VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ANDROID APLIKACE PRO GIT S PODPOROU GIT-LFS A GIT-ANNEX

THESIS TITLE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **PETR MAREK**
AUTHOR

VEDOUCÍ PRÁCE — **RNDr. MAREK RYCHLÝ, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Marek Petr**
Program: Informační technologie
Název: **Android aplikace pro Git s podporou git-lfs a git-annex**
Android Application for Git with git-lfs and git-annex Support
Kategorie: Uživatelská rozhraní

Zadání:

1. Seznamte se s Git a jeho rozšířeními git-lfs a git-annex. Prozkoumejte existující aplikace (nejen pro operační systém Android) pro ovládání repositářů Git, git-lfs a git-annex, soustřeďte se zejména na aplikace s grafickým uživatelským rozhraním.
2. Navrhněte aplikaci pro operační systém Android, která umožní ovládat Git repositáře s podporou git-lfs a git-annex. Zaměřte se na uživatelskou přívětivost a minimalizaci velikosti repositářů ("shallow clone", ignorování a odstraňování nepotřebných souborů, aj.). Řešte také problémy kompatibility s úložištěm (např. podpora symbolických odkazů).
3. Po konzultaci s vedoucím aplikaci pro operační systém Android implementujte.
4. Řešení otestujte, vyhodnoťte a diskutujte výsledky. Výsledný software publikujte jako open-source.

Literatura:

- Ľuboslav Lacko. Vývoj aplikací pro Android. Computer Press, Brno, 2015. ISBN 978-80-251-4347-6.
- Scott Chacon. Pro Git. CZ.NIC, Praha, 2009. ISBN 978-80-904248-1-4

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2 a započatá práce na bodu 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Rychlý Marek, RNDr., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 14. května 2020
Datum schválení: 21. října 2019

Abstrakt

Cílem práce je návrh a vývoj aplikace pro zařízení systému android. Tato aplikace umožňuje použití programu Git a jeho rozšíření Git LFS a Git annex, s podporou uživatelského rozhraní. Poslouží zejména vývojářům k usnadnění práce s GIT a velkými soubory. Aplikace tedy předpokládá, že ji budou používat uživatelé obeznámení s tímto verzovacím systémem. Uživatelské rozhraní je tak maximálně transparentní za účelem efektivního řešení problémů vznikajících při použití Git.

Abstract

This thesis aims to design and develop an android application. The application's purpose is to serve Git and its extensions Git LFS and Git annex with the aid of user interface. Its target audience is mainly developers looking for easier work with Git and large files on an android system. Its user interface is therefore designed to provide a transparent environment, which makes collision resolving easier.

Klíčová slova

android, android-studio, git, git-lfs, git-annex, lfs, annex, asynchronní programování, vlákno, proces, room, databáze, křížová kompilace

Keywords

android, android-studio, git, git-lfs, git-annex, lfs, annex, async-task, thread, process, room, database, cross-compilation

Citace

MAREK, Petr. *Android aplikace pro Git s podporou*
git-lfs a git-annex. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce RNDr. Marek Rychlý, Ph.D.

Android aplikace pro Git s podporou git-lfs a git-annex

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana X... Další informace mi poskytli... Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Petr Marek
11. dubna 2020

Poděkování

V této sekci je možno uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc (externí zadavatel, konzultant apod.).

Obsah

Kapitola 1

Úvod

Trend poslední doby byl a stále je neustálé zvětšování obrazovek zařízení i jejich výkonu. Dostali jsme se již do takové fáze, že je tyto zařízení možné využívat obdobně jako klasické počítače a tak je jejich využití pro synchronizaci souborů na snadě. Vyvíjená aplikace poskytuje systém, který může každý uživatel využít pro svůj účel a svým způsobem. Nejčastěji je git využíván programátory pro verzování souborů vyvíjených programů. Rozšíření *Git LFS* a *Git Annex* pak pro přidání velkých souborů do těchto repozitářů. Jejich využitím se dosáhne efektivity využití prostoru zařízení a současně plného využití systému git. Tyto rozšíření lze ale využívat i samostatně. Například pro ukládání videí nebo i jiných velkých souborů na externí úložiště pro pozdější synchronizaci mezi různými zařízeními různých systémů.

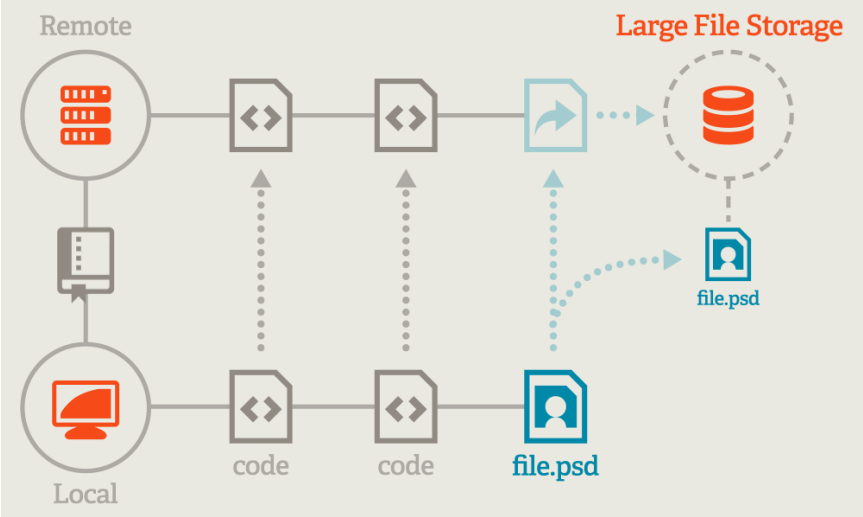
Cílem práce je navrhnout, implementovat a otestovat aplikaci určenou pro operační systém Android. Tato aplikace bude uživateli zprostředkovávat git pro tato zařízení formou přívětivého grafického rozhraní. Dále bude implementovat rozšíření *Git LFS* a *Git Annex* pro práci s velkými soubory. Aplikace je určena zejména vývojářům a jiným pokročilým uživatelům. Je tedy navržena jako maximálně transparentní při zachování prvků jednoduchého ovládání mobilních zařízení.

1.1 Git

Git slouží zejména programátorům k verzování jejich práce, popřípadě jejího sdílení s ostatními členy týmu. Nicméně jeho využití je široké a to zejména při využití rozšíření *Git LFS*[1] nebo *Git Annex*[2], která se zaměřují na práci s velkými soubory.

1.2 Git LFS

git Large File Storage (LFS) nahrazuje velké soubory v repozitářích ukazateli. Samotné soubory jsou pak uloženy na vzdáleném serveru. Tento systém tedy slouží k efektivnímu uložení velkých souborů v git. Jedná se například o video záznamy, zvukové stopy, datasety a jiné velké binární soubory.



Obrázek 1.1: Architektura *Git LFS*1

1.3 Git Annex

Git annex slouží k indexaci, synchronizaci a sdílení souborů mezi více úložišti nezávisle na komerční službě nebo centrálním serveru[1]. V repozitáři je uložen symbolický odkaz na klíč, který je hash daného souboru. Samotný soubor je pak uložen v adresáři *.git/annex/*. Při změně souboru se mění jen jeho hash a aktualizuje symbolický odkaz. Tímto způsobem je zajištěno šetření místa, jelikož samotný soubor je v repozitáři uložen maximálně jednou.

[1] https://git-lfs.github.com/
[2] https://git-annex.branchable.com/

Kapitola 2

Specifikace řešení

Hlavním cílem aplikace je nabídnout uživatelům řešení pro verzování a synchronizaci velkých souborů jejich git repozitářů na zařízeních systému Android. Priorita tedy bude kladena spíše na funkčnost než perfektní uživatelské rozhraní.

2.1 Funkce aplikace

Aplikace bude mít dvě základní funkce. Jsou jimi správa repozitářů a provádění příkazů gitu. Uživatel bude moci spravovat git repozitáře následujícím způsobem. K přidání nového repozitáře bude mít tři možnosti. Buď může vybrat adresář s daným repozitářem z místních souborů zařízení, inicializovat úplně nový ve zvolené složce nebo klonovat vzdálený. Při otevření repozitáře nad ním může provádět základní příkazy gitu a také některé vybrané funkce již zmíněných rozšíření.

2.2 Cílová skupina

Cílovou skupinou jsou především programátoři nebo i jiní technicky zdatní uživatelé. Ti aplikaci využijí nejčastěji pro prohlížení jejich repozitářů, ale mohou je také jakkoliv měnit a pracovat na nich třeba i z veřejné dopravy. Git annex využijí například při procházení souborů uložených na více fyzických úložištích. Všechny takto sledované soubory budou přehledně zobrazeny v repozitáři a uživatel tak v danou chvíli ani nemusí přemýšlet, kde jsou právě uložené. Jelikož git annex používá jednoduchý formát git repozitáře, je navíc garantováno, že tyto data budou v budoucnu dostupná i bez jeho použití.

2.3 Průzkum existujících řešení

Během průzkumu již existující aplikací jsem se zaměřil jak na aplikace operačního systému Android, tak na desktopové operační systémy Linux a Windows.

2.3.1 Android

Pro operační systémy Android je trh s řešeními gitu velice omezený. Existují zde několik málo aplikací s podporou pouze pro čtení repozitáře ale i takové, které zvládají i ostatní základní příkazy gitu. Jejich popis a mé postřehy z nich se dočtete na následujících řádkách.

MGit[1]

Za zmínku z nich stojí MGit. Bohužel neposkytuje podporu pro *Git LFS* ani *Git Annex*. K implementaci funkcí gitu využívá knihovnu *JGit*. Ta sice v aktuální verzi podporuje *Git LFS*, ale v té, kterou aplikace využívá ji ještě nemá. K jejím přednostem patří otevřený kód a velice intuitivní ovládání. Úvodní obrazovka aplikace se seznam repozitářů. Po kliknutí na některý se zobrazí obrazovka s jeho detaily. Nalezneme zde prohlížeč jeho souborů, log a status repozitáře. Na této obrazovce se také nachází základní ovládací prvek gitu aplikace. Jím je drawer, který se vysunuje z pravé strany obrazovky. V něm jsou obsaženy všechny poskytované příkazy gitu. Tedy jeho užívání není při porozumění obecného užívání gitu nijak náročné. Tato aplikace má integrovaný prohlížeč souborů i jejich editování. Ovšem tento editor není dokonalý. Špatně se v něm posouvá kurzor a navíc nemaže konce řádků. Práce s ním je tedy spíše na obtíž. Naštěstí zde autoři přidali i možnost zvolení vlastního editoru z nainstalovaných aplikací.

Pocket Git[2]

Dále existuje například aplikace *Pocket Git*. Ta je placená a její kód není veřejně přístupný. Využívá integrovaného správce souborů, ale editor již nechává plně na jiných aplikacích. *Pocket Git* má na první pohled přehlednější uživatelské rozhraní. Jednotlivé příkazy gitu rozděluje do různých kategorií a vedle souborů přidává ikonku o jeho stavu. Nicméně *Add* a *Commit* jsou natolik integrované do prohlížeče souborů, že jejich správné použití není vůbec intuitivní. Navíc při práci s touto aplikací často narazíte na nejednoznačná chybová hlášení, která neobsahují bližší popis chyby.

Termux[3]

Pro vývojáře upřednostňující příkazový řádek je možnost instalace aplikace *Termux* a nainstalování gitu do prostředí jeho terminálu. Tam je i možné doinstalovat rozšíření *Git LFS* a *Git Annex*. *Git LFS* lze doinstalovat přímo jako balíček. *Git Annex* je možné stáhnout z jeho oficiálních webových stránek[4] a dle návodu[5]uvést do provozu. Obě tato rozšíření lze ovládat z příkazové řádky, přičemž *Git Annex* i přes uživatelské rozhraní. To je implementováno v prohlížeči. Tato webová aplikace je přehledná i pro mobilní zařízení a umožňuje synchronizaci souborů mezi repozitáři různých zařízení.

2.3.2 Desktop

Na Linux i Windows existuje mnoho aplikací, které práci s repozitáři zvládají velice dobře. Nicméně prostředí Androidu je od toho desktopového natolik rozdílné, že prostor pro inspiraci je značně omezený.

GitKraken[6]

Dobré zkušenosti mám například s aplikací *GitKraken*. Ta zobrazuje repozitář přehledně ve stromové struktuře. V ní lze přímo najetím myši na uzel provádět změny. Příkazy gitu má

[1] https://play.google.com/store/apps/details?id=com.manichord.mgit
[2] https://play.google.com/store/apps/details?id=com.aor.pocketgit&hl=en
[3] https://play.google.com/store/apps/details?id=com.termux&hl=en
[4] https://git-annex.branchable.com/
[5] https://git-annex.branchable.com/Android/
[6] https://www.gitkraken.com/

přehledně zobrazené v horním panelu. Navíc jsou zde dobře řešeny konflikty v souborech. Na jedné straně obrazovky vidíte jednu verzi a na druhé straně druhou. Ve spodní části obrazovky se generuje nová verze. Tu vytváříte postupným procházením obou současných verzí a vybíráním vyhovující varianty. *GitKraken* umí pracovat i s *Git LFS*. K ovládání takto sledovaných souborů používá zvláštní vysouvací nabídku s funkcemi *Git LFS*. Ta se v případě práce s repozitářem podporující toto rozšíření zobrazí vedle základních funkcí. Které soubory takto sleduje lze měnit v nastavení repozitáře nebo při přidávání souborů do stage.

Ungit[7]

Na první pohled dobrým dojmem působí i aplikace *Ungit*. Ta vás při každé akci naviguje krok po kroku a usnadňuje tak používání gitu pro méně zkušené uživatele. Jedná se o webovou aplikaci založenou na *node.js*. Pro její instalaci je třeba příkazová řádka, pro spuštění pak webový prohlížeč. Její hlavní výhoda je tedy nezávislost na platformě. Její ovládání je rychlé, jelikož aplikace zjednodušuje určité procedury gitu. Například sama nabízí `Commit` bez nutnosti přidávat soubory do `Stage`. Nicméně aplikace tím zapouzdřuje většinu funkcí. Na základní obrazovce kromě stromu změn repozitáře není další ovládací prvek a aplikace se tak v konečném důsledku jeví až příliš uzavřeně.

2.3.3 Zhodnocení průzkumu

Z testování aplikací vyplynulo, že nejjednodušší způsob práce s gitem je tehdy, když aplikace transparentně zobrazuje příkazy gitu a jejich použití nechá na uživateli. Předejde se tím chybám, jejichž hlášení nejsou vždy dostačující k vyřešení problému. Pokud je funkce dobře zpracována, není třeba vést uživatele krok po kroku. Ovládání se tak urychlí a je stále přehledné.

Testované aplikace často využívají vlastní textový editor a správce souborů. V obou případech tyto aplikace integrují velice jednoduché verze a jejich použitelnost je tak značně omezená.

Dalším bodem jsou chybová hlášení. Těm by měla aplikace pokud možno předcházet. Pokud chybě již není vyhnutí, alespoň by měla mít dobrý popis a nebo i návrh jejího řešení.

Poslední bod se týká uživatelského rozhraní. Aplikace *MGit* při klonování repozitáře užívá skrývání určitých položek při jejich nadbytečnosti. To je sice užitečný prvek, nicméně při skrytí položky dojde k posunutí těch následujících na její místo a to působí velice rušivě.

[7] https://github.com/FredrikNoren/ungit

Kapitola 3

Vývoj aplikací pro systém Android

Android je open-source platforma vyvinutá společností *Google*. Její první oficiální verze se dostala na svět 23. Října 2008 a od té doby značně vyspěla. Je založena na systému Linux a většina fyzických zařízení, které ji podporují staví na *arm* architektuře. Android totiž není mířen přímo na konkrétní zařízení tak jako například *iOS* od firmy *Apple*. To přináší mnohé kompromisy, které musí postupovat jak její vývojáři, tak samotní programátoři aplikací. Zařízení se liší svým hardwarem i softwarem. Mají různé velikostí pamětí i displejů.

Při vývoji aplikací pro ni je tedy nutné brát ohled na nejnovější trendy a sledovat procentuální zastoupení kritických parametrů tak, aby výsledná aplikace splňovala zadané požadavky na většině cílových zařízení. Různá zařízení na trhu různých značek se navíc liší svým aplikačním binárním rozhraním. To vše má za následek roztříštěnost aplikací podle mnoha kritérií tak, aby byli uživatelsky přívětivé na co možná nejvíce zařízeních. *Android* o těchto problémech samozřejmě ví a při jejím vývoji je k dispozici mnoho nástrojů, které se kterými je možné je řešit.

Tato kapitola se zabývá teoretickými základy tohoto systému, které je užitečné mít pro úspěšný vývoj jeho aplikací na paměti. Při získávání přehledu o principu programování Android poslouží zejména oficiální online dokumentace[1] a návody [2]. Především z těch čerpá následující text. Také je možné najít různou kvalitní tištěnou literaturu. Pro účely této aplikace se například osvědčila kniha *Vývoj aplikací pro Android*[2].

3.1 Základy aplikace

Aplikace pro Android mohou být psány v Kotlinu, Javě, nebo C++. Nástroje Android SDK kompilují kód spolu s ostatními potřebnými daty do APK souboru. Prakticky se jedná o zip archiv, který Android používá pro instalaci aplikací.

Každá aplikace pracuje ve svém vlastním uzavřeném prostoru. Android implementuje princip nejmenších pověření (v originále *principle of least privilege*). Ten zaručuje, že každá aplikace má práva k přístupu jen ke zdrojům, které potřebuje. Další práva lze aplikaci přidat pouze s explicitním souhlasem uživatele.

3.2 Komponenty aplikace

[1] https://developer.android.com/docs
[2] https://developer.android.com/guide/

Kapitola 4

Návrh aplikace

Dle zhodnocení průzkumu a vlastních zkušeností jsem usoudil, že aplikace bude

1. přehledná, ale nebude příliš zapouzdřovat příkazy gitu.
2. uživatele přehledně informovat o tom co právě dělá, co očekává a jaký je výstup.
3. využívat externí správce souborů i textový editor.
4. mít co nejmenší počet za sebou následujících aktivit a tedy i přechodů mezi nimi.

4.1 Funkce aplikace

Git je velice komplexní systém a proto hrozí, že jeho plná implementace by na zařízeních android byla velice nepřehledná. Vybrány byly tedy nejdůležitější funkce, které jsou nutné pro prohlížení a úpravu repozitářů.

4.1.1 Správa repozitářů

Po spuštění aplikace uživatele uvítá obrazovka se seznamem sledovaných repozitářů. Repozitář je možné do něj přidat několika způsoby. Prvním je přidání již existujícího repozitáře specifikováním jeho cesty v úložišti zařízení. Druhým je klonování nebo inicializace repozitáře z prostředí aplikace. Tento seznam repozitářů je synchronizován s úložištěm zařízení. Příkazy gitu bude moci uživatel provádět po otevření daného repozitáře.

4.1.2 Příkazy gitu

Jelikož žádné aplikace pro Android kromě *Termux* neimplementují rozšíření *Git Annex* a nenalezl jsem knihovnu, která by toto dokázala, bylo rozhodnuto pro příkazy gitu využít zkompilované binární soubory. Mezi tyto funkce patří i funkce jednotlivých rozšíření. Všechny tyto příkazy budou dostupné při otevření repozitáře v bočním výsuvném panelu aplikace. V případě velkého množství takových příkazů budou rozděleny do kategorií, či se zobrazí jen v případě, kdy má jejich užití smysl. Pro transparentní zobrazení stavu repozitáře budou tyto funkce zobrazovat i svůj klasický textový výstup.

4.2 Použité technologie a nástroje

Před samotným programováním aplikace bylo třeba udělat průzkum nástrojů, které se při vývoji na zařízení Android používají. Tyto nástroje byly vybrány s důrazem na efektivitu vývoje i náročnost jejich použití. Nejdůležitějším z nich je *Android Studio*[1], prostředí, ve kterém probíhal vývoj aplikace. Aplikace byla dále vyvíjena za použití Android Jetpack[2]. Ty přináší komponenty pro efektivní vývoj aplikací. Pro verzování byl použit nástroj git, prostřednictvím aplikace *GitKraken*[3]. Kód aplikace byl synchronizován se vzdáleným repozitářem na serveru *GitHub*[4]. Pro dynamické generování instalačních souborů aplikace byl repozitář navíc synchronizován s *GitLab CI/CD*[5]. Pro vytváření binárních souborů gitu a git LFS byl použit *Docker*[6]. Obraz pro jejich kompilace poskytuje aplikace *Termux packages*[7].

4.3 Architektura aplikace

Aplikace je psána v jazyce *Java*. Dále využívá návrhového vzoru *Model–view–viewmodel* (dále jen MVVM) a *SQLite* databáze. K přístupu k databázi používá *Room Persistence Library* (dále jen Room). Tato knihovna poskytuje abstraktní vrstvu nad databází *SQLite*, zajišťující robustní přístup při plném využití potenciálu tohoto systému řízení báze dat.

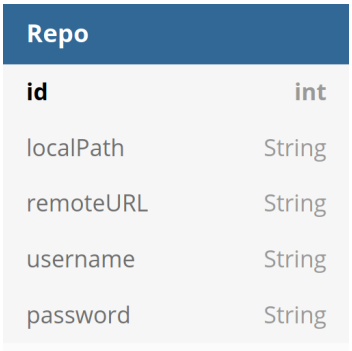
4.3.1 Kotlin vs. Java

Od 7. května 2019 se *Kotlin* stal preferovaným jazykem vývoje pro Android. Proto jsem od začátku plánoval programovat aplikaci právě v něm. Nicméně během programování aplikace jsem zjistil, že naprostá většina zdrojů na internetu pro řešení problémů pro tuto platformu je psána v Javě. Android studio sice umožňuje zkonvertovat kód do Kotlinu, nicméně ani to není to vždy dokonalé. Užití Kotlinu má tu výhodu, že dovoluje programátorovi vynechat určité části kódu, které jsou nutné pro běh aplikace, ale přímo neřeší daný problém. V angličtině se pro ně vžil výraz *boilerplate code*. Ovšem tento kód je přesto třeba vygenerovat, ale o to se již stará Kotlin. To je také jeden z důvodů, proč Kotlin trvá déle zkompilovat. Pokročilým Android vývojářům jistě přijde rychlejí práce vhod, ale jako začínají programátor na této platformě více ocením transparentnost Javy.

[1] https://developer.android.com/studio
[2] https://developer.android.com/jetpack
[3] https://www.gitkraken.com/
[4] https://github.com/
[5] https://docs.gitlab.com/ee/ci/
[6] https://www.docker.com/
[7] https://github.com/termux/termux-packages

4.3.2 Databáze

Databáze je využívána pro získání přehledu o repozitářích gitu, které chce uživatel aplikací sledovat. Každý takový repozitář je reprezentován entitou databáze `Repo`. Tato tabulka obsahuje absolutní cestu ke složce repozitáře, status repozitáře, dále URL vzdáleného repozitáře, uživatelské jméno a heslo pro přístup k němu. Všechny tyto položky je třeba při provádění příkazů gitu aktualizovat tak, aby stav entity v okamžitém časem odpovídal stavu repozitáře.



Obrázek 4.1: Entita repozitáře

4.3.3 Návrhový vzor

Model–view–viewmodel je v době psaní této práce doporučovaným návrhovým vzorem Android aplikací. K volbě tohoto návrhového vzoru dopomohlo také využití knihovny *Room*. Tato knihovna totiž spoléhá na využití *MVVM* vzoru alespoň pro účely funkčnosti databáze. Je tomu tak proto, že data, která závisí na databázi se ukládají do proměnné datového typu *LiveData*. Hodnotu této proměnné lze sledovat a na jejím základě řídit běh aplikace. Aby byla hodnota této proměnné perzistentní při běhu aplikace, uchovává se její hodnota ve *viewmodelu*. Hlavní takovou proměnnou je v této aplikaci seznam všech git repozitářů. Ta se nachází ve třídě `RepoRepository`, nicméně její instanciace se provádí pouze v části `view_model` MVVM.

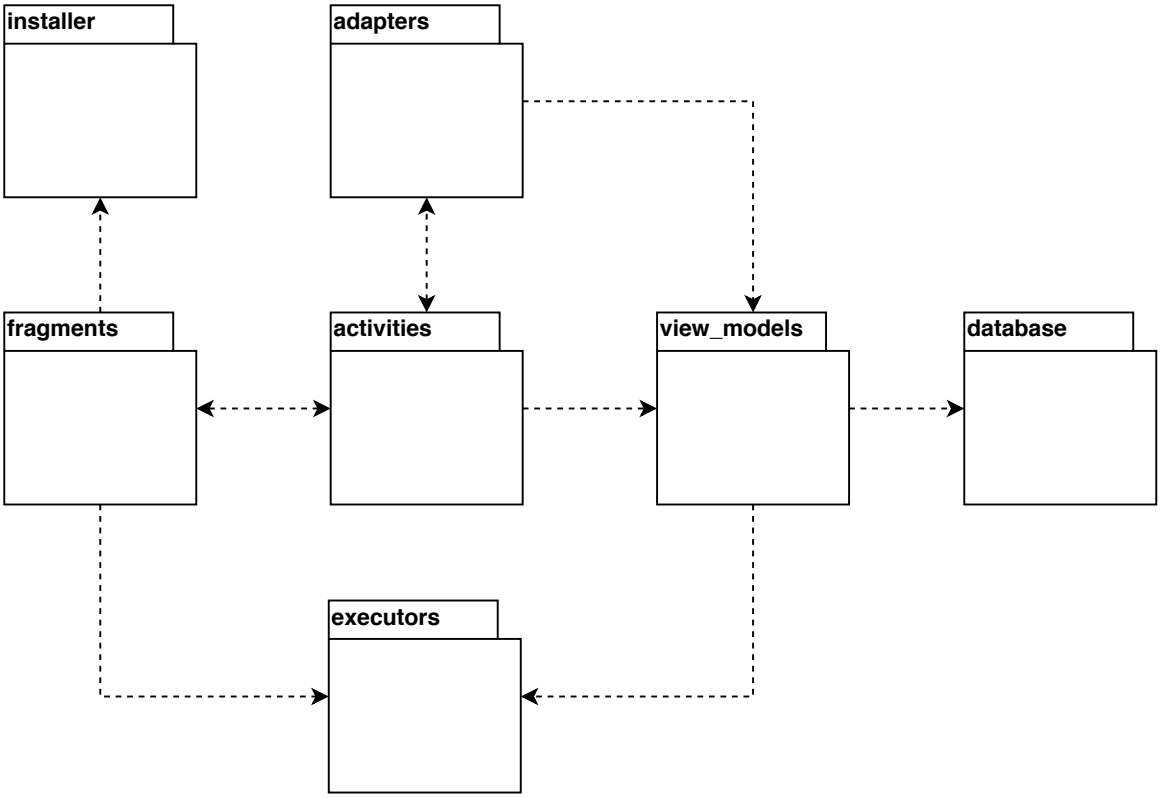
4.3.4 Obrazovky Aplikace

Aplikace bude rozdělena podle obrazovek do aktivit. Tyto aktivity dále mohou obsahovat různé fragmenty. Ke každé aktivitě, která poskytuje určitou obrazovku je připojen její ViewModel. Ten jí poskytuje data a funkcionalitu. Vzhled obrazovky je dán jejím layoutem.

4.3.5 Dělení aplikace do balíčků

Podle zaměření tříd je aplikace dělena do třech základních balíčků. Jsou jimi `java`, `cpp`, a `res`. V balíčku `java` je specifikováno chování aplikace. Hlavním obsahem balíčku `cpp` jsou archivy s binárními soubory gitu a program `bootstrap.c` pro jejich instalaci. Posledním balíčkem je `res`. Ten obsahuje všechny layouty, ikony a další grafické i textové prvky, které aplikace používá pro grafické rozhraní.

Následující popis funkčnosti a závislostí balíčků se bude týkat balíčku `java`. Záměrně byl z diagramu vynechán balíček `utilities`. Jeho třídy lze použít kdekoliv v aplikaci a pro budoucí vývoj aplikace jeho zahrnutí nemá opodstatnění.



Obrázek 4.2: Diagram závislostí balíčků

activities

Nejdůležitější součást balíčku `Java` je balíček `activities`. Ten aplikaci dělí na obrazovky a o každou z nich se stará jedna třída. Tedy v případě, že aplikace potřebuje určitou obrazovku, je zavolána příslušná aktivita s jejím chováním. Všechny aktivity aplikace rozšiřují základní aktivitu `BasicAbstractActivity`. Ta implementuje společné prvky rozhraní aktivit. Například získávání oprávnění, zobrazení různých oznámení a dialogů.

adapters

Tento balíček obsahuje třídy, které slouží k zobrazení položek stejného typu. Tato aplikace je využívá k zobrazení seznamu repozitářů základní obrazovky a funkcí gitu v bočním výsuvném panelu.

database

Tento balíček obsahuje balíček `model`, ve kterém se nachází třída Repo. Ta implementuje tabulku databáze uchovávající všechny potřebné informace o repozitáři. Instance databáze se uchovává ve třídě `RepoDatabase`. K přístupu k ní se využívá třída `RepoDao`. Tato třída obsahuje metody volající *SQLite* dotazy databáze. Aplikaci je databáze zprostředkována třídou `RepoRepository`, která odpovídá `Repository` modelu MVVM. Databáze se ukládá do bezpečného vnitřního prostoru balíčku aplikace.

fragments

Fragmenty, třídy které dynamicky rozšiřují nebo mění obsah aktivity. Aplikace používá fragmenty například k instalaci a nastavení. Každá aktivita může obsahovat několik fragmentů, které samostatně řeší určitou část aktivity.

install

O instalaci binárních souborů se stará třída `InstallTask` v balíčku `install`. Ta při prvním spuštění aplikace zkopíruje potřebné soubory ze složky `cpp` do interní paměti zařízení.

executors

Základní příkazy gitu i jeho rozšíření jsou implementovány v balíčku `executors`. Tyto třídy využívají základní třídu `BinaryExecutor` využívající `ProcessBuilder`. Ta spustí binární soubor, který vykoná danou funkci na zařízení.

utilities

Jedná se o statické metody a proměnné, které jsou použitelné kdekoliv v aplikaci.

view_models

Třídy obsahující perzistentní data a implementující logiku nad nimi. Tento balíček odpovídá části *ViewModel* MVVM a poskytuje aplikaci metody zajišťující její funkčnost. Komunikace mezi třídami ViewModelů a aktivitami je zajištěna pomocí *databindingu*, *observerů* a veřejných metod, které tyto třídy poskytují.

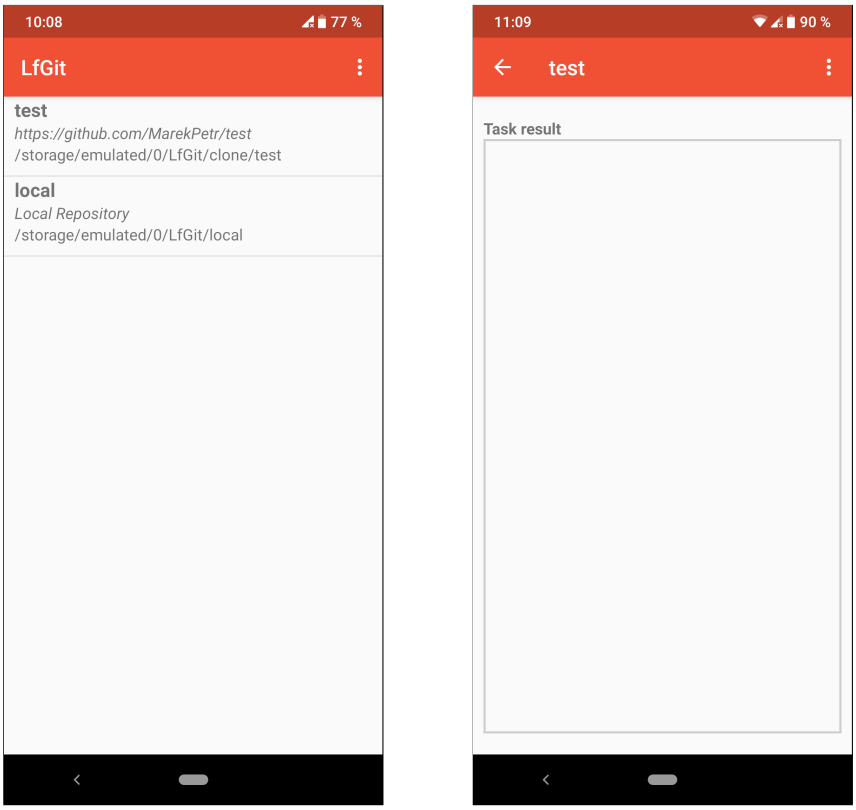
4.4 Grafické uživatelské rozhraní

Významnou součástí řešení mobilní aplikace je i její uživatelské rozhraní. To bylo navrženo s důrazem na užití *Material Designu*[8]. Uživatelé Android jsou na něj zvyklí z většiny populární aplikací a orientace v něm je tedy pro ně bezproblémová. Navíc *Android studio* používá jeho prvky jako výchozí při tvorbě aplikace.

Protože důraz této práce je spíše na funkčnost, než samotné rozhraní, bude toto rozhraní co nejjednodušší.

Grafické rozhraní se nejvíce inspiruje aplikací *MGit* a přidává prvky vzniklé z požadavků na aplikaci. Především se jedná o přidání funkcí rozšíření a změnu rozhraní pro práci s repozitářem. Uživateli bude po volání funkcí gitu sdělen přesný textový výstup, který mu poslouží pro další práci s gitem.

Nejprve byly na papír navrženy velice jednoduché wireframy pro ujasnění obsahu nejdůležitějších obrazovek. Ty byly postupně testovány a přepracovávány tak, aby poskytly přívětivé ovládaní aplikace. Poté byly tyto výsledné obrazovky naprogramovány přímo v Android studiu a užity pro první prototypy aplikace. Ukázka takto získaných obrazovek je vidět na obrázku 4.3.



(a) Seznam repozitářů (b) Otevřený repozitář

Obrázek 4.3: Základní obrazovky

Obrazovka 4.3a zobrazuje úvodní obrazovku aplikace. Nachází se na ní seznam repozitářů. U každého z nich jsou uvedeny jeho detaily. Jedná se o název repozitáře, jeho vzdálené

[8] https://material.io/

umístění na serveru a místní cestu v zařízení. Pro přidání repozitářů a nastavení aplikace slouží menu v pravém horním rohu.

Po otevření repozitáře aplikace přejde na druhou obrazovku 4.3b. Tam uživatel vykoná operace nad repozitářem. Ty budou dostupné v pravém draweru. Výsledky jednotlivých operací gitu budou vypisovány do *Task result* pole. Pokud aplikace narazí na problém v rámci vstupu uživatele, upozorní ho příslušným *Toastem*.

4.5 Manipulace se soubory

Správce souborů je možné implementovat různými způsoby s různým stupněm jeho komplexnosti. Pro otevírání a editování souborů, včetně symbolických odkazů uživatel užije externí aplikace. Je mu tak ponechána volnost při volbě tohoto správce a předejde se hledání kompromisů pro jeho implementaci. Navíc aplikace získá větší prostor pro ostatní funkce a uživatelské rozhraní se zjednoduší. Nevýhodou může být chybějící přehledný výběr souborů pro funkce, které pracují s jednotlivými soubory. Ale i s tím lze v aplikaci pracovat využitím *SAF*[9].

4.6 Možné způsoby instalace binárních souborů

Jak již bylo zmíněno, aplikace pro příkazy gitu využívá binárních souborů. Ty je samozřejmě nejprve nutné do prostoru aplikace nějakým způsobem přenést. Z důvodu architektury systému Android není tato operace tak přímočará jako například na desktopovém systému Linux. Existují zde dvě možnosti. První je využití nativní knihovny o jejíž přenos a spouštění se postará systém Android. Druhá možnost je tyto operace provádět v rámci aplikace po instalaci balíčku.

4.6.1 Nativní knihovny

Z implementačního pohledu nejjednodušší způsob je první možnost. Tedy užití binární knihovny s příponou *.so*. Tuto knihovnu je třeba v rámci struktury aplikace umístit do správného adresáře a systém Android si s její instalací poradí během samotné instalace aplikace. Tato metoda je dobře aplikovatelná v případě, že máte k dispozici staticky linkované binární soubory se strojovým výstupem. Z nich je pak snadné za použití Android NDK[10] a JNI[11] vytvořit funkce, které lze používat přímo v kódu a získávat tak z těchto knihoven jejich výstup. Staticky linkované binární soubory v sobě obsahují všechny potřebné závislosti a jejich použití je tak možné samostatně. Lze jim tedy jednoduše přiřadit potřebnou příponu a budou zcela funkční. Získat tyto soubory je možné například křížovou kompilací daného programu. Staticky linkovaný git je možné zkompilovat využitím již existujících nástrojů[12]. Problém s tímto způsobem tkví v tom, že takto získaný binární soubor nelze jednoduše modifikovat. Je nutné ho pokaždé znovu zkompilovat, což je časově velice náročné. Navíc tyto binární soubory musí obsahovat veškeré knihovny, které pro svůj běh potřebují. Tedy při použití více těchto binárních souborů dochází k jejich redundanci.

[9] https://developer.android.com/guide/topics/providers/document-provider
[10] https://developer.android.com/ndk
[11] https://developer.android.com/training/articles/perf-jni
[12] https://github.com/EXALAB/git-static

4.6.2 Vlastní binární soubory

Druhá možnost je využít binárních souborů dynamicky linkovaných a jejich spouštění implementovat v rámci aplikace. Tyto binární soubory nemají jejich závislosti obsažené přímo v nich samých, ale hledají je v daných umístěních. Tím dochází k úspoře místa. Také jsou jednoduše rozšiřitelné. Tento způsob řešení ale není pro zařízení Android zcela běžný a přináší tak další řadu problémů. Předně je nutné mít je zkompilované pro pevně danou cestu a správně nastavovat systémové proměnné. Dále tyto soubory nelze spouštět přes rozhraní JNI, ale dosáhne se toho využitím tříd, které vytváří vlastní proces na základě dodaných parametrů. Těmi jsou například metoda `exec` třídy `Runtime`[13] nebo právě již zmíněná metoda `command` třídy `ProcessBuilder`[14]. Toto spouštění i následné čtení výsledků je nutné provádět na samostatném vlákně a implementace tak není zcela triviální.

4.7 Návrh instalace binárních souborů

Pro tuto aplikaci bylo použito dynamicky linkovaných binárních souborů s vlastní instalací i spouštěním. Lépe se s nimi pracuje a aplikace tak bude lépe do budoucna rozšiřitelná. Navíc je velké množství dynamicky linkovaných programů již připraveno pro kompilaci prostřednictvím *Termux-packages*[15]. Pro samotné spouštění bylo využito třídy `ProcessBuilder`. Je snadno použitelná a umožňuje intuitivní nastavování různých parametrů běhu procesu.

4.8 Aplikační binární rozhraní - ABI

Různá zařízení Android mají osazeny různé procesory, které podporují různé sady instrukcí. Každá taková kombinace procesoru a instrukční sady má vlastní Aplikační binární rozhraní - *ABI*[16]. Většina fyzických zařízení používá architekturu *arm*[17]. Pro ladění aplikací se používá Android emulátorů a ty naopak pro nejlepší výkon používají architekturu *x86*. Pro vydání aplikace na *Google Play* je nutné, aby aplikace podporovala 32-bitové i 64-bitové verze dané architektury. Proto aplikace podporuje obě architektury pro obě verze. Jelikož by byl takto vytvořený *APK* balíček příliš velký, aplikace používá pro distribuci formát *Android App Bundle*[18].

[13] https://developer.android.com/reference/java/lang/Runtime
[14] https://developer.android.com/reference/java/lang/ProcessBuilder
[15] https://github.com/termux/termux-packages/
[16] https://developer.android.com/ndk/guides/abis
[17] https://handstandsam.com/2016/01/28/determining-supported-processor-types-abis-for-an-android-device/
[18] https://developer.android.com/guide/app-bundle

Kapitola 5

Implementace

Po návrhu řešení aplikace následuje implementační část. V předchozí kapitole zabývajícím se návrhem byly popsány základní části aplikace, využité nástroje a architektura vývoje. Tato kapitola pojednává o postupu vývoje aplikace od získání binárních souborů po samotné vydání aplikace.

5.1 Získání spustitelných binárních souborů

Následující text rozebírá postup získání binárních souborů pro účely funkcí gitu, způsob jejich instalace a spouštění.

5.1.1 Kompilace binárních souborů

Jak již bylo zmíněno při návrhu, binární soubory jsou zkompilovány využitím repozitáře *Termux-packages*[1]. Ten pro tento účel poskytuje obraz *Dockeru*. Pro jeho použití pro jinou aplikaci je nutné upravit skript `scripts/build/termux_step_setup_variables.sh` tak, aby cesta ke spustitelným souborům odpovídala cílové aplikaci. Při kompilaci pro tuto aplikaci byla nastavena cesta `TERMUX_PREFIX` na `/data/data/com.lfgit/files/usr`. Takto byly zkompilovány binární soubory pro git i *Git LFS*. *Git Annex* touto cestou bohužel získat nelze. Jeho kompilace je velice problémová a přes veškeré úsilí se nakonec nepodařila. Další informace a provedený postup naleznete v sekci 5.2.4.

5.1.2 Instalace binárních souborů

Pro instalaci binárních souborů bylo využito třídy `TermuxInstaller` aplikace Termux[2]. Ta řeší podobný problém při instalaci linuxového prostředí a využití části metody `setupIfNeeded` vyřešilo problémy s instalací symbolických odkazů do aplikaci. Běžné kopírování souborů, jehož metody jsou popsány například zde [3] totiž kopírují soubory, na které tyto symbolické soubory ukazují a tím dochází k jejich redundanci a nabývání velikosti instalace. Tato část byla implementována třídou `installTask` v balíčku `install`.

[1] https://github.com/termux/termux-packages/

[2] https://github.com/termux/termux-app/blob/master/app/src/main/java/com/termux/app/TermuxInstaller.java

[3] https://www.baeldung.com/java-copy-file

Tento postup instalace vyžaduje užití *Android NDK*[4] pro získání zdroje dat ze zkomprimovaného souboru ve formátu *ZIP*. Ty jsou využity jak pro snížení velikosti, tak pro snadnou implementaci načtení a přenosu souborů.

Po kompilaci byly smazány některé nepotřebné soubory zvětšující velikost instalace. Dále bylo třeba vygenerovat seznam symbolických odkazů pro všechny architektury. Ten byl vygenerován příkazem `find . -type l -ls > SYMLINKS.txt`. Poté byl tento seznam upraven tak, aby odpovídal formátu `symlink` → `file`. Třída `installTask` byla upravena tak, aby s tímto formátem pracovala. Soubor `SYMLINKS.txt` byl dále připojen ke složce s binárními soubory dané architektury. Celá tato složka byla zkomprimována a archiv přesunut do složky `cpp`. Z těchto archivů se poté během instalace aplikace za pomocí *Android NDK* kopírují soubory do zařízení.

5.1.3 Spouštění binárních souborů

Jak již bylo zmíněno 4.6.2, aplikace používá pro spouštění spustitelných souborů `Process Builder`. Ten je implementován metodou `run` třídy `BinaryExecutor`. `Process Builder` se sice postará o vytvoření nového procesu, ale pokud po jeho ukončení očekáváme nějaký výstup, nemůže se samozřejmě běh aplikace během čekání blokovat. Proto se tyto příkazy spouští na samostatném vlákně a výsledek se předává prostřednictvím příslušného *callbacku*, tedy zpětného volání. Toto zpětné volání je implementováno rozhraním `ExecListener` třídy `executors`. Toto rozhraní poté implementuje třída, která očekává výsledek daného volání.

[4] https://developer.android.com/ndk/

5.2 Aplikace

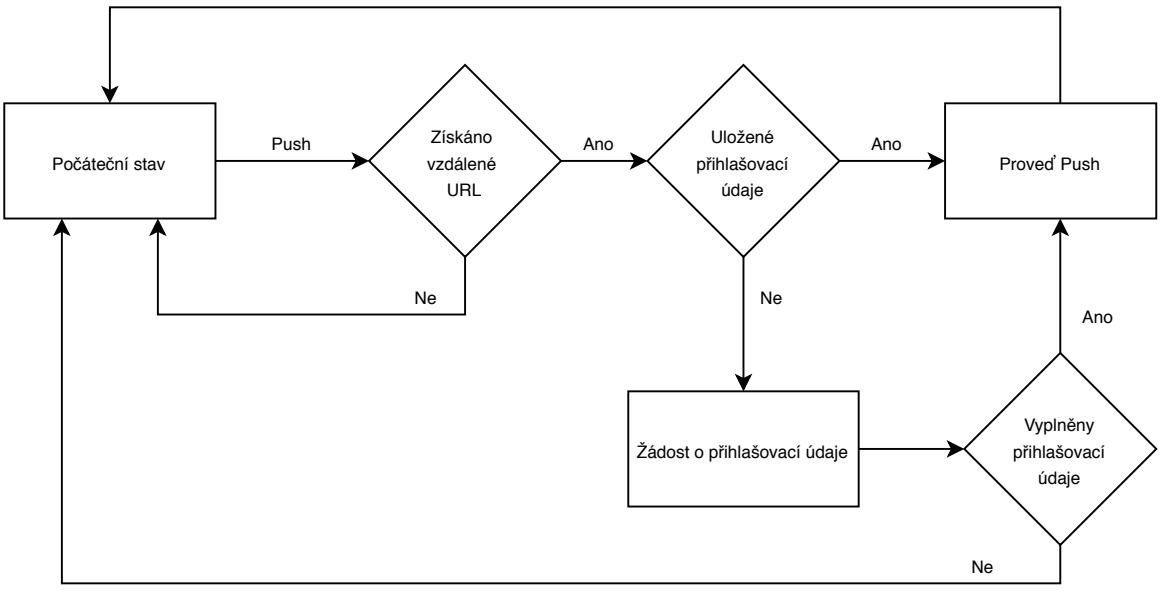
Po implementaci spouštění binárních souborů přichází na řadu implementace samotné aplikace. Základní popis implementace stěžejních částí je rozdělen podle jednotlivých obrazovek aplikace.

5.2.1 Příkazy gitu

Příkazy gitu jsou implementovány ve view modelech aktivit, které je používají. Jelikož všechny volané funkce vrací hodnotu zpětným voláním, bylo nutné pro získání všech vstupů pro vykonání daného příkazu definovat stavy jeho zpracování. K tomu slouží třída `TaskState`. Obsahuje atribut `mInnerState` určující aktuální stav rozpracovaného příkazu a `mPendingTask` definující zpracovávaný příkaz. Existují zde dva speciální stavy atributu `mInnerState` `FOR_APP` a `FOR_USER`. Jsou to stavy, které určují, jestli bude výsledek následujícího příkazu zobrazen uživateli, či bude využit jako vstupní argument pro jiný účel aplikace.

Push

Nejsložitější příkaz na implementaci je příkaz *push*. Ten pro přístup do vzdáleného repozitáře potřebuje jak jeho *URL*, tak přihlašovací údaje uživatele. Ty jsou žádány pro každý nově přidaný repozitář pouze při prvním vykonávání příkazu *push*. Po jejich zadání jsou uloženy v databázi v tabulce daného repozitáře. Jelikož je tato databáze uložena ve vnitřním prostoru aplikace, kam má přístup jen ona samotná, je uložení těchto citlivých údajů relativně bezpečné. Následující diagram znázorňuje vykonání úlohy tohoto příkazu.



Obrázek 5.1: Zjednodušený diagram provedení úlohy příkazu push.

5.2.2 Seznam repozitářů

Uvítací obrazovka je implementována aktivitou `RepoListActivity`. Ta při prvním spuštění nainstaluje potřebné binární soubory a dále zobrazí prázdný seznam repozitářů. Repozitář uživatel do seznamu přidá prostřednictvím menu. Tato akce spustí daný *Intent* a přesune uživatele na další obrazovku.

5.2.3 Přidání repozitáře

Repozitář lze do seznamu přidat třemi způsoby. Pokud již existuje v paměti zařízení, lze ho přidat tlačítkem menu `Add repository`. V tomto případě dojde k přidání cesty repozitáře do databáze repozitářů. Před tím se ještě zkontroluje, jestli repozitář obsahuje vzdálenou adresu.

5.2.4 Problémy objevené při implementaci

Symbolické odkazy

Autentizace Git

Git LFS

Git annex

Kapitola 6

Testování

Kapitola 7

Závěr

Literatura

[1] CONTRIBUTORS, W. *Git-annex* [online]. Wikipedia, The Free Encyclopedia., květen 2015 [cit. 2020-04-03]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Git-annex&oldid=915249727.

[2] LACKO Luboslav. *Vývoj aplikací pro Android*. Computer Press (CP Books), 2015. ISBN 978-80-251-4347-6.