat v rámci distribučního kanálu platformy, Google Play, který nepovoluje vložení aplikace s již existujícím názvem balíčku.

Vylepšení by mohlo představovat ID konkrétního sestavení aplikace. Vzhledem k tomu že není dost dobře možné, aby aplikace byla na všech zařízeních používána v nejnovější verzi, musely by být v serverové databázi uloženy ID všech minulých sestavení aplikace, aby nenastala chyba při ověřování žádné z minulých verzí.

ID zařízení
Při komunikaci s webovým rozhraním je potřeba ověřit, jestli webové rozhraní komunikuje s již dříve autentizovaným zařízením. K tomu je zapotřebí ID zařízení, které může být získáno několika způsoby (čerpáno z [70]):

- ID telefonního modulu – nejspolehlivější identifikátor, jestliže zařízení obsahuje telefonní modul. Toto ID ale logicky nelze použít k identifikaci zařízení bez telefonního modulu (nejčastěji tabletů obsahujících pouze WiFi modul). Dalším

problémem je perzistence tohoto identifikátoru, který zůstává stejný i po restartu zařízení do továrního nastavení a jeho předání jinému uživateli. Tento identifikátor navíc vyžaduje speciální oprávnění, jehož schvalování při instalaci může dráždit některé uživatele.

- MAC adresa – získána z WiFi nebo Bluetooth modulu. Tento identifikátor není doporučován z důvodu, že ne každé zařízení obsahuje WiFi nebo Bluetooth a navíc je u některých zařízení MAC adresa načtena pouze pokud je daný modul zapnutý. Navíc zde zůstává, stejně jako u ID telefonního modulu, problém perzistence
- Sériové číslo – od Android verze 2.3 je dostupné sériové číslo zařízení. Zde je hrozba chyby pouze u zařízení se starší verzí Androidu a při použití emulátorů, kde zařízení sériové číslo neobsahují. I u tohoto identifikátoru je problémem vysoká perzistence.
- ANDROID_ID – 64 bitový identifikátor generovaný a uložený při prvním bootování zařízení. Toto ID je resetováno během restartu do továrního nastavení. Pokud od prvního bootování nebyl k zařízení přidán Google účet, nastane chyba. Chyba může nastat také při použití zařízení s Android verzí nižší než 2.2. Navíc došlo v minulosti k minimálně jedné chybě v populárním modelu mobilního telefonu od významného výrobce, kdy všechny jeho instance měly stejné ANDROID_ID.

Získat stoprocentní identifikátor z každého zařízení tedy není možné. V praktické části této práce bylo nakonec k identifikaci zařízení zvoleno sériové číslo, u něhož lze předpokládat nejmenší problémy při zanedbání perzistence. Pokud by nastala chyba, je zvoleno ID telefonního modulu a jako krajní řešení může být použito ANDROID_ID.

6.7 Komunikace s rozhraním webové aplikace

Aplikace vyvíjená v praktické části této práce komunikuje s rozhraním webové aplikace, s jehož pomocí může data číst, aktualizovat či přidávat do serverové databáze. V této podkapitole bude tato komunikace rozebraná.

Formát

Existuje velké množství formátů pro přenos dat skrze internetovou síť. Pro komunikaci mezi mobilní aplikací a webovou stránkou je ovšem potřeba zvolit formát, který není závislý na použité platformě (je tedy dobře čitelný i snadno zapisovatelný).

K použití byl nakonec zvolen jednoduchý textový formát JSON, který je nejrozšířenějším multiplatformním formátem pro přenos dat a tyto kritéria splňuje. Formát není dále nijak zakódován.

Android Asynchronous Http Client

Pro zjednodušení kódu pro komunikaci s webovým rozhraním je vhodné použít některou z externích knihoven. Vzhledem k tomu, že komunikace probíhá pouze asynchronně, byla nakonec zvolena knihovna Android Asynchronous Http Client [71], která představuje asynchronního http klienta postaveného na Apache knihovnách, který je založen na zpětných voláních.

Všechny požadavky jsou vedeny mimo hlavní UI vlákno aplikace a veškerá logika zpětného volání je provedena ve stejném vlákně, ve kterém bylo zpětné volání vytvořeno pomocí Android Handleru při vytvoření požadavku (toto ošetřuje knihovna).

Knihovna také zvládá práci s JSON formátem, kdy odpovědi parsuje přímo do tříd *JSONObject* a *JSONArray* a zachytává je pomocí speciálního handleru.

Akce

Akce jsou speciální operace, které jsou v praktické části této práci používány pro přenos dat z/na server. Dělí se na:

- Select – slouží ke čtení dat ze serverové databáze. Aplikace si nemůže vybrat, ze kterých tabulek bude data číst a po úspěšné autorizaci vždy dostane pouze ta data, na která má daný uživatel oprávnění. Kromě autorizačních údajů je potřeba odeslat také ID uživatele.
- Insert – slouží k přidávání dat do serverové databáze. Odeslán může být pouze jeden objekt nebo i pole objektů, čímž je zajištěna efektivnost přenosu. V této akci je potřeba uvést název tabulky, do níž jsou data vkládána, a vkládaná data. Ze serveru je zpět odesláno pole objektů obsahujících mobilní ID a serverové ID, které je potřebné k případné budoucí aktualizaci dat.
- Update – slouží k aktualizaci dat v serverové databázi. Tato akce je velmi podobná přechozí akci s tím, že aktualizované objekty musí navíc obsahovat i serverové ID, které je potřeba k vyhledání objektů, které mají být na serveru aktualizovány. Tentokrát odpověď ze serveru obsahuje pouze potvrzení o úspěchu či neúspěchu operace.

Původně měla být implementována také akce Delete, ale z důvodu vazeb mezi tabulkami a složité signalizaci aplikaci, že data byla smazána, nakonec mazání dat na serverové databázi neprobíhá. Místo toho jsou data odebírána tak, že jsou skryta (např. uzavřením projektu nebo úkolu).

6.8 Práce v offline režimu

Aplikace není primárně určena pro práci offline, ale za splnění určitých podmínek se dá v tomto režimu nějaký čas používat. V této podkapitole bude rozebrána funkčnost aplikace v momentě, kdy nemá k dispozici připojení k internetu.

Interní databáze

Nejen pro práci v offline režimu, ale také pro pohodlnou a rychlou interaktivní práci s aplikací je potřeba ukládat data ze serveru offline na mobilní zařízení. Android pro tyto potřeby nabízí vestavěnou interní SQLite databázi, která je součástí Android Frameworku.

Její možnosti využívá také aplikace vyvíjená v praktické části této práce. Databázový model je téměř totožný s modelem databáze na serveru. Chybí jen některé tabulky, které slouží pro data, která mobilní aplikace nepotřebuje. Tabulky ve SQLite databázi jsou navíc doplněny o sloupec se serverovým ID (je potřeba nejen pro efektivní přenos dat mezi interní databází a serverovou databází, ale hlavně při aktualizaci dat v serverové databázi) a sloupec s logickou hodnotou udávající, jestli již proběhla synchronizace po poslední aktualizaci.

Přihlášení a práce s aplikací

Přihlášení probíhá pomocí ID přihlašovaného uživatele a jeho platného autentizačního klíče. Aplikace si ID uživatele (pomocí něhož posléze zjišťuje, jestli je v interní databázi uložen platný autentizační klíč tohoto uživatele) ukládá do nastavení v případě úspěšné autentizace na serveru. Naopak v případě odhlášení uživatele z aplikace je jeho ID z nastavení vymazáno.

Z toho plyne, že přihlášení v offline režimu je možné, pouze pokud se uživatel při posledním používání aplikace neodhlásil a jeho nejnovější autentizační klíč uložený v interní databázi je stále platný. Pokud jsou tyto podmínky splněny, může uživatel pracovat s aplikací bez omezení i bez připojení k internetu, dokud se neodhlásí. Vše je zobrazeno na vývojovém diagramu v příloze C.

Vytváření a aktualizace dat

Když uživatel v aplikaci během offline režimu vytvoří nová data nebo aktualizují již dříve vytvořená data, uloží se do interní databáze v zařízení a nahrají se na server až ve chvíli, kdy má uživatel k dispozici připojení k internetu a v menu stiskne položku *Synchronizovat data* (příp. poté co se odhlásí a opětovně přihlásí v režimu online).

Konflikty při synchronizaci

Pokud jsou data aktualizována během offline režimu, je pak potřeba při synchronizaci dat řešit některé konflikty:

1. Běžná synchronizace aktualizovaných dat – synchronizace probíhá tak, že jsou nejprve stažena aktuální data ze serveru. Pokud tyto data mají novější čas poslední aktualizace, je aktualizována interní databáze. Pokud tomu je naopak, jsou aktualizována data na serveru.
2. Synchronizace aktualizovaných projektů a úkolů – synchronizace probíhá jako v předchozím druhu konfliktu s výjimkou toho, když jsou projekty (úkoly) na serveru již uzavřené nebo z nich byl uživatel odebrán. V tom případě synchronizace neproběhne.

6.9 Objektově relační mapování

Android bohužel nenabízí žádné vestavěné řešení ORM. Přitom ORM ušetří opravdu velké množství kódu a činí komunikaci s databází velmi jednoduchou. Existuje velké množství externích knihoven zaměřených na tuto problematiku, ale žádná plně nevyhovovala zamýšlené implementaci usnadňující komunikaci jak s interní, tak se serverovou databází.

Proto byla vytvořena kompletně nová implementace ORM, která se nejvíce inspirovala v principech Vertabelo Mobile ORM. [72]

Vytvoření databáze

Vytvoření databáze bylo navrženo tak, aby SQL příkazy, které jsou jeho součástí, nemusely být zapisovány přímo do Java kódu, ale byly parsovány z externího SQL souboru. Ten tak může být vytvořen ve specializovaných nástrojích, díky čemuž je menší riziko chyby v příkazech.

Mapování databázových vlastností a operací na objekty

Aby bylo používání ORM co nejjednodušší, je potřeba všechny používané operace, jejich atributy a vlastnosti databázových tabulek převést na objekty. Díky tomu je pak menší pravděpodobnost chyby, než kdyby byly např. parametry do operací dodávány jako text, a navíc je jejich použití jednodušší.

Kromě komunikace s interní databází jsou sestavené operace používané také při přidávání a úpravě dat v serverové databázi.

DAO

Pro samotnou komunikaci s databází jsou používané tzv. DAO. Tyto objekty, které pracují s POJO představující jednotlivé řádky tabulek, jsou vytvářeny (ideálně by měly být generované) pro každou tabulku zvlášť.

Probíhá to tak, že je nejprve vytvořeno rozhraní, které v sobě definuje strukturu tabulky, ke které DAO přistupuje, a své speciální metody (např. pro spojování tabulek). Zároveň toto rozhraní také dědí několik dalších rozhraní, která obsahují základní metody. Následně je vytvořená implementace tohoto DAO rozhraní v podobě třídy, která zároveň dědí základní metody z abstraktní třídy *BaseDAO*.

Vytváření DAO rozhraní a jejich implementací je tak poměrně jednoduchá záležitost, která by neměla zabrat moc práce. Pro opakované použití tohoto ORM v budoucnu by ale bylo samozřejmě vhodné uvažovat o rozšíření o automatizované generování těchto rozhraní a jejich implementací během parsování SQL souboru a vytváření interní databáze.

MicroOrm

Aby nemuselo být zvlášť řešeno mapování POJO na *ContentValues*, se kterými pracuje Android Framework při komunikací se SQLite databází, je vhodné použít knihovnu MiroOrm. [73]

Tato knihovna funguje tak, že jsou v POJO třídách prostřednictvím anotací nad atributy mapována jejich jména. Díky tomu pak lze při zavolání metod třídy *MicroOrm* konvertovat POJO na *ContentValues* a zpět. Pro atributy referenčních typů je pak navíc potřeba vytvořit adaptéry a ty následně zaregistrovat do *MicroOrm* objektu použitého při převodu.

6.10 Automatizované testování

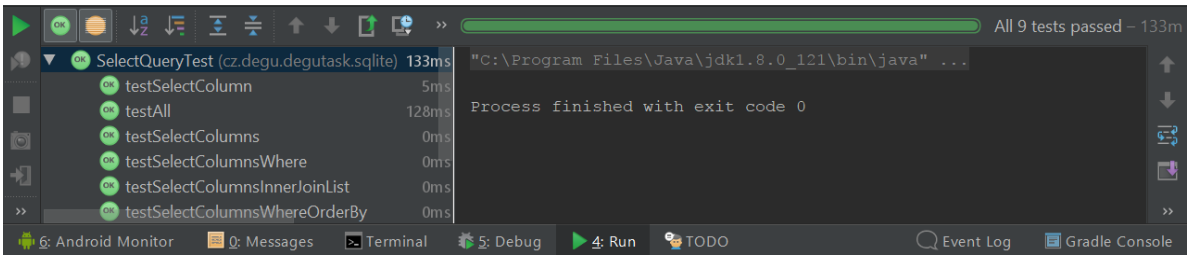
Android Framework nabízí při vývoji dva druhy automatizovaných testů. Jednotkové testování probíhá lokálně na JVM a používá se pro něj JUnit framework určený pro psaní jednotkových testů v Javě. Integrační testování probíhá pomocí tzv. instrumentálních testů, které vyžadují reálný nebo emulovaný Android OS, díky čemuž jsou mnohem pomalejší než testy jednotkové.

JUnit framework dokáže tvorbu testů velmi usnadnit. Namísto testování provedeném přímo v kódu metod aktivit jsou v balíčku *test* vytvořeny speciální třídy obsahující testovací metody (označené anotací *@Test*). V těchto metodách je pak správné fungování testovaných jednotek ověřeno pomocí metod třídy *Assert*, které porovnávají očekávanou hodnotu s aktuální hodnotou produkovanou testovacími jednotkami (zobrazeno v ukázce níže).

```
public class SelectQueryTest {
    @Test
    public void testSelectColumn() {
        String expected = "SELECT priorities._id FROM priorities";
        StringBuilder actual = new StringBuilder();
        SelectQuery.from(PrioritiesDAO.TABLE_STRUCTURE)
                .column(EXPECTED_NUMBER_COLUMN).build(actual);
        assertEquals(expected, actual.toString());
    }
}
```

Pro všechny testy lze také připravit vstupní data pomocí metody označené anotací *@Before*, která proběhne vždy před testovacími metody. Zároveň lze také provést určité úkony (např. zavření výstupního proudu) po skončení testovacích metod pomocí metody označené anotací *@After*, která proběhne vždy poté, co skončí poslední testovací metoda.

Samotné spuštění testů testovací třídy pak proběhne pouhým kliknutím pravým tlačítkem myši kdekoli uvnitř dané třídy a vybráním možnosti spuštění testů z kontextové nabídky nebo pomocí nastavené klávesové zkratky (ve výchozím stavu Ctrl+Shift+F10). Po skončení testů se zobrazí, které testy proběhly úspěšně a u kterých nastala chyba (zobrazeno na obrázku 19).



Obrázek 19: Výsledek JUnit testů v Android Studiu

V praktické části této práce bylo použito pouze několik jednotkových testů. Dva testy se zaměřovaly na správnou funkčnost Gson a MicroOrm typových adapterů a devět se jich zaměřovalo na ověření toho, jestli ORM správně sestavuje textové řetězce databázových operací.

Závěr

Hlavním cílem této bakalářské práce bylo navrhnout a implementovat mobilní aplikaci, která bude schopná vizualizovat a manipulovat s daty Task systému. Na základě těchto požadavků byla vytvořena mobilní aplikace Degu Task pro operační systém Android.

Dále byla představena mobilní platforma Android a možnosti vývoje aplikací pro tuto platformu s důrazem na vývoj v programovacím jazyce Java. Spolu s nimi byly také představeny vývojové nástroje, které jsou pro tento vývoj ideální.

Současně byly rozebrány možnosti použití jednotlivých architektur v rámci Android Frameworku spolu s detaily implementace MVP architektury. Nakonec byly rozebrány problémy, které jsou součástí tvorby mobilních aplikací, spolu s problémy, které jsou způsobeny implementací samotného Android Frameworku, a byly nabídnuty možnosti jejich řešení.

Tato práce byla součástí týmového projektu, v rámci kterého byla vytvořena webová a mobilní aplikace. Kromě tvorby mobilní aplikace bylo součástí této práce také podílení se na tvorbě datového modelu serverové databáze a značná úprava webového rozhraní vytvořeného v programovacím jazyce PHP, se kterým mobilní aplikace komunikuje.

Funkčnost mobilní aplikace byla ověřena pomocí několika automatizovaných jednotkových testů. Dále testování probíhalo na dvou reálných a několika emulovaných zařízeních. V rámci testování vyvstala potřeba úpravy datových modelů databází (interní i serverové) a webového rozhraní, která byla následně provedena. Do budoucna by určitě stálo za úvahu rozšířit procento pokrytí kódu automatizovanými testy.

Mobilní aplikace byla vyvíjena tak, aby byla snadno rozšiřitelná a její možné budoucí úpravy nevyžadovaly velké zásahy do kódu. Do budoucna se proto nabízí např. přidání push notifikací, možnost prohledávat aplikaci, optimalizace aplikace pro tablety pomocí lepšího využití fragmentů, šifrování komunikace s webovým rozhraním a převedení celé odpovědnosti za rozhodování při synchronizaci do webového rozhraní.

Seznam použitých zdrojů

[1] Number of Android applications. In: *AppBrain* [online]. AppBrain, 12. dubna 2017 [cit. 2017-04-05]. Dostupné z: https://www.appbrain.com/stats/number-of-android-apps

[2] Android. In: *Britannica Academic* [online]. Encyclopædia Britannica, 30. prosince 2008 [cit. 2017-04-05]. Dostupné z: http://academic.eb.com/EBchecked/topic/1483582/Android

[3] CHLUP, Vladimír. ART vs. Dalvik aneb bitva uvnitř Androidu. *Mobilizujeme* [online]. 13. listopadu 2013 [cit. 2017-04-15]. Dostupné z: https://mobilizujeme.cz/clanky/art-vs-dalvik-aneb-bitva-uvnitr-androidu

[4] HUB, Miloslav. Autentizace v přímém bankovnictví: Význam autentizace v přímém bankovnictví. *Scientific papers of the University of Pardubice: Series D Faculty of Economics and Administration* [online]. Pardubice: Ústav systémového inženýrství a informatiky, FES, Univerzita Pardubice, 2003, 8 roč., s. 39-43 [cit. 2017-04-14]. ISSN 1211-555X. Dostupné z: http://dspace.upce.cz/bitstream/handle/10195/32263/CL404.pdf

[5] FLAMÍK, Martin. *Bezpečnost v elektronickém bankovnictví*. Pardubice, 2010. Bakalářská práce. Univerzita Pardubice. Fakulta ekonomicko-správní. Ústav systémového inženýrství a informatiky.

[6] STROUD, Forrest. Backport. *Webopedia* [online]. ©2017 [cit. 2017-04-12]. Dostupné z: http://www.webopedia.com/TERM/B/backport.html

[7] ŽAMPACH, Jan. *Volání nativního kódu z javovské aplikace pomocí JNI*. Pardubice, 2011. Bakalářská práce. Univerzita Pardubice. Fakulta elektrotechniky a informatiky.

[8] Jihočeská univerzita v Českých Budějovicích. Pedagogická fakulta. Nastavení BIOSu počítače. *Pf.jcu.cz* [online]. 2002 [cit. 2017-04-05]. Dostupné z: http://www.pf.jcu.cz/stru/katedry/fyzika/prof/Tesar/diplomky/pruvodce_hw/komponenty/zakladni/deska/bios.htm

[9] ZAPLETAL, Lukáš. Android bez Javy dál válcuje konkurenci. *LinuxEXPRES* [online]. CCB, spol. s.r.o., 20. srpna 2010 [cit. 2017-04-07]. ISSN 1801-3996. Dostupné z: https://www.linuxexpres.cz/aktuality/android-bez-javy-dal-valcuje-konkurenci

[10] VÁVRŮ, Jiří a Miroslav UJBÁNYAI. *Programujeme pro Android*. 2., rozšíř. vyd. Praha: Grada, 2013. Průvodce (Grada). ISBN 978-80-247-4863-4.

[11] Dependency Injection. *Nette Framework* [online]. ©2008-2017 [cit. 2017-04-05]. Dostupné z: https://doc.nette.org/cs/2.4/dependency-injection

[12] BRCHAŇ, Jan. *Tvorba moderních aplikací v jazyce Groovy*. Brno, 2005. Bakalářská práce. Masarykova Univerzita. Fakulta informatiky.

[13] KOLLARIKOVÁ, Kateřina. *Hardwarové principy virtualizace*. Zlín, 2011. Bakalářská práce. Univerzita Tomáše Bati ve Zlíně. Fakulta aplikované informatiky.

[14] BREJCHA, Jiří. Podpora hardwarové virtualizace. *Jiří Brejcha* [online]. 5. ledna 2010 [cit. 2017-04-05]. Dostupné z: http://www.jiribrejcha.net/2010/01/podpora-hardwarove-virtualizace-intel-vt-amd-v

[15] Smartphony. *Mobilmania.cz* [online]. CN Invest, ©2017 [cit. 2017-04-15]. ISSN 1213-8991. Dostupné z: http://www.mobilmania.cz/smartphony/sc-319/default.aspx

[16] BROŽEK, Josef. *Ekonomicko správní systém pro malé a střední společnosti*. Pardubice, 2010. Bakalářská práce. Univerzita Pardubice. Fakulta elektrotechniky a informatiky.

[17] Instalace Javy pro vývoj programů. *Java.vse.cz* [online]. 28. září 2016 [cit. 2017-04-05]. Dostupné z: https://java.vse.cz/Java/Instalace

[18] Úvod do JSON. *Json.org* [online]. 200-? [cit. 2017-04-14]. Dostupné z: http://www.json.org/json-cz.htm

[19] KONEČNÝ, Pavel. *Multiplatformní správce souborů v jazyce Java*. Pardubice, 2011. Bakalářská práce. Univerzita Pardubice. Fakulta elektrotechniky a informatiky.

[20] History of Linux explained. *Everything.explained.today* [online]. ©2009-2016 [cit. 2017-04-05]. Dostupné z: http://everything.explained.today/History_of_Linux

[21] HELEŠIC, Martin. *Návrh modelové topologie s využitím L2 a L3 switchu*. Pardubice, 2014. Bakalářská práce. Univerzita Pardubice. Fakulta elektrotechniky a informatiky.

[22] Mobile Application (Mobile App). In: *Technopedia* [online]. Technopedia Inc., ©2017 [cit. 2017-04-18]. Dostupné z: https://www.techopedia.com/definition/2953/mobile-application-mobile-app

[23] Mobile application development. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 23. března 2006, aktualizováno 6. dubna 2017 [cit. 2017-04-18]. Dostupné z: https://en.wikipedia.org/wiki/Mobile_application_development

[24] ARNOŠT, Pavel. Co je to Open Source Software. *Root.cz* [online]. Internet Info, s.r.o., 22. srpna 2001 [cit. 2017-04-05]. ISSN 1212-8309. Dostupné z: https://www.root.cz/clanky/co-je-to-open-source-software

[25] TORRES, Alexandre, Renata GALANTE, Marcelo S. PIMENTA a Alexandre Jonatan B. MARTINS. Twenty years of object-relational mapping: A survey on patterns, solutions, and their implications on application design. *Information and software technology* [online]. listopad 2017, roč. 82, s. 1-18 [cit. 2017-04-22]. ISSN 0950-5849. Dostupné z: http://www.sciencedirect.com/science/article/pii/S0950584916301859

[26] MITRA, Kumar Vivek. What is the pojo class? *Stackoverflow.com* [online]. 20 listopadu 2012 [cit. 2017-04-12]. Dostupné z: http://stackoverflow.com/questions/12517905/what-is-java-pojo-class-java-bean-normal-class

[27] Programming Language. In: *Technopedia* [online]. Technopedia Inc., ©2017 [cit. 2017-04-12]. Dostupné z: https://www.techopedia.com/definition/24815/programming-language

[28] ŽALUD, Michal. *Tvorba WWW aplikace s využitím relační databáze pro zimní sporty*. Pardubice, 2009. Bakalářská práce. Univerzita Pardubice. Fakulta elektrotechniky a informatiky.

[29] Offline. *It-slovnik.cz* [online]. ©2008-2017 [cit. 2017-04-05]. Dostupné z: http://it-slovnik.cz/pojem/offline

[30] ZEMEK, Petr. Chyby v návrhu: Singleton. *Cs-blog.petrzemek.net* [online]. 6. července 2013 [cit. 2017-04-05]. Dostupné z: https://cs-blog.petrzemek.net/2013-07-06-chyby-v-navrhu-singleton

[31] SQL. In: *Britannica Academic* [online]. Encyclopædia Britannica, 15. ledna 2009 [cit. 2017-04-05]. Dostupné z: http://academic.eb.com/levels/collegiate/article/SQL/438617#

[32] About SQLite. *Sqlite.org* [online]. 28. března 2017 [cit. 2017-04-05]. Dostupné z: https://www.sqlite.org/about.html

[33] Using Databases. *Android Developers* [online]. 2016? [cit. 2017-04-05]. Dostupné z: https://developer.android.com/guide/topics/data/data-storage.html#db

[34] Špagetový kód. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 14. května 2014, aktualizováno 5. října 2016 [cit. 2017-04-15]. Dostupné z: https://cs.wikipedia.org/wiki/Špagetový_kód

[35] BÍLEK, Petr. Vývojová prostředí a překladače. *Sallyx.org* [online]. 2. října 2016 [cit. 2017-04-05]. Dostupné z: http://www.sallyx.org/sally/c/vyvojova-prostredi.php

[36] CARBONNELLE, Pierre. TOP IDE index. *Pypl.github.io* [online]. 2017 [cit. 2017-04-05]. Dostupné z: http://pypl.github.io/IDE.html

[37] HLAVÍK, Jiří. 2. díl – Android programování – Vývojové prostředí. *Itnetwork.cz* [online]. 2015 [cit. 2017-04-05]. Dostupné z: http://www.itnetwork.cz/java/android/tutorial-programovani-pro-android-v-jave-vyvojove-prostredi

[38] SARBJEET, Singh. Software Testing. *International journal of advanced research in computer science* [online]. [b.j.], 1. října 2010, roč. 1, č. 3, s. 403-406 [cit. 2017-04-21]. ISSN 0976-5697. Dostupné z: http://search.proquest.com/docview/1443701695/F3BFDD3F196649F6PQ/1?accountid=17239

[39] HLAVA, Tomáš. Fáze a úrovně provádění testů. *Testování softwaru: Portál zabývající se problematikou ověřování kvality software. Manuální i automatizované testování.* [online]. 21. srpna 2011 [cit. 2017-04-21]. Dostupné z: http://testovanisoftwaru.cz/tag/akceptacni-testovani

[40] HARMAN, Mark. Why Source Code Analysis and Manipulation Will Always Be Important. In: SCAM 2010: *International Working Conference on Source Code Analyisis and Manipulation*, Timișoara 12.-13. listopadu 2010 [online]. Timișoara: [b.j.], 2010 [cit. 2017-04-12]. Dostupné z: http://www0.cs.ucl.ac.uk/staff/M.Harman/scam10.pdf

[41] SIMS, Gary. What is a kernel – Gary explains. *Android Authority* [online]. 30. března 2016 [cit. 2017-04-05]. Dostupné z: http://www.androidauthority.com/what-is-a-kernel-gary-explains-681744

[42] SMYTH, Neil. *Android Studio 2.3 Development Essentials: Android 7 Edition*. 1. [b.m.]: Payload Media, 2017. ISBN 978-1544275437.

[43] KRABÁČ, Tomáš. *Historie ČVUT: Android mobilní aplikace*. Praha, 2016. Diplomová práce. České vysoké učení technické. Fakulta informačních technologií.

[44] Activity: Activity Lifecycle. *Android Developers* [online]. 23. října 2008 [cit. 2017-04-05]. Dostupné z: https://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle

[45] Fragments. *Android Developers* [online]. 22. března 2011 [cit. 2017-04-05]. Dostupné z: https://developer.android.com/guide/components/fragments.html

[46] Support Library. *Android Developers* [online]. 2016? [cit. 2017-04-05]. Dostupné z: https://developer.android.com/topic/libraries/support-library/index.html

[47] MÁDR, Vojtěch. Xamarin: Představujeme nástroj pro multiplatformní vývoj mobilních aplikací (díl 1). *eMan* [online]. 29. února 2016 [cit. 2017-04-05]. Dostupné z: https://www.eman.cz/blog/xamarin-predstavujeme-nastroj-pro-multiplatformni-vyvoj-mobilnich-aplikaci-dil-1

[48] SIMS, Gary. I want to develop Android Apps: What language should I learnt. *Android Authority* [online]. 29. července 2016 [cit. 2017-04-05]. Dostupné z: http://www.androidauthority.com/want-develop-android-apps-languages-learn-391008

[49] DUCROHET, Xavier, Tor NORBYE a Katherine CHOU. Android Studio: An IDE built for Android. *Android Developers Blog* [online]. 15. května 2013 [cit. 2017-04-05]. Dostupné z: https://android-developers.googleblog.com/2013/05/android-studio-ide-built-for-android.html

[50] EASON, Jamal. Support Ended for Eclipse Android Developer Tools. *Android Developers Blog* [online]. 2. listopadu 2016 [cit. 2017-04-05]. Dostupné z: http://android-developers.blogspot.cz/2016/11/support-ended-for-eclipse-android.html

[51] What is Gradle in Android Studio? *Stackoverflow.com* [online]. 16. listopadu 2015 [cit. 2017-04-05]. Dostupné z: http://stackoverflow.com/questions/16754643/what-is-gradle-in-android-studio

[52] Gradle Overview. *www.tutorialspoint.com* [online]. ©2017 [cit. 2017-04-05]. Dostupné z: https://www.tutorialspoint.com/gradle/gradle_overview.htm

[53] Use Java 8 Language Features. *Android Developers* [online]. 2017 [cit. 2017-04-05]. Dostupné z: https://developer.android.com/guide/platform/j8-jack.html

[54] CARBALLO, Iván. Android Application Architecture. *ribot labs* [online]. 1. prosince 2015 [cit. 2017-04-05]. Dostupné z: https://labs.ribot.co.uk/android-application-architecture-8b6e34acda65

[55] KARPOUZIS, Thanos. Android Architecture: A simple guide for MVC, MVP and MVVM on Android projects. *AndroidPub* [online]. 24. srpna 2015 [cit. 2017-04-13]. Dostupné z: https://android.jlelse.eu/android-architecture-2f12e1c7d4db

[56] RIPPLE, Chris. Applying MVP in Android. *GoXuni* [online]. 26. května 2016 [cit. 2017-04-05]. Dostupné z: http://www.goxuni.com/673883-applying-mvp-in-android

[57] MIKHEEV, Konstantin. Nucleus. *GitHub* [software]. 2016 [cit. 2017-04-05]. Dostupné z: https://github.com/konmik/nucleus

[58] BIRCH, Joe. Approaching Android with MVVM. *ribot labs* [online]. 21. září 2015 [cit. 2017-04-05]. Dostupné z: https://labs.ribot.co.uk/approaching-android-with-mvvm-8ceec02d5442

[59] IVANOV, Vladimir. Going with MVVM on Android via Data Binding. *Cases.azoft.com* [online]. 1. prosince 2015 [cit. 2017-04-05]. Dostupné z: http://cases.azoft.com/mvvm-android-data-binding

[60] BELTRAN, Miquel. 4 Reasons I'm Not Using Android Data Binding. *Medium* [online]. 29. září 2016 [cit. 2017-04-05]. Dostupné z: https://medium.com/@Miqubel/4-reasons-im-not-using-android-data-binding-e62127c2650c

[61] Java's Dependency (Mis)Management: How Maven and Gradle Cope. *Implements Blog* [online]. 10. května 2014 [cit. 2017-04-20]. Dostupné z: https://implementsblog.com/2014/05/10/javas-dependency-mismanagement-how-maven-and-gradle-cope/

[62] JAIN, Anshul. Resolving Conflicts in android gradle dependencies. *Mindorks: All About Android Application Development, iOS App Development, Mobile App Development, Machine Learning, Software Development Best Practices* [online]. 7. ledna 2017 [cit. 2017-04-13]. Dostupné z: https://blog.mindorks.com/avoiding-conflicts-in-android-gradle-dependencies-28e4200ca235

[63] Known Issues. *Android Studio Project Site* [online]. 2017? [cit. 2017-04-20]. Dostupné z: http://tools.android.com/knownissues

[64] WHARTON, Jack. Butter Knife. *GitHub* [software]. 2013 [cit. 2017-04-05]. Dostupné z: http://jakewharton.github.io/butterknife

[65] SHAPIRO, Ron. Dagger 2. *GitHub* [software]. 2016 [cit. 2017-04-05]. Dostupné z: https://github.com/google/dagger

[66] LAU, James. Future of Java 8 Language Feature Support on Android. *Android Developers Blog* [online]. 14. března 2017 [cit. 2017-04-09]. Dostupné z: https://android-developers.googleblog.com/2017/03/future-of-java-8-language-feature.html

[67] TATARKA, Evan. Gradle Retrolambda Plugin. *GitHub* [software]. 2013 [cit. 2017-04-05]. https://github.com/evant/gradle-retrolambda

[68] WHARTON, Jack. ThreeTenABP. *GitHub* [software]. 2015 [cit. 2017-04-05]. Dostupné z: https://github.com/JakeWharton/ThreeTenABP

[69] GOOGLE INC. google-gson. *GitHub* [software]. 2013 [cit. 2017-04-05]. Dostupné z: https://github.com/google/gson

[70] BRAY, Tim. Identifying App Installations. *Android Developers Blog* [online]. 30. března 2011 [cit. 2017-04-24]. Dostupné z: https://android-developers.googleblog.com/2011/03/identifying-app-installations.html

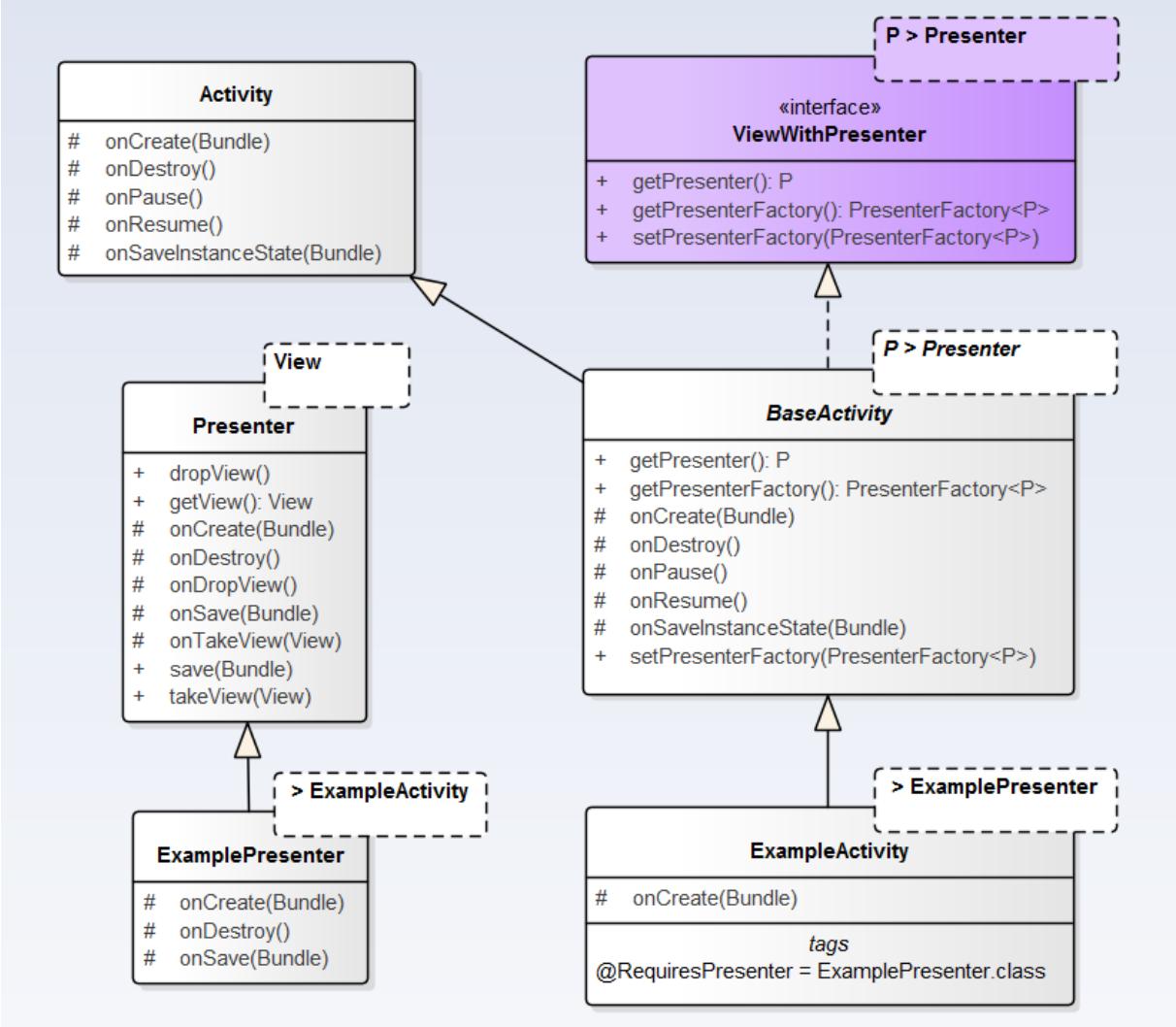
[71] SMITH, James. Android Asynchronous Http Client. *GitHub* [software]. 2013 [cit. 2017-04-22]. Dostupné z: https://github.com/loopj/android-async-http

[72] *Vertabelo Mobile ORM* [online]. Vertabelo sp. z o.o, ©2015-2016 [cit. 2017-04-22]. Dostupné z: http://mobile-orm.vertabelo.com

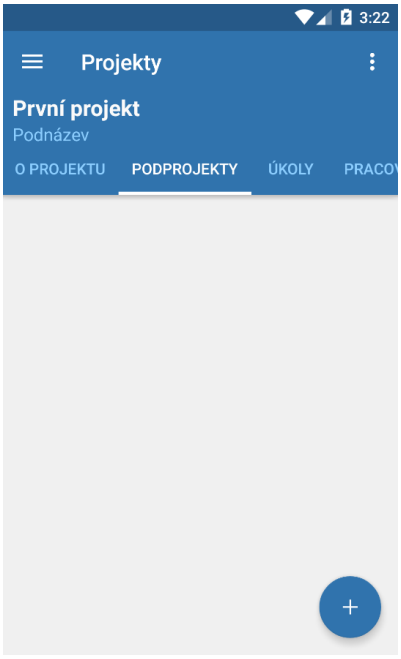
[73] Chalupski, Jerzy. MicroOrm. *GitHub* [software]. 2013 [cit. 2017-04-22]. Dostupné z: https://github.com/chalup/microorm

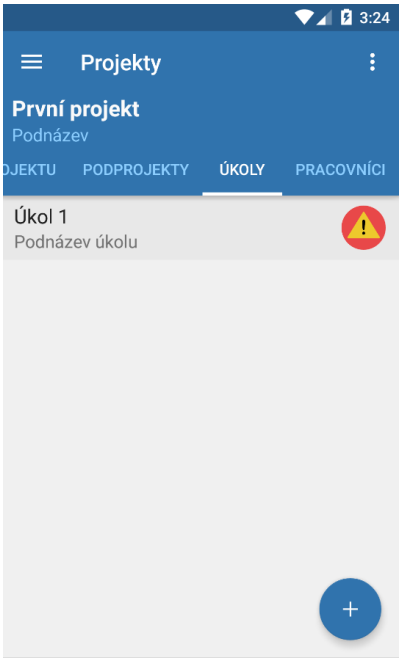
Seznam příloh

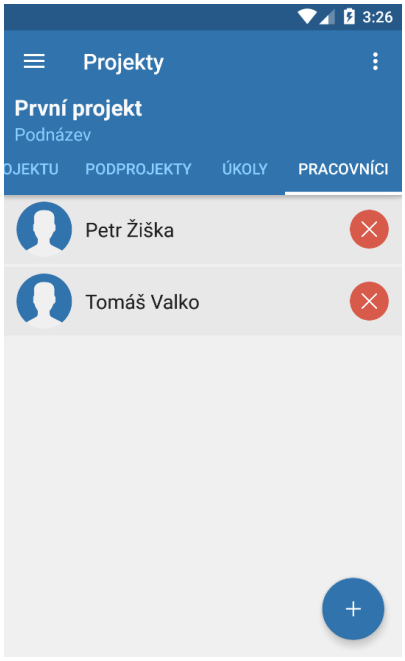
Příloha A – *Příklad propojení vrstev View a Presenter v MVP*

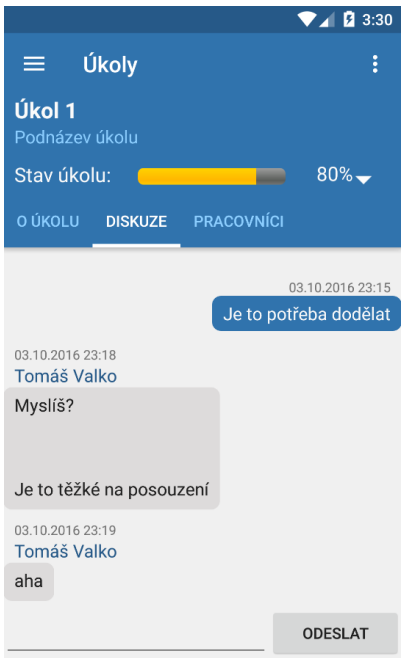


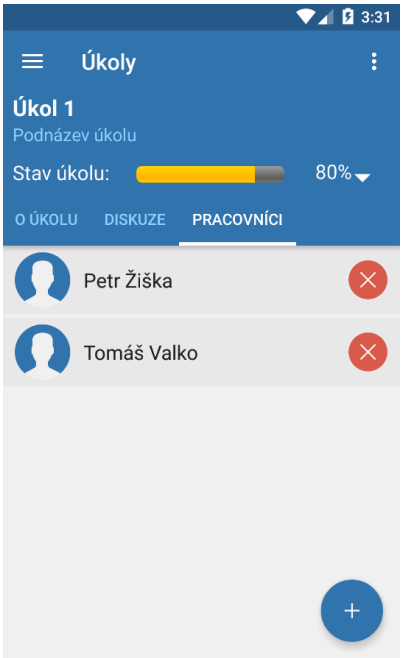
Příloha B – *Další obrazovky aplikace z praktické části práce*



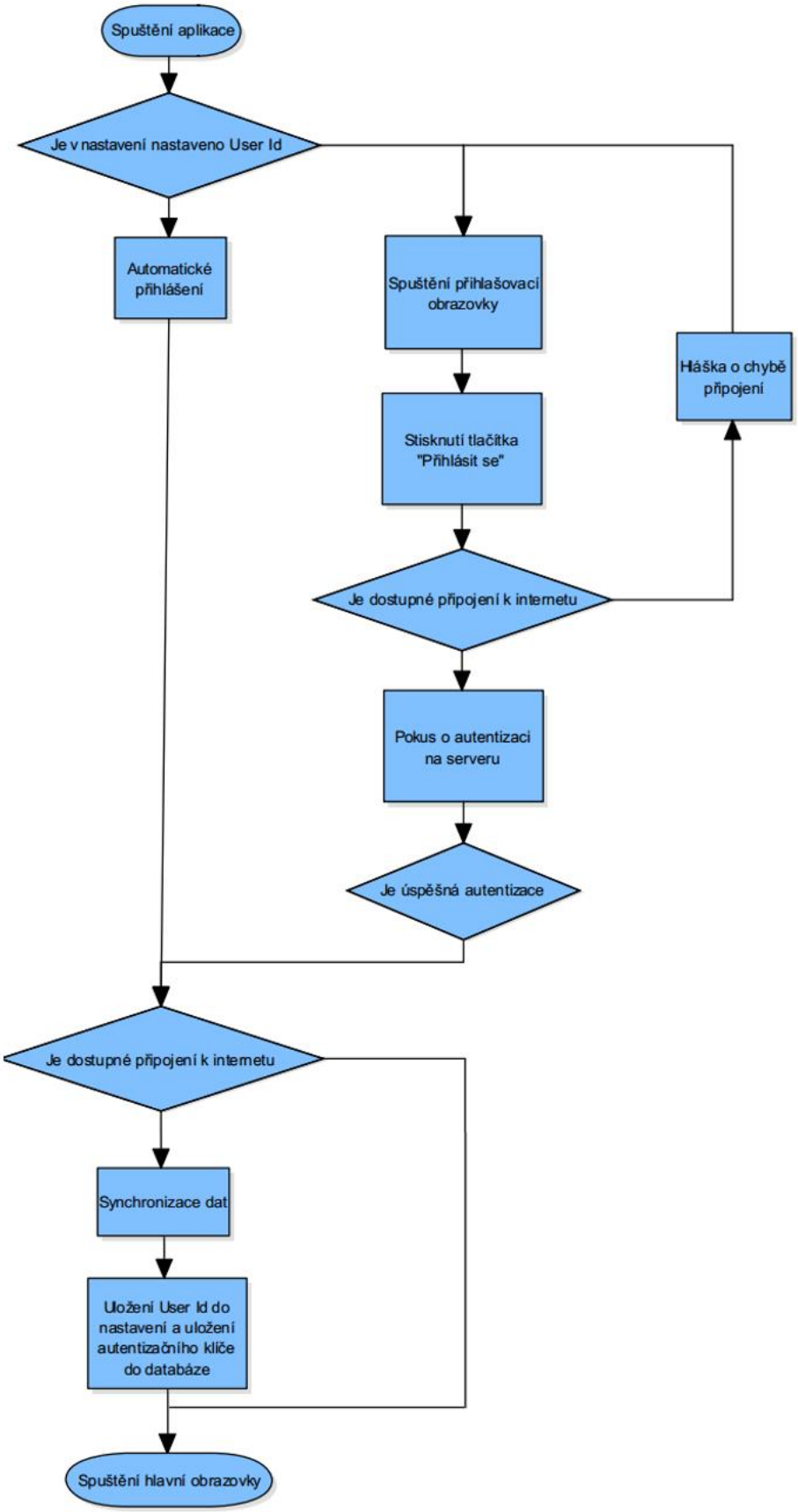








Příloha C – *Vývojový diagram spuštění aplikace*



Příloha D – *Obsah volně loženého CD*

K práci je volně loženo CD obsahující její praktickou část a tento text ve formátu PDF. Níže je uveden strom složek a souborů, které jsou na něm přítomny.

- *android projekt*
  - *app*……... složka se soubory aplikačního modulu
  - *build*…… soubory vytvořenými při sestavování aplikace
  - *build.gradle*...……..……...…… konfigurace gradle skriptu
  - *DeguTask.iml*……..…….……. soubor pro spuštění Android projektu
  - …
  - …
  - …
- *apk*
  - *DeguTask.apk*……... APK soubor konfigurovaný pro localhost server
- *db*
  - *server-databaze.sql*…….……. soubor pro vytvoření serverové databáze
- *text*
  - *ctime.txt*……...……. popis obsahu CD
  - *BP_Valko_Tomas.pdf*…… text práce ve formátu PDF
- *web*
  - *JsonPresenter.php*……...…….…… rozhraní webové aplikace pro komunikaci