VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

MINIMALISTICKÝ SPRÁVCE APLIKACÍ A SOUBORŮ PRO OS ANDROID

MINIMALIST FILE MANAGER AND LAUNCHER FOR ANDROID OS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **VOJTĚCH HERTL**
AUTHOR

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

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Hertl Vojtěch**
Program: Informační technologie
Název: **Minimalistický správce aplikací a souborů pro OS Android**
Minimalist File Manager and Launcher for Android OS
Kategorie: Uživatelská rozhraní

Zadání:

1. Seznamte se s vývojem aplikací pro operační systém Android. Prozkoumejte rozhraní pro přístup a správu instalovaných aplikací a správu souborů a adresářů. Prozkoumejte možnosti optimalizace Android aplikací.
2. Navrhněte software - správce aplikací a souborů, který bude mít minimální požadavky na systém (minimální velikost, spotřeba paměti a energie, co nejmenší verze Android API, maximální využití plochy obrazovky, čitelnost i při velkém počtu informací o zobrazených objektech a akcích). Správce se bude nastavovat pomocí editace konfiguračních souborů, a to i za běhu aplikace.
3. Po konzultaci s vedoucím aplikaci správce implementujte. Proveďte optimalizaci pro minimální požadavky na systém a výsledky změřte.
4. Výsledek popište, vyhodnoťte a zveřejněte jako open-source.

Literatura:

- Ľuboslav Lacko. Vývoj aplikací pro Android. Computer Press, Brno, 2015. ISBN 978-80-251-4347-6.
- Faisal Alam, Preeti Ranjan Panda, Nikhil Tripathi, Namita Sharma, and Sanjiv Narayan. Energy optimization in Android applications through wakelock placement. In Proceedings of the conference on Design, Automation & Test in Europe (DATE '14). European Design and Automation Association, Belgium, 2104.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Rychlý Marek, RNDr., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 31. října 2018

Abstrakt

Cílem této práce je vytvořit mobilní aplikaci pro operační systém Android. Aplikace má být minimalistický správce souborů a aplikací. Po důkladné analýze a širokém studiu byla aplikace navrhnuta a implementována. Splnila požadavky a naplnila stanovené cíle. Výsledkem této práce je jedinečný minimalistický a plně funkční správce podporující velký rozsah zařízení a je nenáročný na systém.

Abstract

The aim of this thesis is to create a mobile application for the Android operating system. The application is supposed to be a minimalist file manager and launcher. After thorough analysis and extensive study, the application was designed and implemented. It met the requirements and fulfilled the set goals. The result of this work is a unique minimalist and fully functional application that supports a wide range of devices and has low system requirements.

Klíčová slova

Android, správce, správce souborů, správce aplikací, optimalizace, minimalistický

Keywords

Android, manager, file manager, file explorer, application manager, launcher, optimization, minimalist

Citace

HERTL, Vojtěch. *Minimalistický správce aplikací a souborů pro OS Android*. Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce RNDr. Marek Rychlý, Ph.D.

Minimalistický správce aplikací a souborů pro OS Android

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana RNDr. Marka Rychlého, Ph.D. a uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Vojtěch Hertl
12. května 2019

Poděkování

Chtěl bych poděkovat především mému vedoucímu, panu RNDr. Markovi Rychlému, Ph.D., za skvělé vedení práce a jeho podnětné poznatky. Dále děkuji všem kolegům, se kterými jsem mohl tuto práci konzultovat a po jejichž boku jsem prošel bakalářským studiem. A v neposlední řadě také děkuji rodině a přítelkyni za podporu a pochopení.

Obsah

Kapitola 1

Úvod

V současné době bývá trendem vytvářet takové mobilní aplikace, které poskytují mnoho funkcí, kladou důraz na vizualizaci a zahlcují tak uživatele. Aplikace pro operační systém Android, které slouží ke správě souborů a aplikací, tyto trendy následují. Na úkor přehlednosti a požadavků na systém se snaží podporovat nadbytečnou funkcionalitu a používat nejnovějších prvků uživatelského rozhraní.

Úkolem této práce je podrobně prozkoumat tyto aplikace a stanovit cíle a navrhnout samotnou aplikaci správce souborů a aplikací. Bude proveden návrh řešení, které bude zaměřeno právě na jednoduchost ve všech ohledech. Řešení, které bude funkční na co nejvíce různě výkonných zařízeních a bude přehledné a intuitivní. A v neposlední řadě také řešení, které bude mít minimální požadavky na systém. Následně bude navržená aplikace implementována a důkladně otestována na různých mobilních zařízeních s operačním systémem Android. Na závěr bude celá aplikace optimalizována, aby své stanovené cíle splňovala ještě lépe a výsledné požadavky na systém budou změřeny.

Celá aplikace je volně dostupná v online repozitáři[1] ke klonování či úpravě.

[1] https://github.com/hertl/minimalistic_manager

Kapitola 2

Úvod do problematiky a rozbor existujících řešení

Cílem této práce je vytvořit minimalistického správce aplikací a souborů pro operační systém Android. Tato kapitola má za úkol seznámit čtenáře se zadáním, vysvětlit základní pojmy a nastínit, proč je důležité se tímto tématem zabývat. Jelikož podobných implementací existuje už mnoho, budou také analyzována existující řešení pro případnou inspiraci.

2.1 Správce souborů a aplikací

Správce souborů, nebo také prohlížeč souborů, je počítačový program, který poskytuje uživatelské rozhraní pro správu a souborů a adresářů. Mezi nejběžnější operace prováděné nad soubory a adresáři patří vytváření, otevírání, přejmenování, přesunování nebo kopírování a mazání. Správce aplikací přidává možnost zobrazení a manipulace s nejen systémovými, ale i jinými nainstalovanými aplikacemi v zařízení.[28]

Výsledkem by tedy měl být program, který pracuje se systémovými prvky a poskytuje rozhraní pro uživatele. Uživatel bude mít možnost provádět alespoň základní operace se souborovým systémem.

2.2 Android

Android je operační systém primárně určený pro mobilní zařízení. V současné době patří společnosti Google. Stejně jako operační systém Linux, nad kterým je implementován, se Android vyskytuje ve formě open source napsaný primárně v programovacím jazyce Java. Na rozdíl od jazyku Java, který běží na virtuálním stroji JDM (Java Development Machine), si společnost Google vytvořila svůj virtuální stroj Dalvik speciálně pro Android. Dalvik se zaměřuje na optimalizaci Java kódu, aby byl použitelný v přenositelných zařízeních.[1]

V současné době existují dva primární programovací jazyky pro Dalvik – Java a Kotlin. Java byla podporována od samotného začátku Androidu a stále je nejrozšířenějším jazykem pro tento účel. Oproti tomu Kotlin se začal používat pár let zpátky a je ještě novinkou.[34]

V roce 2019 stojí za zmínku dva nejrozšířenější mobilní operační systémy. Jedním je Android, který masivně převažuje se svými 75 procenty v zastoupení na trhu. Druhým největším operačním systémem je iOS.[30]

To znamená, že aplikace bude vyvíjena pro nejrozšířenější operační systém pro mobilní platformy. Android má poměrně dlouhou historii a je zapotřebí brát v potaz kompatibilitu

i se staršími verzemi. Zároveň existuje mnoho vydavatelů, kteří si Android upraví, a tak je těžce proveditelná bezchybná podpora pro všechna zařízení. Světlá stránka rozšířenosti je, že je vývoj podporován a není složité najít dokumentace a obecně informace o Androidu.

2.3 Význam slova minimalistický

Slovo minimalistický v tomto kontextu může znamenat úplně něco jiného, než by si průměrný čtenář představil. Jako první věc pravděpodobně každého napadne vzhled. Co nejméně nepodstatných informací na obrazovce. Z programátorského hlediska by se dal tento termín nahradit souslovím „co nejvíce optimalizovaný“. Kromě uživatelského rozhraní se však v této souvislosti musí počítat i s dalšími aspekty, které je třeba udržet pokud možno optimální. Mezi ně patří zejména:

- velikost,
- spotřeba paměti a energie,
- verze Android API[1],
- maximální využití obrazovky a
- čitelnost i při velkém počtu informací o zobrazených objektech a akcích.

Z tohoto vyplývá, že aplikace nebude vyvíjena v nejnovějších, propracovaných technologiích, které jsou náročné na všechna zmíněná hlediska. Uživatel by neměl být odrazen vysokým počtem nepodstatných informací a funkcí. Oproti tomu je ale potřeba, aby program podporoval všechny funkce, které jsou od správce souborů a aplikací očekávány.

2.4 Rozbor současných produktů

Ještě před návrhem bude nezbytné zanalyzovat aktuální zajímavá řešení a zamyslet se nad jejich nedostatky a po důkladném rozboru zjistit, jakým způsobem jsou tyto aplikace pojaty. Rozbor může sloužit k případné inspirace při práci. Hlavní pozornost bude věnována známým aplikacím, které mají různé vlastnosti. Některé produkty jsou moderní a snaží se o co nejlepší interakci s uživatelem, některé zase neřeší vizuální stránku a jen splňují požadované vlastnosti. Novodobé aplikace jsou spíše prvního typu. Mají všelijakou nepotřebnou funkcionalitu a rozhraní obsahuje mnoho elementů a někdy i reklamy.

Cílem této práce nebude vytvořit aplikaci, která bude následovat tyto trendy, ale spíše naopak. Určitě existuje mnoho uživatelů, kteří k prohlédnutí paměti ve svém telefonu nebo ke spuštění aplikace nechtějí stahovat programy, v nichž se nachází nespočetné množství funkcí, které pravděpodobně zůstanou nevyužity. V dnešní době už sice většina lidí pravděpodobně vlastní zařízení s obrovskou kapacitou úložiště, ale i přesto se najdou tací, jenž jsou za každý ušetřený bajt vděční. Rozhodně ale existují i řešení, která jsou optimalizovaná a minimalistická a ta budou také předmětem tohoto rozboru.

Android ve svých novějších verzích (6.0 a výše) již poskytuje vestavěného správce souborů. U starších verzí tomu tak nebylo, ale většinou se o nějakého implicitního správce postaral výrobce telefonu či distributor. Kromě implicitních aplikací tohoto typu se jich na

[1] API - Application Programming Interface (rozhraní pro programování aplikací)

trhu nachází spousta. Některé z nich stojí za povšimnutí buď díky své unikátnosti, nebo dobrému hodnocení.[35]

Rozebrána budou dvě různá řešení, která obsahují jak správce aplikací, tak souborů. Obě aplikace jsou od sebe velmi odlišné.

ES File Explorer

ES File Explorer je momentálně jedním z nejlépe hodnocených správců souborů. Stažen byl již téměř třemi sty milióny uživatelů. Kromě obvyklých funkcí disponuje například velmi zajímavým grafickým uživatelským rozhraním, svými textovými editory, možností vyčištění paměti nebo podporou FTP.[9]

Po nainstalování se může zdát velmi kvalitně zpracovaný. Nechybí žádná základní funkcionalita a vypadá na první pohled zajímavě. Avšak pokud si uživatel při prvním použití chce zkopírovat soubor, může trvat poměrně dlouhou dobu, než se zorientuje. Nachází se zde nespočet různých prvků uživatelského rozhraní, které činí orientaci v aplikaci složitou. Ojediněle také vyskočí okénko, které žádá o zhodnocení aplikace, nebo dokonce reklama. Tato vyskakovací okna se dají zrušit, ale za poplatek. Toto je uživatelsky nepřípustné. Aplikace v této práci bude cílit na nevyrušování uživatele při jeho práci. Samozřejmě zdarma.

Dalším krokem v průzkumu je zkontrolování požadovaných povolení a využití systémových zdrojů. Aplikační archív APK má velikost bezmála 17 MB. Nainstalovaná aplikace v zařízení samotná zabírá 45 MB v úložišti. Toto číslo naroste, pokud k němu přičteme velikost vyrovnávací paměti a uživatelských dat. Paměti RAM využívá jen po spuštění kolem dvou set megabajtů. Co se týká oprávnění, dohromady je jich vyžadováno 36, z toho 5 z kategorie nebezpečných.

Tato aplikace cílí na vizuální reprezentaci dat uživateli. Snaží se následovat trendy a v důsledku zahlcuje uživatele neskutečným množstvím informací. Pro starší zařízení je nepoužitelná kvůli systémovým nárokům. Pokud zařízení nepodporuje sekundární externí úložiště, je pravděpodobně, že kvůli nedostatku místa si ji ani uživatel nebude moci nainstalovat. Počet vyžadovaných oprávnění je absurdní. Tato práce se bude snažit od těchto problému vzdalovat.

Total Commander

Total Commander má dlouhou historii nejen pro mobilní platformy. Vyskytuje se v několika verzích pro starší i pro novější zařízení. Disponuje jednoduchým rozhraním, je lehce pochopitelný a neobsahuje žádné reklamy a žádná vyskakovací okna. Podporuje veškerou základní funkcionalitu a některé funkce navíc.[25]

Tato aplikace má hlavní rozhraní, které je na první pohled pochopitelné. Bohužel přece jenom ale není ideální z hlediska minimálnosti. Má mnoho postranních tlačítek, jejichž význam nemusí být na první pohled zřejmý. Dále poskytuje dvě okna, což je dobrá funkce, ale vyžaduje zbytečné zdroje. Samotné položky správců potom obsahují obrázky, které slouží k vizuálnímu popisu položky. Toto všechno ubírá na jednoduchosti. Správce aplikací zase nezobrazuje některé důležité informace, které by mohly uživateli přijít vhod.

Velikost archívu je něco před 2 MB. Počet vyžadovaných oprávnění je 12.

Oproti předchozí aplikaci je vidět, že je tento správce mnohem méně náročný na systém. Total Commander může sloužit jako inspirace při této práci. Přesto však není minimalistický. Aplikace může být ještě optimalizovanější a menší.

Shrnutí

Je zřejmé, že řešení existují různá. Dvě rozebraná konkrétní řešení jsou zástupci jiných přístupů. První správce následuje trendy a nehledí na minimalizaci. Druhá aplikace se jeví jako dobrý příklad optimalizovaného správce souborů a aplikací. Na vyjmenovaná řešení bude brán ohled při návrhu a implementaci. Od některých aspektů se bude tato práce odvracet a některými se zase bude inspirovat.

Kapitola 3

Použité technologie a aplikační rozhraní

Tato kapitola se věnuje použitým technologiím a API při řešení. Hlavním obsahem bude samozřejmě programování v systému Android. Před vývojem aplikace je podstatné nastudovat aplikační rozhraní, které souvisí s tématem práce.

3.1 Základy vývoje pro Android

Při studiu základů programování pro operační systém Android bylo využito doporučené literatury[33]. Tato literatura poskytla všeobecný přehled o problematice a pomohla při začátku studia.

Je potřeba se seznámit s aplikačním rozhraním, které bude v projektu použito. Jak již bylo zmíněno, vývoj aplikační části může probíhat zejména v jazycích Java nebo Kotlin. Kromě aplikační části je využito jiných jazyků, například grafické rozhraní se definuje v jazyce XML[1]. Jakmile je zdrojový kód napsán, společně i s ostatními daty a zdrojovými soubory se pomocí nástrojů Android SDK vše zkompiluje do souboru APK[2], což je komprimovaný archívní soubor s příponou .apk. Tento soubor obsahuje všechno potřebné pro instalaci na zařízení s Androidem.[5]

Android SDK[3] je sada vývojových nástrojů, které jsou použity k vývoji aplikací pro platformu Android. Android SDK obsahuje mimo jiné potřebné knihovny, debugger, emulátor reálného zařízení, dokumentaci apod. S každou novou verzí Androidu bývá zveřejněn odpovídající SDK, který si musí vývojář pořídit, aby mohl psát programy s nejnovějšími prvky. SDK bývá často integrováno do vývojového prostředí s grafickým rozhraním.[4]

Každá aplikace má v systému Android svoje vlastní prostředí. Běží na vlastním procesu v izolaci na virtuálním stroji a na jádru Linuxu. Standardně má přístup jen ke komponentám nezbytně nutným k vykonání její práce a nic více. Pro přístup k ostatním částem systému potřebuje povolení.[5]

3.1.1 Komponenta aktivita

Komponenty aplikace jsou nezbytné části Android aplikace. Každá komponenta je vstupním bodem, skrz nějž se systém nebo uživatel dostane do aplikace. Typy komponent existují čtyři

[1] XML – Extensible Markup Language (značkovací jazyk)
[2] APK – Android package (Android balík)
[3] SDK – Software Development Kit (sada vývojových nástrojů)

– aktivita, služba, broadcast reciever a content provider[5]. V této práci bude využita jen komponenta **aktivita**, a proto je potřeba ji důkladně rozebrat.

Aktivita je vstupním bodem pro interakci s uživatelem. Reprezentuje jednotlivé obrazovky a uživatelské rozhraní (obecně jedna aktivita = jedna obrazovka). Například pro správce souborů může být jedna obrazovka seznam souborů v aktuálním adresáři nebo třeba obrazovka s nastavením. Každá aktivita je nezávislá na ostatních, ale může s nimi komunikovat a tímto dohromady tvoří uživatelské rozhraní celé aplikace. Zároveň jedna aplikace může spustit i jinou. Třeba při otevírání souboru reprezentující obrázek se otevře aplikace umožňující zobrazení obrázků.[5]

Aktivity mají svůj životní cyklus s několika stavy. Třída `Activity` poskytuje metody, které umožňují aktivitě poznat, že se stav změnil. Díky dobré implementaci se dají podchytit nežádané situace jako pád aplikace.[16]

onCreate() Povinná metoda, spuštěna, když systém vytvoří aktivitu. Měla by obsahovat inicializace podstatných částí dané aktivity. Například načítání dat do seznamů. Také se v ní musí zavolat metoda `setContentView()` na definování rozložení pro uživatelské rozhraní.

onStart() Jakmile skončí předchozí metoda, aktivita vstoupí do stavu „nastartovaná“ a bude viditelná uživatelem. Toto zpětné volání obsahuje závěrečné přípravy, aby se mohla aktivita dostat do popředí a stát se interaktivní.

onResume() Tato metoda je vyvolána těsně před zahájením interakce s uživatelem. Aktivita je v tomto okamžiku na vrcholu zásobníku aktivit a zachycuje uživatelské vstupy. Zde je implementována většina základní funkčnosti. Nyní aktivita běží.

onPause() Když aktivita ztratí fokus, dostane se se do stavu „pozastavená“ a vyvolá toto metodu. Znamená to, že aktivita je částečně viditelná, ale slouží jako znamení uživateli, že bude brzy opuštěna a dostane se do stavu „zastavená“ nebo „obnovená“. V tomto stavu může stále být aktualizováno uživatelské rozhraní.

onStop() Volána, pokud se aktivita ztratí z viditelnosti. Může se tak stát, protože je právě ničena, startuje nová aktivita, nebo jiná existující vstupuje do „obnovená“ a překrývá tuto.

onRestart() Spustí se, jestliže aktivita ve stavu „zastavená“ se chystá startovat znovu. Obnoví stav aktivity z toho momentu, kdy byla zastavena.

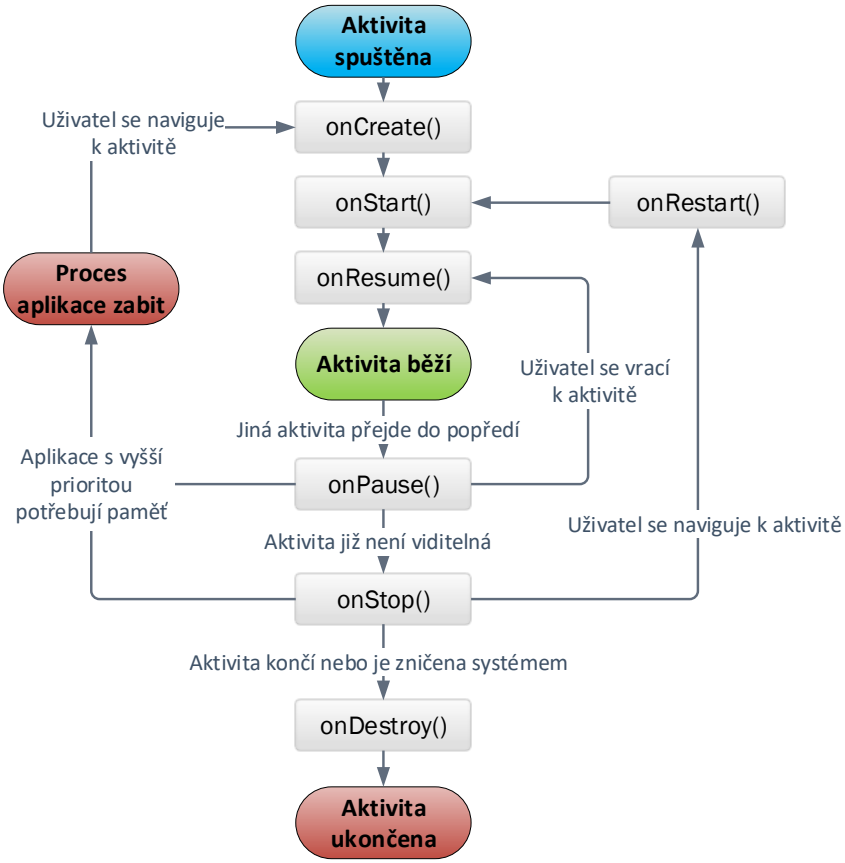
onDestroy() Poslední zpětné volání předtím, než je aktivita zničena. Tady přichází na řadu uvolňování alokovaných zdrojů.

Lepší přehled vzájemné interakce těchto stavů a zpětných volání zobrazuje obrázek 3.1.

Intent

Aby bylo možné aktivitu spustit, musí být nejprve aktivována. K tomuto účelu slouží asynchronní zpráva zvaná **intent**. Intent svazuje jednotlivé aktivity mezi sebou za běhu aplikace.

[5] Převzato a upraveno z https://developer.android.com/guide/components/activities/activity-lifecycle.html



Obrázek 3.1: Životní cyklus základní komponenty aktivity. Sestává se z několika stavů a systémových volání, které jsou po dobu interakce s aktivitou vystřídány a zavolány.[5]

Intenty mají schopnost vyžádat akci od ostatních komponent, ať náleží stejné či jiné aplikaci. Intent je vytvořen pomocí třídy `Intent`, při jejíž instanciaci se blíže specifikuje jeho typ. Pro aktivity definuje intent akci, která má být provedena (například něco zobrazit nebo poslat) a může specifikovat typ dat nebo jiných věcí, které může startovaná komponenta potřebovat.[5]

Intent může být dvou různých typů – explicitní nebo implicitní. U explicitního intentu se musí zadat cílová komponenta, v čemž se liší od implicitního, kterému se určí, jaká akce se má po spuštění stát, případně se připojí i nějaká data. Implicitní intent umožní systému, aby sám našel vhodnou komponentu, která dokáže akci vykonat a spustit. Pokud je takových komponent více, uživatel dostane nabídku validních intentů, ze kterých si vybere.[5]

3.1.2 Android Manifest

Předtím, než má systém Android možnost spustit některou komponentu, musí mít informaci o tom, že ta komponenta existuje. K tomuto účelu slouží soubor manifest, `AndroidManifest.xml`. Každá aplikace musí tento soubor obsahovat v kořenové složce projektu a

deklarovat v něm všechny komponenty. Systém má k tomuto souboru přístup a na základě jeho obsahu ví o každé aplikaci podstatné informace.[6]

Kromě komponent deklaruje manifest i jiné věci:

- uživatelská oprávnění, které aplikace vyžaduje,
- minimální, cílovou a kompilační verzi API aplikace,
- hardwarové a softwarové vlastnosti vyžadované aplikací (kamera, bluetooth, atp.) a
- vyjmenovává externí knihovny, se kterými se musí aplikace propojit.

Deklarace komponent

Primárním úkolem je tedy informování systému o tom, které komponenty se v aplikaci vyskytují. Pokud je některá z vytvořených aktivit vytvořena, ale není zde deklarována, stává se pro systém neviditelná a nemůže se tak nikdy spustit ani být spuštěna. Deklaruje se pomocí odpovídajících elementů, pro aktivity je jím `<activity>`. V těchto elementech se definují tzv. *filtry intentů* pomocí elementu na nižší úrovní s názvem `<intent-filter>`. Právě díky těmto filtrům se systém dozví, jaké intenty s jakými typy může daná aktivita přijímat, viz 3.1.1.[6]

Oprávnění

Účelem oprávnění je ochránit soukromí uživatele. Aby mohla aplikace přistupovat k citlivým datům uživatele nebo k systémovým funkcím, musí si vyžádat oprávnění. Podle důležitosti funkce dává systém povolení automaticky nebo vyzve uživatele, aby ho schválil. Žádná aplikace nemá standardně oprávnění provádět operace, které by mohly mít dopad na ostatní aplikace, systém nebo uživatele. Toto zahrnuje čtení a zápis soukromých dat uživatele nebo dat jiných aplikací, přístup k internetu anebo třeba udržování zařízení zapnutém stavu.[21]

Oprávnění se rozdělují podle nebezpečnosti do několika kategorií. V manifestu se tato oprávnění nastavují elementem `<uses-permission>`, kde se v atributu definuje jeho typ. Zde musí být všechna oprávnění, nehledě na nebezpečnost. Programátor implementuje žádost o povolení pro nebezpečná ve formě výzvy za běhu aplikace.[21]

Deklarace požadavků aplikace

Existuje mnoho různých zařízení s operačním systémem Android s různými verzemi a schopnostmi, viz 3.3. Aby se zabránilo nainstalování na zařízení, které není uzpůsobené dané aplikaci, se v manifestu deklarují požadavky. Toto je důležité zejména na určení minimální a cílové požadované verze SDK (a tedy i API) a potvrzení potřebných funkcionalit jako například kamera.

U deklarování použitého SDK jsou povinné dva atributy na určení verze API – `minSdkVersion` a `targetSdkVersion`. První z nich deklaruje minimální možnou verzi SDK, u které by měla být zaručena plná funkcionalita. Druhý atribut zase určuje verzi, na které byla aplikace testována a není-li deklarován, implicitně se uvažuje stejná hodnota jako u prvního. Aplikace by měla tedy podporovat všechny verze SDK od minimální verze po cílovou.[26]

Pokud by se například nastavila minimální verze na 10 a cílová na 20, uživatel by mohl předpokládat bezchybný chod na verzích 10 až 20. Kdyby si však aplikaci nainstaloval na

mobilní telefon s nepodporovaným SDK, pravděpodobně by mu aplikace nefungovala vůbec nebo s chybami. Více v sekci o verzích API 3.3.

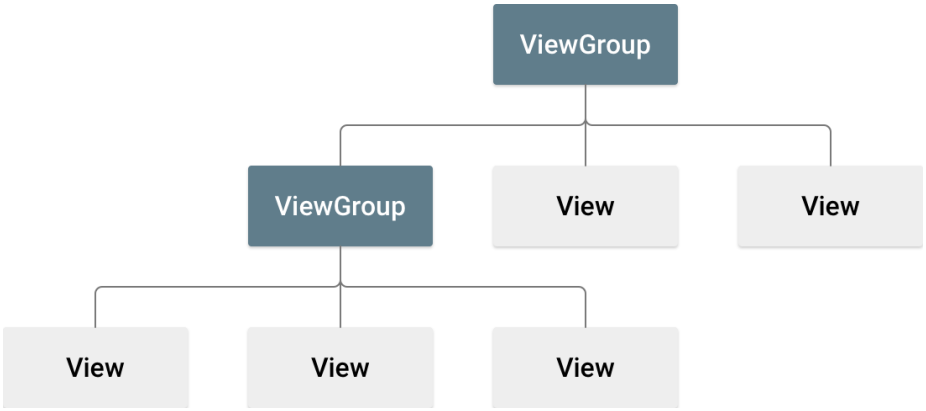
3.1.3 Zdroje

Kromě kódu se aplikace pro operační systém Android skládá i ze zdrojů. Zdroje jsou například obrázky, texty, zvuky a další soubory v souvislosti s vizuální prezentací. Tyto zdroje definují animace, styly, menu, barvy a rozložení prvků uživatelského rozhraní u aktivit. Každý zdroj je pak pomocí SDK použitelný v kódu a lze se na něj odkázat. Zdroje umožňují jednoduchou implementaci kompatibility.[5]

Využití zdrojů v této práci bude hlavně v podobě definování rozložení.

Rozložení

Rozložení definuje strukturu pro uživatelské rozhraní v aplikaci, například v aktivitě. Elementy jsou skládány v hierarchii pomocí objektů **View** a **ViewGroup**. View většinou vykresluje přímo interaktivní prvek, přičemž druhý zmiňovaný objekt je neviditelný kontejner a definuje strukturu pro další objekty. Obrázek 3.2 ilustruje příklad hierarchie.[17]



Obrázek 3.2: Ilustrace hierarchie objektů definujících rozložení uživatelského rozhraní.[6]

Tato rozložení samo o sobě nemá žádnou funkcionalitu a obsah je vyplněný jen staticky. Mohou být definována právě v souboru se zdroji nebo je lze vytvářet a přiřazovat dynamicky za běhu aplikace. Nemá-li dané rozložení předem definovaný obsah, ale má být přidán dynamicky, použije se třída `Adapter`. Pro interakci s uživatelem se použije třída `Listener`.[17]

Adapter

Jak již bylo zmíněno, pokud musí být obsah dynamický, použije se rozložení založené na třídě AdapterView. Tato třída totiž může využívat právě Adapter na propojení funkcionality programu s uživatelským rozhraním. Adapter se chová jako prostředník mezi zdrojem dat a rozložením – Adapter načítá data a konvertuje každý záznam do instance třídy View, která může být přidána do rozložení. Příkladem tříd odvozených od AdapterView můžou být ListView, Spinner nebo GridView.[17]

[6] Převzato z https://developer.android.com/guide/topics/ui/declaring-layout

Listener

Listener je rozhraním ve třídě View, které obsahuje jednu metodu. Tato metoda bude volána Androidem, pokud uživatel vyvolá interakcí danou akci s položkou View, ke kterému je Listener registrován. Tyto metody se liší od typu View. U tlačítka je taková metoda například `onClick()`, která je vyvolána při kliknutí na tlačítko. U textového pole zase příkladem může být metoda `onKeyDown(int, KeyEvent)`, která je spuštěna, pokud uživatel stiskne určité písmenko na klávesnici. V dané metodě se potom implementuje reakce na danou akci.[15]

3.2 Přehled úložných prostorů

Pro přístup k souborům bude potřeba nastudovat úložné prostory v zařízeních s Androidem. Android poskytuje několik možností, jak ukládat data. Existuje několik typů úložných prostorů, které se liší v několika aspektech. Zejména jsou jimi přístupnost a velikost. K ukládání souborů v Androidu se vyskytují následující možnosti:

- Interní úložný prostor.
- Externí úložný prostor.
- Shared preferences.
- Databáze.

Kromě některých typů souborů v externím úložišti jsou všechny tyto možnosti určené pro privátní data aplikace – k těmto datům nemají ostatní aplikace přístup. Pokud chce vývojář sdílet soubory s ostatními aplikacemi, měl by použít API `FileProvider`.[8]

3.2.1 Interní

Ve výchozím stavu jsou soubory uložené na interním úložišti přístupné pouze aplikací, která je vlastní a ostatní aplikace ani uživatel bez root přístupu k nim nemohou přistupovat. Toto dělá z interního úložného prostoru ideální místo pro interní data aplikace, ke kterým nepotřebuje uživatel přistupovat. Systém poskytuje soukromý adresář pro každou aplikace, v němž uschovává svoje soubory. Jakmile uživatel odinstaluje aplikaci, všechny tyto soubory budou smazány.[8]

3.2.2 Externí

Všechna zařízení s Androidem podporují sdílené externí úložiště. Tento prostor se nazývá externí, jelikož není garantováno, že bude dostupná. Je to totiž úložný prostor, jež mohou uživatelé připojit k počítači jako externí zařízení, nebo dokonce odpojit od zařízení úplně (například SD karta). Zde uložené soubory mohou být zobrazené, čtené i upravované všemi uživateli. Před přístupem k tomuto prostoru by se měla zkontrolovat jeho dostupnost a práva. Nejčastěji zde jsou uložena data, která mají být přístupna uživateli a ostatním aplikacím a nebudou smazána při odinstalaci.[8]

Výjimku tvoří adresář specifický pro danou aplikaci, který je právě naopak smazán při odinstalaci. Tento adresář slouží jako užitečná alternativa k interní paměti, pokud na ní dochází místo. Samozřejmě ale není zaručená neustálá přítomnost externí paměti, takže se

na to musí brát ohled. Pořád to je ale externí úložiště, a proto se musí počítat s přístupností odevšad.[8]

Pro přístup k externímu úložišti musí být povolena oprávnění.

3.2.3 Shared preferences

Další možností, jak ukládat data, je API zvané Shared preferences pomocí třídy `SharedPreferences`. Toto API umožňuje čtení a zápis perzistentních dat ve tvaru klíč-hodnota datových typů: boolean, float, int, long a string. Všechny uložené hodnoty jsou zapsány do souborů typu XML, které přetrvají přes uživatelská sezení.[8]

3.2.4 Databáze

Android podporuje databáze typu SQLite. Všechny vytvořené databáze jsou přístupné pouze v rámci dané aplikace.[8]

3.3 Aplikační rozhraní

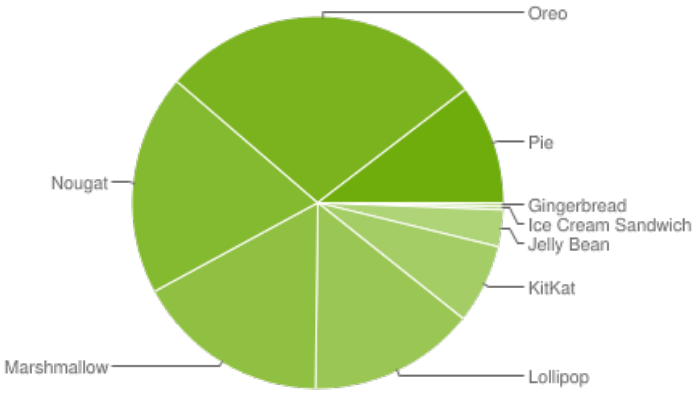
Operační systém Android se neustále rozvíjí a vylepšuje. Každé verzi Androidu je přiřazena úroveň API. API je balík knihoven a tříd, které může vývojář využívat a ovládat či komunikovat se softwarem. Úroveň API v Androidu je celočíselná hodnota, která jednoznačně identifikuje API nabízenou verzí platformy Android. Android poskytuje rozhraní a knihovny, které mohou aplikace používat pro interakci se systémem Android. API Androidu obsahuje několik položek:

- Základní balíky a třídy.
- XML pro deklarování souboru manifestu 3.1.2.
- XML pro deklaraci a přístup ke zdrojům 3.1.3.
- Intenty 3.1.1.
- Oprávnění.

Každá následující verze platformy může obsahovat aktualizace, které zprostředkovává. Aktualizace API jsou navrhovány tak, aby nová API zůstávala kompatibilní s dřívějšími verzemi. To znamená, že většina změn přidává novou nebo nahrazuje starou funkcionalitu. Při nahrazování jsou staré verze označeny za zastaralé, ale nejsou odebrány, aby mohly být využívány existujícími aplikacemi.

S každou novou verzí Androidu vychází i nová verze API. Každá verze Androidu označuje právě jednu úroveň API, ačkoli podpora je implicitní pro všechny dřívější úrovně. Na obrázku 3.3 můžeme vidět aktuální rozložení verzí Androidu na trhu. Z obrázku je zřejmé, že většina momentálně působících zařízení běží na několika nejnovějších verzích. V době psaní práce byla nejmenší verze Androidu přesahující podíl 0,1 % verze Gingerbread se svými 0,3 %, což reprezentuje Android 2.3.3 až 2.3.7 a API úroveň 10. Největší zastoupení má verze s názvem Oreo s 28,3 %. Oreo označuje Android 8.0 (API úroveň 26) a 8.1 (API úroveň 27).[26]

Z definice se dá usoudit, že čím větší verze, tím větší velikost knihoven a naopak. Proto bude v této práci podstatné použít co nejmenší verzi, aby se nesnížil počet podporovaných



Obrázek 3.3: Rozložení verzí Androidu na trhu v době psaní práce. Čím je tmavější část, tím je novější verze Androidu. Verze s menším podílem než 0,1 % nejsou zobrazeny.[8]

zařízení. V oficiální dokumentaci Androidu je u každé položky uvedeno, ve které úrovni API byla přidána.

3.3.1 API pro přístup k souborům

Správa souborů je možná využitím API jak jazyka Java tak samotného Androidu. V této části budou popsána aplikační rozhraní, která souvisí s přístupem k souborům a budou využita při práci nebo zmíněna dále v textu této práce.

File

Android používá souborový systém podobný jiným souborovým systémům založených na discích. Pro manipulaci se soubory a složkami je určené API zvané **File**[10], specifičtěji **java.io.File**.

Třída `File` je abstraktní reprezentací souborových a adresářových cest. Uživatelské rozhraní a operační systém používají názvy cest závislé na systému. Tato třída představuje abstraktní, systémově nezávislé zobrazení hierarchických cest. Abstraktní cesta má dvě složky:

1. Volitelnou předponu jako například specifikátor diskové jednotky – "/"pro kořenový adresář systémů založených na UNIX a
2. posloupnost nula nebo více řetězců názvů.

První řetězec v abstraktní cestě je v Androidu název adresáře. Každý následující název v abstraktní cestě označuje adresář a poslední označuje buď adresář nebo soubor. Převod cesty do nebo z abstraktní cesty je závislý na systému. Když je abstraktní cesta převedena na název cesty, všechna jména jsou od sebe oddělena znakem oddělovače. Znak oddělovače se také liší systém od systému a je dostupný prostřednictvím statické proměnné této třídy.

Cesta, ať už abstraktní nebo ve tvaru řetězce, může být absolutní či relativní. Absolutní cesta je kompletní v tom smyslu, že žádné jiné informace nejsou potřeba k lokalizaci

[8] Převzato z https://developer.android.com/about/dashboards

souboru, který označuje. Relativní musí naopak být interpretována pomocí informací z jiných cest. Třídy v balíku API `java.io` pracují ve výchozím nastavení pouze s relativními cestami.

Z abstraktní cesty lze zjistit několik informací. Například rodičovskou složku aktuálního souboru zjistí metoda `getParent()`. Ta složka se pak sestává z předpony a všech názvů v posloupnosti kromě posledního. Absolutní cesta každého adresáře je předkem libovolného objektu File s absolutní abstraktní cestou, která začíná absolutní cestou adresáře. Například cesta „/sdcard/0/“ je předkem složky označené cestou „/sdcard/0/DCIM/Camera“.

Koncept předpon je využit k zjišťování kořenových složek na Androidu. Předpona každé absolutní cesty je vždy „/“. Relativní cesty nemají žádnou předponu. Abstraktní cesta označující kořenový adresář má předponu právě „/“ a prázdnou posloupnost názvů.

Instance této třídy mohou i nemusí označovat skutečný objekt souborového systému jako soubor nebo adresář. Pokud takový objekt označuje, pak se tento objekt nachází v tzv. *oddílu*. Oddíl je část úložiště souborového systému. Jeden úložné zařízení může obsahovat více oddílů. Objekt, pokud existuje, bude umístěn na oddílu, který je pojmenován některým předkem cestou v podobě jeho absolutní cesty.

Souborový systém může implementovat nad skutečným objektem přístupová oprávnění, vlastně omezení některých operací, jako například čtení, psaní a spuštění.

Manipulace se soubory

Ostatní metody u objektů typu File určeného abstraktní cestou důležité v této práci jsou následující:

- `createNewFile()` vytvoří nový prázdný soubor jen pokud takový soubor ještě neexistuje
- `delete()` smaže soubor nebo adresář
- `getAbsoulteFile()`, `getAbsoultePath()` vrátí objekt File nebo absolutní cestu
- `getFreeSpace()` a `getTotalSpace()` získají aktuální volné nebo celkové místo v úložišti
- `getName()`, `getPath()` vrátí jméno nebo abstraktní cestu souboru/adresáře
- `getParentFile()`, `getParent()` vrátí objekt File nebo cestu nadřazeného adresáře, pokud takový existuje
- `isDirectory()` a `isFile()` otestují, zda se jedná o adresář nebo soubor
- `isHidden()` otestuje, zda se jedná o skrytý soubor
- `lastModified` vrátí datum poslední modifikace
- `length()` vrátí velikost souboru
- `list()`, `listFiles()` pokud je objekt typu adresář, potom vrátí pole všech potomků ve tvaru cest nebo objektů File, vrácené pole může být filtrováno předáním filtru jako parametr těmto funkcím
- `mkdir()`, `mkdirs()` vytvoří nový adresář, u druhé z těchto metod také všechny neexistující rodičovské

- `renameTo()` přejmenuje soubor

Třída `File` však neobsahuje metody k jednoduchému kopírování a přesouvání souborů.

Files

Od úrovně API 26 může být k velmi jednoduché manipulaci se soubory použita třída `java.nio.file.Files`[14]. Některé z metod jsou podobné předchozím, ale objevují se i nové:

- `copy()` zkopíruje soubor
- `lines()` přečte všechny řádky souboru
- `move()` přesune soubor
- `readAllBytes()` přečte všechny byty souboru
- `write()` zapíše řádky nebo byty do souboru

Díky této třídě nemusí být některé elementární operace implementovány.

Storage Access Framework

Android 4.4 (API úroveň 19) zavádí Storage Access Framework (SAF, framework pro přístup k úložišti). SAF zjednodušuje procházení a otevírání dokumentů, obrázků a ostatních souborů poskytovatelů úložišť (např. Google Drive) pomocí intuitivního rozhraní. Sestává se ze tří částí. **Document provider** odvozená od komponenty content provider, která umožňuje zobrazení úložiště služby poskytovatele. **Klientská aplikace** implementující intent, pomocí něhož se zavolá daná služba a zpracuje se výsledek. A poslední **Picker** – systémové rozhraní dovolující uživateli přístup k souborům.[19]

Další systémové metody

Protože přesné lokace, kde mohou být soubory uloženy, se mohou lišit mezi různými zařízeními, existuji metody pro přístup k interním i externím cestám namísto použití absolutních cest, pomocí kterých se docílí lepší rozšířitelnosti.

Jak bylo zmíněno v předchozí kapitole, úložiště se dělí na interní a externí. U starších zařízení se většinou interní pamětí myslela paměť telefonu a externí nějaké vyjímatelné médium. Nyní však většina zařízení má jednu paměť rozdělenou na interní a externí části. Objevují se také mobilní telefony, které kromě vnitřní paměti rozdělené na dvě obsahují slot na další média nebo se k nim dá připojit USB.

API pro přístup k souborům poskytuje několik metod souvisejících s úložišťmi.

Pro interní paměť existuje několik metod na získání cest. Například metoda `getFilesDir()` vrací objekt File reprezentující kořenový adresář aplikace v interním úložišti. Nic jiného zde není potřeba, protože interní úložiště je dostupné neustále a aplikace má přístup ke svým souborům. Aplikace nemůže za normálních okolností získat přístup k jiným souborům než svým vlastním v interním úložišti. Plný přístup k internímu úložišti se dá získat pomocí rootnutí zařízení.

Cesty pro externí úložiště jsou složitější na určení kvůli možnosti absence média. Navíc se musí myslet na rozdíly mezi novými a staršími zařízeními. Proto je potřeba zařídit několik věcí: získat oprávnění, zjistit dostupnost a získat cesty.

Získání oprávnění

K zápisu do veřejného externího úložiště musí být vyžádáno oprávnění **WRITE_EXTERNAL_STORAGE**. Toto oprávnění patří mezi nebezpečné, více na 3.1.2. Stačí když aplikace používá toto oprávnění a systém automaticky počítá i s oprávněním ke čtení externího úložiště. Pro čtení se musí zaregistrovat oprávnění **READ_EXTERNAL_STORAGE**. Od verze Androidu 4.4 (API úroveň 19) není potřeba pro zápis ani čtení do soukromého adresáře aplikace žádné oprávnění. Od verze Androidu 6.0 (API úroveň 23) musí být oprávnění k zápisu vyžádáno za běhu aplikace.

Vyžádání oprávnění za běhu aplikace od Androidu 6.0

Na všech verzích Androidu musí být deklarována oprávnění v manifestu. Poté záleží na nebezpečnosti oprávnění. Normální oprávnění jsou udělena po instalaci, na nebezpečná je uživatel tázán.

Aplikace musí zkontrolovat, zda má potřebná oprávnění pokaždé, když chce provést operaci požadující dané oprávnění. Uživatel má totiž možnost povolení kdykoli odvolat. Pro kontrolu oprávnění existuje metoda `ContextCompat.checkSelfPermission()`, jejíž parametrem je právě oprávnění, které má být zkontrolováno. Pokud aplikace oprávnění má, funkce vrátí `PERMISSION_GRANTED`, jinak `PERMISSION_DENIED`.[23]

V druhé situaci se pak musí oprávnění vyžádat pomocí metody `requestPermissions()`. Této metodě se předá pole požadovaných oprávnění jako parametr. Běží asynchronně a po návratu je nutné zpracování uživatelovy volby v metodě `onRequestPermissionsResult()`.[23]

Zjištění dostupnosti

Pro zjištění dostupnosti existují metody `getExternalStorageState()`. Tato metoda vrací stav externího zařízení. Například stav **MEDIA_MOUNTED** znamená, že čtení i zápis je povolen.

Získání cesty

Programátor má možnost uložit privátní soubory aplikace také do externího úložiště. Adresář se lokalizuje metodou `getExternalFilesDir()`. Je potřeba si uvědomit, že složka je dostupná ostatními aplikacemi, nemusí být přístupná a je smazána při odinstalaci.

Od Androidu 4.4 (API úroveň 19) mají některá zařízení více externích slotů. Pro výpis všech soukromých cest na všech externích zařízení se použije metoda `getExternalFilesDirs()`. K získání kořenové složky externího úložiště

Poslední metoda týkající se získání cesty je metoda `getExternalStorageDirectory()`, která vrací kořenovou složku primárního externího úložiště.[24]

Otevírání souborů

Kromě procházení musí správce umět také soubory otevírat. Soubory mohou a nemusí obsahovat příponu. Přípona je identifikátor použitý jako přípona k názvu souboru v operačních systémech. Příponu lze považovat za metadata. Pomáhá operačnímu systému porozumět charakteristikám souborů a do jisté míry jejím zamýšleným použitím. Její důležitá role je, že umožňuje operačnímu systému vybrat vhodnou aplikaci, ve které bude soubor spuštěn.[11]

Pomocí jména souboru a zejména jeho přípony se dá určit také typ MIME[9]. MIME je originálně internetový standard, který pomáhá rozšířit možnosti emailu povolením vkládání jiných než čistě textových elementů. Zároveň však je MIME použit k určení typu souboru.[18]

MIME se používá společně s URI [10] v intentu při otevírání souboru. URI je řetězec znaků, který jednoznačně identifikuje konkrétní zdroj. Aby byla zajištěna jednotnost, URI dodržují několik pravidel syntaxe. Pro zdroj typu soubor je syntaxe **file://host/path**, kde **file** je povinná předpona, **host** je plně kvalifikované doménové jméno systému, na kterém je cesta dostupná a **path** je hierarchická cesta adresářů končící jménem daného souboru.[29]

API obsahuje třídu `Uri` a její metodu `fromFile()`, která vytvoří URI z objektu File. Z URI se prostřednictvím metody `getType()` získá typ MIME.

FileProvider

Počínaje verzí Androidu 7.0 (API úroveň 24) jsou URI typu **file://** mimo rámec aplikace zakázány kvůli změnám v oprávněních. Ke sdílení souborů mezi aplikacemi se nově používá URI typu **content://** a mělo by být udělené dočasné oprávnění pro přístup k souborům. K tomuto slouží třída `FileProvider`.[2]

FileProvider je speciální podtřída komponenty content provider, která umožňuje bezpečné sdílení souborů vytvářením Uri typu **content://** místo **file://**. Změny jsou zejména v oprávnění. Metoda `getUriForFile()` této třídy vrací právě URI druhého typu. Dočasná oprávnění se nastaví u intentu použitého pro otevírání souborů.[13]

3.3.2 API ke správě aplikací

Správa aplikací je více přímočará. Ke zjišťování informací o nainstalovaných aplikacích slouží API `android.content.pm`. Toto API obsahuje třídy pro zjišťování informací o aplikačních balících včetně informací o jejich aktivitách, oprávněních a dalších komponent. Většina informací o těchto balících je definována v aplikačním manifestu (viz sekce 3.1.2).

Následující třídy budou u této práce zásadní:

PackageManager Třída pro zjišťování různých druhů informací týkajících se balíků aplikací, které jsou aktuálně nainstalovány v zařízení[3]. Nejdůležitější metody:

- `queryIntentActivities()` získá všechny aktivity, které najde vůči danému intent předanému parametrem
- `getLaunchIntentForPackage()` vrací intent, který by byl spuštěn při spuštění aplikace. Intent může být poté použit k otevření aplikace
- `getPackageInfo`, `getApplicationInfo` vrací odpovídající objekty specifikované volitelnými příznaky

ApplicationInfo Informace o konkrétní aplikaci. Obsahuje atributy:

- `sourceDir` – cesta k APK (viz sekce 3.1) dané aplikace
- `targetSdkVersion` – cílová verze SDK (sekce 3.1)

[9] MIME – Multipurpose Internet Mail Extensions (víceúčelová rozšíření internetové pošty)
[10] URI – Uniform Resource Identifier (jednotný identifikátor zdroje)

- `flags` – příznaky spojené s aplikací, příznaky mohou být porovnány s konstantami a vyhodnoceny, například příznak `FLAG_SYSTEM` znamená, že aplikace je systémová

PackageInfo Obsahuje celkové informace o obsahu balíku. Atributy:

- `firstInstallTime` a `lastUpdateTime` – datum a čas instalace nebo poslední aktualizace aplikace
- `versionName` a `vesrsionCode` – název nebo kód verze balíku
- `requestedPermissions` – pole všech aplikací požadovaných oprávnění

ResolveInfo Informace, které jsou získány z aplikování intentu na intent filtr aplikace. Pomocí této třídy se dají získat všechny aplikace v zařízení.

PackageItemInfo Třída obsahující informace společné pro všechny balíky. Poskytuje základní atributy:

- `icon` – identifikátor ikony
- `labelRes` – identifikátor názvu balíku
- `packageName` – název balíku

3.3.3 Konfigurační soubory

Jedním z požadavků zadání nastavování správce pomocí editace konfiguračních souborů. Konfigurační soubory jsou soubory používané ke konfiguraci parametrů a počáteční nastavení některých počítačových programů. Bývají použity pro nastavování uživatelských aplikací, serverových procesů a operačních systémů.

Některé aplikace poskytují nástroje k vytváření, modifikování a ověření syntaxe jejich konfiguračních systémů. Některé programy čtou své konfigurační soubory jen při startu, některé je periodicky kontrolují kvůli potenciálním změnám. Neexistují však definitivní standardy nebo zavedené konvence.

Formáty

Existuje řada univerzálních serializačních formátů, které dokážou reprezentovat komplexní datové struktury. Tyto formáty jsou použity často pro konfigurační soubory. Specifikace popisující tyto formáty jsou běžně zpřístupňovány veřejnosti, čímž se zvyšuje dostupnost parserů a vysílačů napříč různými programovacími jazyky.[27]

Dva nejběžnější formáty pro serializaci dat jsou eXtensible Markup Language (XML) a JavaScript Object Notation (JSON). Oba jsou široce používány a dobře dokumentovány, ale byly vytvořeny ještě před existencí chytrých telefonů. Také se místo těchto běžných textových formátů někdy používají binární datové serializační formáty. Zejména jeden známý binární formát je Protocol Buffer (ProtoBuf).

Pro všechny zmíněné formáty existuje API pro jejich parsování v jazyce Java.

XML XML byl navrhnut pro jednoduchou použitelnost napříč internetem. Podporuje širokou škálu aplikací a je čitelný člověkem i strojem a snadno vytvořitelný. Má velkou uživatelskou základnu a je široce používán v různých webových službách. Existuje také mnoho jazyků založených na XML. Přes jeho široké použití, XML je často kritizován za jeho přílišnou podrobnost a nevhodnost pro mobilní prostředí.[36]

JSON JSON je často vychvalován jako odlehčená a efektivnější alternativa XML, jelikož nepoužívá tolik značek. V této době mnoho firem nabízí jejich webové služby ve formátu JSON a je považován jako správná volba pro mobilní zařízení.[36]

ProtoBuf ProtoBuf jakožto binární formát byl navržen tak, aby byl co nejméně náročný a rychlý na serializaci a deserializaci. Na rozdíl od textových alternativ nemusí být parsován znak po znaku, ale využívá tzv. „poziční vazby“. Vazba umožní uložit obě části ze dvojic klíč-hodnota do oddělených souborů. Tyto soubory musí být kompilovány ještě před použitím v programu.[36]

Porovnání

Summaray a Makki[36] provedli porovnání nejen těchto tří zmíněných formátů na mobilním zařízení s operačním sytémem Android . Na toto porovnání bylo použito rychlých a jednoduchých API pro práci se zmíněnými formáty. Byla provedena serializace i deserializace jednoduchých datových struktur a porovnána serializovaná velikost a rychlost jednotlivých formátů.

Z výsledků těchto testů se jeví binární formát jako nejvíce minimalistický. Naopak XML je nejpomalejší a zároveň také zabírají jeho serializovaná data nejvíce místa. JSON se nachází někde mezi – je několikanásobně rychlejší než XML, jeho data mají menší velikost, ale ProtoBuf kraluje ve všech ohledech.

Java Properties Nativní formát v jazyku Java určený pro ukládání dvojic klíč-hodnota. Každý parametr je uložený jako dvojice řetězců, jedním je název parametru (klíč) a druhý jeho hodnota.[31]

Pro manipulaci s tímto formátem existuje třída zvaná `Properties`. Tato třída reprezentuje perzistentní množinu vlastností. Třída může být uložena do nebo načtena ze souboru.[22]

Seznam důležitých metod:

- `load()` slouží k načtení stávajících vlastností.
- `store()` slouží k uložení nových vlastností.

Načítání

Konfigurační soubory musí být nějakým způsobem načteny do aplikace. Načítání může být několika různých způsobů. Zde bude popsáno API, které využívá funkci samotného Linuxového jádra zvané **inotify**.

FileObserver

Abstraktní třída `FileObserver` umožňuje monitorování souborů nebo adresářů. Při detekci nějaké akce nad monitorovanými soubory zavolá událost `onEvent()`, ve které provede kód

reagující na změny. Typ detekované akce může být například: **ACCESS** (čtení), **CREATE** (vytvoření), **DELETE** (smazání), **MODIFY** (zápis) nebo **OPEN** (otevření).[12]

Každá instance této třídy může monitorovat několik souborů nebo adresářů. Pokud je monitorován adresář, pak budou události spuštěny při zvolené akci na všech potomcích tohoto adresáře.

Kapitola 4

Návrh

Tato kapitola má za úkol definovat, co by měla aplikace umět a vytyčit cíle, kterých by mělo být dosaženo. Je zde i návrh samotné aplikace.

4.1 Cíle

Nejprve je potřeba rozebrat body zadání a na základě jejich požadavků stanovit cíle, ke kterým bude implementace směřovat.

Výsledkem této práce by měl být správce aplikací a souborů pro Android, který bude mít minimální požadavky na systém. Konkrétně v těchto ohledech:

- velikost,
- spotřeba paměti a energie,
- co nejnižší úroveň Android API,
- maximální využití plochy obrazovky a
- čitelnost i při velkém počtu informací o zobrazených objektech a akcích.

Dále pak je požadavek na nastavování správce pomocí editace konfiguračních souborů i za běhu aplikace.

Požadavky jsou poměrně jasné a nemusí být provedena důkladná analýza. Základní aplikací bude správce aplikací a souborů. Co by měl správce umět, už bylo řešeno v kapitole 2.1. Cílovou platformou pro správce má být operační systém Android. Nejdůležitější vlastností výsledné aplikace má být její minimálnost. Je potřeba vytyčit si cíle.

Co musí aplikace minimálně umožnit:

- Prohlížet soubory a složky a zobrazit informace o nich.
- Provádět základní operace nad souborovým systémem.
- Prohlížet aplikace a zobrazit informace o nich.
- Poskytnout možnost uživatelského nastavení aplikace.

A co by měla také splňovat:

- Použitelnost na co nejvíce Android zařízeních.
- Verze API musí tedy být co nejmenší, aby podporovala co nejvíce zařízení na trhu, ale dostatečně vysoká, aby bylo možné použít potřebné funkce.
- Pokud možno nepoužívat externí knihovny, také se vyhnout kompatibilním knihovnám.
- Co nejjednodušší algoritmy.
- Absence animací, obrázků a dalších, na systém náročných, elementů.
- Jednoduché uživatelské rozhraní – co nejméně prvků, intuitivnost.
- Prvky uživatelského rozhraní musí být účelné a podstatné, musí využívat co největší plochu obrazovky a být přehledné.
- Nastavení aplikace pomocí konfiguračních souborů, změna i při přepsání konfiguračního souboru mimo aplikaci.
- Objektový návrh musí být kvalitní a samotný kód dobře komentovaný kvůli zveřejnění.
- Po dokončení implementace optimalizovat všechny předchozí body.

Cílová skupina Cílová skupina pro tuto aplikaci bude poměrně široká. Bude se skládat ze všech uživatelů, kteří vlastní mobilní telefon s Androidem. Do cílové skupiny budou patřit jak uživatelé, kteří mají alespoň základní znalosti o souborovém systému a chtějí jednoduchý a intuitivní program pro tento účel, tak zkušenější a náročnější uživatelé, kteří potřebují správce, jenž nebude obsahovat zbytečné funkce a jeho konfigurace bude přenositelná. Také bude určený i pro ty uživatele, kteří vlastní starší zařízení s nízkým výkonem nebo třeba malou pamětí. Předpokládá se, že má uživatel nějakou znalost angličtiny.

Cílová zařízení Aplikace bude určena pro mobilní telefony s operačním systémem Android. Měla by být použitelná na co nejvíce verzích Androidu, počínaje starými verzemi a konče nejnovější verzí. Dále bude podporovat obrazovky různých rozlišení a velikostí v obou orientacích. Bude podporovat zařízení bez externího úložiště, s jedním i více úložišti.

4.2 Návrh řešení problematiky a splnění cílů

Jak bylo řešeno v kapitole 2.4, existuje mnoho aplikací typu správce souborů a aplikací. Tato práce však od nich bude odlišná ve zmíněných aspektech. Při návrhu aplikace bude brán ohled na vytyčené cíle.

Funkcionalita

Aplikace bude splňovat alespoň základní požadavky správce. Základní funkcionalita správce se sestává z několika částí. Základní funkce správce souborů jsou: prohlížet soubory a složky v externích úložištích, zobrazit jejich vlastnosti, otevírat soubory, kopírovat, přesouvat, mazat, vytvářet nové a přejmenovávat. Správce aplikací musí umět zobrazit nainstalované aplikace a jejich vlastnosti a spouštět aplikace. Další funkcí je možnost uživatelského nastavení, které zahrnuje například seřazení.

Poté, co bude implementována veškerá základní funkcionalita, bude možnost rozšíření o další funkce. Například filtrování obsahu, operace s více položkami naráz, možnost přidání aplikace do oblíbených, schránka a podobně.

K implementaci většiny těchto funkcí bude využito API, které bylo popsáno v předchozí kapitole (viz 3.3).

Minimální velikost a API

Jedním z požadavků zadání je, aby měla výsledná aplikace co možná nejmenší velikost. Tohoto se dá docílit několika způsoby. Už tím, že se zvolí minimální verze API co nejmenší, zmenší se celková velikost výsledné aplikace (viz 3.3). Při implementaci bude nejlepší začít vyvíjet pro co nejnižší verzi, aby bylo možné průběžně testovat. Pokud by se používala od začátku vysoká verze API, v konečné fázi vývoje při optimalizaci by muselo být nahrazeno plno různých funkcí, které u dřívějších verzí ještě neexistovaly. Tedy díky tomu, že se už při vývoji použije nízké API, automaticky se tím sníží složitost při implementaci. Úroveň API však také ovlivní, jaké množství zařízení bude aplikací podporováno. Z obrázku 3.3 je zřejmé, že nižší verze než Andorid Gingerbread (úroveň API 9-10) se momentálně na trhu skoro vůbec nevyskytují. Důležité je rozhodnout se, která verze bude ta pravá. Jelikož mezi úrovní API 9 a nižšími nejsou takové obrovské rozdíly, bude vybrána právě API verze 9 jako výchozí minimální.

Už při návrhu by se mělo pamatovat na to, že by mělo být využito co nejméně externích knihoven nebo zdrojů. Každá taková knihovna nebo zdroj můžou rapidně zvýšit velikost aplikace. Zdroji jsou tedy myšleny zejména obrázky. Pokud už bude potřeba použít nějaké obrázky, může být použito například objektů `Drawable`, které zabírají méně místa. Problém s knihovnami však může nastat, když některá novější zařízení podporují takové možnosti, které ještě u starších verzí nebyly vyvinuty (příkladem může být počet úložišť, viz 3.3.1). Pokud takováto situace nastane, bude nevyhnutelné některou kompatibilní knihovnu importovat a použít, aby mohla aplikace cílit jak na staré, tak i na nové zařízení.

Dalším způsobem, jak snížit velikost výsledné aplikace, je lokalizace. V programu bude použito několik řetězců, které budou popisovat některé prvky. Tyto řetězce budou v angličtině. Právě z důvodu zachování co nejmenší velikosti není v plánu vyvíjet multijazyčnou aplikaci. Dále samotný kód může zabírat poměrně značné místo – hlavně tedy generovaný.

K definici rozložení uživatelského rozhraní se používají zdroje (více v sekci 3.1.3). Zdroje mají nezanedbatelnou velikost, a proto, pokud to okolnosti dovolí, se můžou použít několikrát. Při návrhu uživatelského rozhraní se musí myslet na to, aby byla možnost opětovného použití.

Minimální spotřeba paměti a energie

Primárním prostředkem k minimalizování spotřeby paměti a energie je použití správných algoritmů a datových struktur. Datové struktury by neměly obsahovat zbytečné atributy. Pokud je to možné, tak by měly být vytvářeny komplexnější datové struktury těsně před použitím, pokud se používá více stejných datových struktur, měly by být již vytvořené použity znovu. Složitější algoritmy by měly být co nejlépe optimalizované.

Kromě těchto zmíněných způsobů, které mohou mít za následek rapidní pokles spotřeby, se po malých částech dá zmenšit spotřeba paměti pomocí následujících obecných rad, které jsou v knize[32] či v oficiální dokumentaci[20]. V dokumentaci je převzato několik nejdůležitějších částí právě z uvedené knihy. Případné nesrovnalosti z dokumentace byly ověřeny v knize.

Je třeba se vyhnout vytváření nepodstatných objektů. Vytváření objektů je drahá operace. Jelikož Java je jazyk používající garbage collector, objekty nemusí být ihned po použití zahozeny. Statické metody a proměnné jsou výhodou, jelikož pro jejich použití nemusí být vytvořen objekt. Návrhový vzor singleton je přívětivý. Použít vylepšenou syntaxi pro cyklus `for` viz následující algoritmus.

```
int[] array;
//naplneni pole
int sum = 0;
for (int i : array) {
    sum += array[i];
}
```

Dále by se mělo vyhnout používání číselných datových typu s plovoucí řádovou čárkou, jelikož jsou pomalejší než celočíselné typy. Pokud takový typ je potřeba, pak typ `float` je dvakrát menší než `double`. Pro konstanty pak použít modifikátor `static final`.

Při používání základních datových typů jako například integer je možné vytvořit proměnnou buď typu `int` nebo `Integer`. Tyto dvě proměnné se potom liší v několika ohledech. Primitivní datový typ `int` využívá 32 bitů nebo 4 byty paměti a operace nad ním jsou tak rychlé, jak jen to jde. Naopak typ `Integer` je třída, která obaluje typ `int` a poskytuje několik metod na pokročilé operace nad ním. Tyto metody jsou většinou statické, takže není nutné vytvořit tento datový typ pro jednoduché celé číslo. Znamená to, že objekt z této třídy bude vyžadovat větší množství paměti než primitivní typ, přesněji 16 bytů. Podobně na tom jsou typy `bool` a `Boolean`, `double` a `Double` a podobné.

Podobných triků na je mnohem více, tyto jsou však nejdůležitější pro tuto práci a bude na ně brán ohled při implementaci.

Spotřeba energie je nejdůležitější aspekt pro mobilní zařízení. U správce souborů a aplikací není však využití baterie nijak ohromující. Důležité je zařídit, aby aplikace prováděla co nejméně operací v pozadí, nebo dokonce když je zařízení v režimu spánku. Aby se splnil tento požadavek, aplikace by měla provádět operace v jednom okamžiku, klidně si načítat potřebná data do paměti cache. Pokud už aplikace potřebuje některou náročnou funkci vykonat a zároveň ji není potřeba vykonat ihned, může počkat, až se bude zařízení nabíjet. Také po minimalizování aplikace, by se mělo uvolnit co nejvíce používaných zdrojů a po jejím vypnutí všechny.

Maximální využití plochy obrazovky a čitelnost

Další požadavky se týkají uživatelského rozhraní. Aplikace bude určena pro mobilní zařízení různých velikostí. Znamená to, že bude potřeba navrhnout adaptující rozhraní. Android poskytuje možnost definice prvků pomocí jednotek závisejících na rozlišení.

Maximálního využití plochy obrazovky i čitelnosti při velkém počtu zobrazovaných informací může být dosaženo definicí jednoho hlavního rozložení, které bude sloužit jako kostra celého rozhraní. Toto rozložení bude zabírat celou obrazovku. Toto hlavní rozložení bude pak obsahovat prvky uživatelského rozhraní, které se budou vyskytovat uvnitř něj. Aby bylo možné uvnitř zobrazit jakkoli veliký obsah, musí být navrženo rozložení takovým způsobem, který umožní posouvání po obrazovce. Pravděpodobně nebude stačit jen

[0] Inspirováno na https://developer.android.com/training/articles/perf-tips.

jedno základní rozložení, ale bude potřeba definovat nějaké další, které bude jeho potomkem. Rozložení druhé úrovně by mělo pořád splňovat požadavek čitelnosti a maximálního využití plochy.

Výsledkem tedy bude jedno hlavní rozložení zaplňující celou obrazovku, které bude obsahovat několik potomků se stejnou vlastností.

Kromě hlavního rozložení může aplikace obsahovat menu dvou různých typů. Menu aplikace a kontextové. Menu aplikace je dostupné u starších verzí Androidu pomocí určeného tlačítka na zařízení. U nových verzí pak může aplikace obsahovat titulní lištu, která obsahuje tlačítko na otevření menu. Kontextové menu se vytvoří a zobrazí u konkrétní položky uživatelského rozhraní.

4.3 Návrh samotné aplikace

Aplikace se bude sestávat ze dvou hlavních částí. Jednou částí je správce souborů, který bude komplikovanější a druhou částí je jednodušší správce aplikací. Oba správci budou nastavitelní pomocí konfiguračních souborů. Zvlášť bude implementován backend a frontend, k jejich spojení se použije API.

4.3.1 Správce souborů

Návrh správce souborů bude pojat takovým způsobem, aby podporoval základní funkcionalitu. Po implementaci následujícího návrhu bude možnost rozšíření o další funkce.

Jádro správce souborů bude tvořenou jedinou aktivitou. Tato aktivita bude mít takové rozložení, aby zobrazovala řádky a aby těchto řádků dokázala zobrazit jakýkoli počet. Tohoto se docílí pomocí kořenového rozložení, které bude mít možnost scrollování. Jednotlivé řádky budou definovány jako další rozložení, které se bude při běhu aplikace přidávat dynamicky. Tato rozložení druhého řádu se budou skládat z několika položek uživatelského rozhraní. Tyto položky budou sloužit k vypisování informací o jednotlivých souborech a složkách, jako je například jejich název, typ nebo velikost.

Tato aktivita bude obsahovat statickou proměnnou, která bude určovat aktuální zobrazovanou cestu. Díky této cestě bude možné pomocí některého API (více v sekci 3.3.1) vytvořit abstrakci konkrétního zobrazovaného adresáře. V metodě `onCreate()` (viz 3.1.1) u aktivity se potom vypíšou z objektu adresáře všechny adresáře a soubory (souhrnně položky), které jsou jeho přímými potomky. Všichni potomci budou vypsání do uživatelského rozhraní. Pomocí třídy `Adapter` (viz sekce 3.1.3) se dynamicky naplní kořenové rozložení jednotlivými položkami (které zaplní rozložení druhého řádu), kde každá položka bude právě jeden potomek. Tato třída také propojí každou položku uživatelského rozhraní s odpovídající položkou v backendu. Adresář i soubor se od sebe liší v několika ohledech, a proto budou oba reprezentovat jiný datový typ.

Po implementaci bude moci uživatel jen prohlížet kořenovou složku. Třída `Listener` (viz sekce 3.1.3) poté umožní responsibilitu. Každé zobrazené položce frontendu bude přiřazena dvojice listenerů. První listener bude reagovat na jednoduché klepnutí a druhý na delší podržení.

Listener na jednoduché klepnutí se bude chovat odlišně u souboru i adresáře. U adresáře se vyvolá metoda, která vytvoří intent (viz 3.1.1), pomocí něhož se spustí nová aktivita stejného typu s tím, že se přepíše statická proměnná určující momentální zobrazovaný adresář. Poté se zase budou opakovat předchozí kroky – naplní se uživatelské rozhraní, vytvoří se adaptery a listenery. U jiných adresářů, než kořenového daného úložiště bude vytvořena

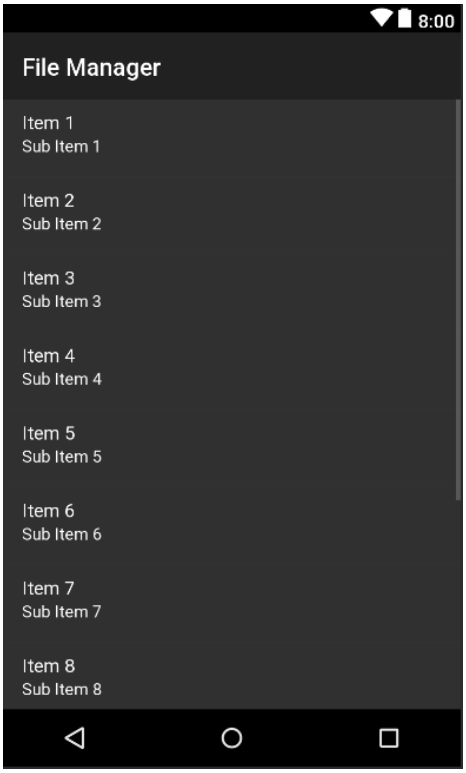
další položka uživatelského rozhraní se stejnými preferencemi. Tato položka bude umístěna na první pozici a bude sloužit k navrácení se o jednu úroveň výše v souborovém systému. Pokud bude klepnuto na položku typu soubor, daný soubor se otevře. Podle přípony se zjistí typ MIME (viz 3.3.1), a uživateli se nabídnou takové aplikace v jeho zařízení, které mohou daný typ otevírat. Jestliže se v zařízení nachází jen jediná aplikace umožňující otevření tohoto souboru, otevře se rovnou v ní. Pokud uživatel nemá potřebné aplikace k otevření souboru, dostane upozornění.

Druhý typ listeneru, tedy ten, jenž reaguje na podržení, bude sloužit k otevírání kontextového menu. Tento typ listeneru bude podobný u obou typů položek. Menu se bude skládat z několika voleb podporujících následující akce: uložení cesty dané položky pro pozdější kopírování či přesunutí, smazání položky, přejmenování a zobrazení podrobností. U souborů bude navíc možnost otevření pomocí jakékoli aplikace nainstalované v zařízení.

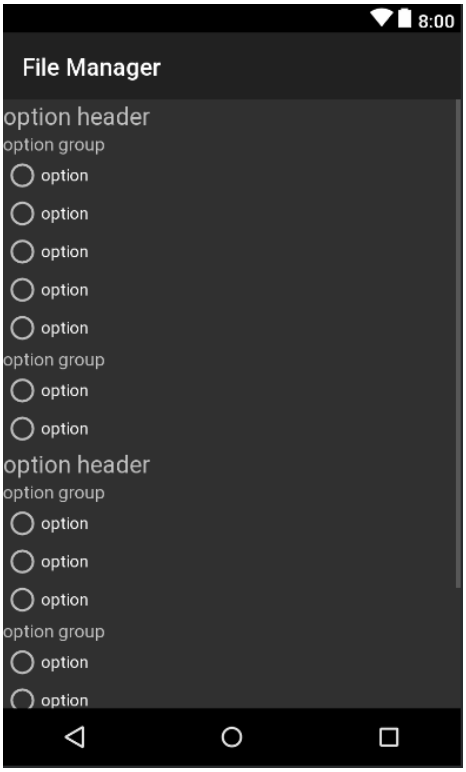
Kromě kontextového menu se v aplikaci bude vyskytovat také menu aplikace. K jeho otevření nebude potřeba dalšího listeneru, protože je již obsažen v API. Menu bude přístupné jedním kliknutím v jakémkoli zobrazovaném adresáři. Menu aplikace bude obsahovat volby, které se budou týkat právě zobrazovaného adresáře. Možnosti v tomto menu budou následující: vytvoření nového adresáře nebo souboru, vložení uloženého souboru (kopírování nebo přesunutí se vybere při ukládání cesty) a přechod do nastavení a do správce aplikací.

Tímto způsobem bude vytvořena základní funkcionalita správce souborů. Po implementaci zmíněných funkcionalit bude možnost rozšíření. Správce klidně může obsahovat některé další funkce, ale pořád se musí myslet na vytyčené cíle.

Ještě před samotným spuštěním si aplikace bude muset vyžádat práva od uživatele, bez kterých mu nebude povoleno ani prohlížení.



Obrázek 4.1: Návrh jednotného uživatelského rozhraní pro hlavní aktivity.



Obrázek 4.2: Návrh jednotného uživatelského rozhraní pro aktivity s nastavením.

4.3.2 Správce aplikací

Správce aplikací bude implementován jako samostatná aplikace, do které se uživatel dostane po klepnutí na odpovídající volbu v menu aplikace. Aby byl brán ohled na vytyčené cíle, správce aplikací bude prosazovat opětovné použití již definovaných zdrojů ze správce souborů. Bude tedy implementován velmi podobným způsobem.

Správce aplikací bude tvořen jednou aktivitou používající stejná rozložení jako správce souborů. Vizuálně tedy budou oba správci velmi podobní. Také komunikace backendu s uživatelským rozhraním bude postavena na stejném principu. Adapter se použije k naplnění obsahem. Listenery budou také stejných typů, na dlouhé i krátké klepnutí. Obsahovat bude také menu obou typů. Díky tomuto se docílí jak intuitivnosti, tak toho, aby byla aplikace co nejvíce minimalistická.

Výčet všech aplikací se zjistí pomocí API (viz 3.3.2). Získané aplikace pak naplní kořenové rozložení. Krátké klepnutí na aplikaci danou aplikaci otevře. Delší podržení potom zase otevře kontextové menu. Toto menu bude obsahovat zobrazení informací, odkaz na systémovou aplikaci, která má za úkol správu aplikací a odinstalování aplikace. Menu aplikace pak bude sloužit k otevření nastavení nebo přesunutí se zpět do správce souborů.

4.3.3 Nastavení a konfigurační soubory

Správce aplikací a souborů budou nastavitelné pomocí konfiguračních souborů. Klíčovým krokem bude vybrání správného formátu konfiguračních souborů. Formát bude vybrán na základě porovnání v sekci 3.3.3. Požadavky na formát jsou následující:

- čitelnost člověkem,

- možnost změny nastavení mimo uživatelské rozhraní,
- rychlé načítání,
- velikost,
- jednoduchá implementace bez zbytečných knihoven.

Po důkladném promyšlení a zvážení všech okolností byl vybrán na účely nastavení formát konfiguračních souborů JSON. JSON podporuje ukládání struktur, čehož může být využito pro složitější nastavení. Dále je dobře čitelný a pokud správně navrhnut, tak není složitá změna. Jeho rychlost a velikost jsou oproti jiným podobným formátům mnohem lepší. Implementace je trochu složitější kvůli tomu, že sice existuje nativní parser v programovacím jazyce Java, ale není dostačující, takže bude potřeba použít externí knihovnu.

V rámci optimalizace byl poté formát konfiguračních souborů změněn z JSON na properties. Tento formát je ještě rychlejší a nativně plně podporován v jazyku Java. Nepodporuje serializaci struktur, ale nakonec tato možnost v této práci nebyla využita. Není tedy potřeba použít externí knihovnu, čímž se sníží velikost APK.

V nastavení si bude moci uživatel přizpůsobit například typ seřazování položek, zobrazení či skrytí některých položek apod. Toto nastavení pak bude jednoduše exportovatelné jednoduchým kopírováním konfiguračního souboru z jednoho zařízení do druhého.

Načítání

Aby bylo možné reagovat na jakoukoli změnu v konfiguračním souboru obsahujícím nastavení, je zapotřebí navrhnout, jakým způsobem se program dozví, že byl soubor změněn. Způsobů načítání souborů existuje několik. Například se může aplikace periodicky dívat do daného souboru a porovnávat, jestli nebyl změněn. Jiný způsob může být, že při každé akci se provede kontrola souboru. V této práci bude použit takový způsob, který bude co nejméně náročný na systém.

Vybraný způsob načítání konfiguračních souborů je načítání souboru pouze při jeho změně prostřednictvím API (viz 3.3.3). Minimalizuje se tak počet přístupu k souborům. Při každé změně konfiguračního souboru se změní nastavení aplikace. Navíc se splní požadavek, že při změně konfiguračního souboru mimo aplikaci se změní nastavení ihned.

4.3.4 Měření, testování a optimalizace

Celý projekt bude implementován v IDE Android Studio[1]. Android Studio poskytuje mnoho užitečných funkcí, které budou využity k měření výsledků a k testování.

Pro zjištění velikosti APK (viz 3.1) obsahuje Android Studio APK analyzátor. Tento analyzátor rozebere celý komprimovaný soubor s příponou .apk a zobrazí velikosti jednotlivých souborů a jejich obsahů. Díky analyzátoru se dají jednoduše najít nechtěné soubory nebo části kódu, které zabírají moc místa. Tohoto analyzátoru bude využito při optimalizaci.

Další poskytovanou funkcí je emulátor reálných zařízení, díky kterému bude možné aplikaci otestovat na všech různých verzích Androidu. Aplikace bude primárně testována na mobilním telefonu s verzí Android 7.0 a dalších několika fyzických zařízeních (Android 2.3.6, Android 8.0 a Android 5.0). Za použití emulátoru se při testování pokryje mnohem více různých zařízení s různými verzemi.

[1] AndroidStudio--https://developer.android.com/studio

Android Studio ve svých funkcích zahrnuje také Android Profiler. Profiler poskytuje statistiky jako využití procesoru, paměti, internetu nebo energie aplikací, a to dokonce i za běhu aplikace. Poskytuje možnost zachycení stop pro pozdější analýzu. Stopy umožní prozkoumání alokací, můžou zabránit únikům paměti, či zjistit, které části aplikace při jakých akcích spotřebovávají nejvíce zdrojů. Na základě informací získaných z komponenty Profiler se bude provádět optimalizace využití paměti, procesoru a baterie.

Kromě Android Studia však budou použity na měření výkonu a spotřeby zdrojů i jiné způsoby. V mobilním zařízení lze ve vývojářských nastaveních spustit funkci, která vizualizuje vykreslování. Tato funkce pomůže zjistit, při jakém chování trvá nejdéle vykreslování a tuto činnost se potom snažit zrychlit.

4.3.5 Zveřejnění

Celý projekt bude publikován jako open source. Bude volně k dispozici pro klonování ostatními programátory. Díky tomuto se aplikace bude moci dále rozšiřovat, upravovat nebo opravovat. Pokud bude nalezena nějaká nedokonalost v programu, nebude ji složité opravit. Kvůli způsobu zveřejnění bude implementace dokumentovaná v Javadoc, aby případní následníci jednodušeji pochopili kód. Pro umístění open source repozitáře poslouží platforma GitHub.

Kapitola 5

Implementace správce souborů a aplikací

V této kapitole se pojednává o implementaci navržené aplikace. Při implementaci se bral ohled na vytyčené cíle 4.1 a vycházelo se z návrhu. V následujících podkapitolách budou rozebrány jednotlivé úseky programu a vysvětleny důležité pasáže.

5.1 Způsob implementace

Vývoj aplikace probíhal metodou prototypování. V průběhu vývoje bylo vytvořeno několik prototypů. Prototypy sloužily k pochopení programování pro operační systém Android a k demonstrování dosažených výsledků vedoucímu a jeho získání jeho zpětné vazby.

Při implementaci bylo dodržováno rozdělení mezi frontend a backend. Obě části spolu komunikují prostřednictvím odpovídajícího API.

V prvním prototypu byl implementován prohlížeč souborů. Tento prohlížeč uměl doopravdy jen prohlížet soubory. Druhý prototyp už uměl i otevírat soubory. Postupně se v dalších prototypech doplňovaly následující funkce: zobrazení vlastností, základní operace nad souborovým systémem, dále označení více položek a nakonec nastavení. Správce aplikací byl implementován naposled. Poté ještě proběhla implementace rozšíření schránka, filtr a oblíbené aplikace.

5.2 Definování základních položek

Jako první věc bylo nutné definovat soubor AndroidManifest.xml (viz 3.1.2). V tomto souboru bylo definováno několik nezbytných položek. Byly zde definovány minimální, kompilační a cílová verze SDK (viz 3.1). Pro splnění cíle, aby byla aplikace použitelná na co největším počtu zařízení, se rozdíl verzí nastavil na co největší, přičemž se počítalo s případnými posuny. Verze kompilační a cílová byly nastaveny na API úrovně 28. To znamená nejvyšší verze, která byla v době implementace dostupná. Minimální verze byla nastavena na úroveň 9, jak bylo navrhnuto.

Pro procházení souborů a manipulaci se soubory se muselo definovat oprávnění s názvem `android.permission.WRITE_EXTERNAL_STORAGE` (viz 3.1.2). Podle toho, jakou verzi Androidu má zařízení, se buď přidělí oprávnění automaticky, nebo musí být potvrzeno uživatelem pomocí dialogu, což je popsáno v kapitole 5.4.3. Jako další krok byla definována aplikace. Aplikaci byl přiřazen styl a název. V aplikaci potom byly definovány jednotlivé

aktivity, jak byly vytvořeny. Každá aktivita dostala svůj název a některé intent-filter (viz 3.1.1).

5.3 Rozvržení aplikace

Už bylo zmíněno, že aplikace je rozdělena na backend a frontend. Soubory, které se týkají frontendu jsou všechny v adresáři `res`, který je pro to určený. Důležité podadresáře v tomto adresáři jsou `layout` a `menu`, ve kterých jsou definovány prvky uživatelského rozhraní. Backend se nachází v adresáři, který nese stejný název jako zvolené jméno balíku. Adresář backendu byl rozdělen do několika podadresářů podle významu. Každý podadresář obsahuje významově podobné soubory, přičemž v každém souboru je právě jedna třída.

activities Tento adresář obsahuje používané aktivity v aplikaci. Aktivity slouží zejména k propojení backendu a frontendu. Pracují s prvky uživatelského rozhraní. Aktivity jsou dohromady 4. Dvě z nich jsou aktivity samotných správců a dvě obsahují nastavení. `MainActivity.java` je hlavní aktivita, která se otevře jako první při spuštění programu. Je to aktivita správce souborů. `AppLauncherActivity.java` je pak aktivita určená pro správce aplikací, je dostupná z hlavní aktivity z aplikačního menu. Druhé dvě aktivity jsou aktivity nastavení. `SettingsLauncherActivity.java` obsahuje nastavení pro správce aplikací a `SettingsManagerActivity.java` pro správce souborů.

listeners_adapters Soubory obsažené v tomto adresáři úzce souvisí s aktivitami. Obsahuje třídy typu listener a adapter, které slouží k propojení uživatelského rozhraní s funkcionalitou. Soubory obsahující ve jméně `Adapter` jsou typu adapter a slouží k naplnění rozložení správnými daty. Soubory s podřetězcem `Listener` jsou typu listener a mají za úkol dávat již naplněným rozložením správnou funkcionalitu. Některé jednoduché třídy těchto dvou typů se nevyskytují zde, ale přímo v souboru s aktivitou jako anonymní třídy kvůli velikosti.

items Zde se nachází třídy definující položky správce souboru. Rozhraní `ItemInterface.java` deklaruje metody, které musí implementovat veškeré položky a u každé položky jsou metody rozdílné. Abstraktní třída `Item.java` obsahuje metody a atributy, které jsou pro společné každý typ položky. A konečně tři různé třídy specifikující typ položky. `ItemDirectory.java` reprezentuje adresář, `ItemFile.java` soubor a nakonec `ItemUp.java`, který je potřeba pro krok o úroveň výš ve souborovém systému.

apps Obsahuje jedinou třídu `App.java` reprezentující aplikaci nainstalovanou v zařízení. Používá ji správce aplikací.

settings Tento adresář má čtyři soubory. `AppComparator.java`, `ItemComparator.java`, `Settings.java` a `SizeTruncator.java`. První dva soubory jsou třídy pro porovnávání a seřazování položek podle vybraného nastavení. Ve třetím souboru je třída s konstantami a se statickými proměnnými používány k uživatelskému nastavení pro oba správce. A poslední soubor řeší zobrazování velikostí.

singletons Adresář obsahující dva návrhové vzory typu jedináček. `FileObserverSingleton.java` slouží ke sledování změn v konfiguračních souborech a `FileOperatorSingleton.java` udržuje podstatné informace o operacích nad souborovým systémem.

helpers Zde jsou umístěny třídy, které využívají ostatní třídy ke své funkcionalitě. V souboru `ManagerActionsHelper.java` jsou třídy, které jsou statické a jsou volány aktivitou správce souborů například při vybrání nějaké akce z menu. Podobný účel má soubor `LauncherActionsHelper.java`, ale pro správce aplikací. `ClipboardHelper.java` má na starosti schránku a `FavoriteAppsHelper.java` oblíbené aplikace.

exception Jednoduchá výjimka `SomethingWrongException.java`, která je vyhozena při nějakém selhání aplikace.

5.4 Implementace jednotlivých částí

Implementace byla dosti zkomplikována právě omezením minimální úroveň API na 9. Mnoho nových funkcí nemohlo být použito a musely být místo nich použity některé zastaralé. Příkladem můžou být fragmenty, vylepšené dialogy, pokročilá titulní lišta a také třeba hlavní rozložení `RecyclerView` z uživatelského rozhraní. Při implementaci funkcionality nemohlo být použito kvalitních aplikačních rozhraní jako například `Files` nebo `SAF` a mnoho dalších. Také, aby bylo možno nainstalovat a bezchybně používat aplikaci na nových i starých verzích Androidu, se muselo udělat více verzí některých funkcionalit. V této podkapitole bude popsána implementace jednotlivých částí aplikace.

5.4.1 Uživatelské rozhraní

Uživatelské rozhraní bylo implementováno podle návrhu a tak, aby byly všechny prvky co nejpodobnější kvůli intuitivnosti a aby byla možnost jejich opětovného použití pro minimalizování velikosti. Nejprve byly vytvořeny zdroje (viz 3.1.3). Na kořenové rozložení hlavní aktivity byl použit prvek **ListView**[1]. ListView je ideální prvek pro potřeby této aplikace. Nejen, že v sobě dokáže uchovat a zobrazovat potomky, ale dokonce je sám od sebe scrollovatený. Funguje tím způsobem, že v daný okamžik neviditelné potomky recykluje, čímž ušetří paměť. Každý prvek má svoje vlastní rozložení a je umístěný ihned pod předchozím. Orientace ListView byla nastavena na vertikální.

Rozložení druhé úrovně (tedy samotných položek) reprezentuje prvek **LinearLayout**[2]. LinearLayout je ViewGroup, která uspořádává své potomky za sebe ve vybrané orientaci. V tomto případě byla vybrána orientace vertikální. Potomci ListView jsou potom dva textové prvky. První potomek je určen pro hlavní popis položky, druhý pro jeho podrobnější specifikaci.

Kromě těchto rozložení pro hlavní aktivity bylo vytvořeno rozložení pro nastavení, které je odlišné u správce souborů i u správce aplikací. Další rozložení je pro dialogy, které se starají zobrazování informací. Splňují ale pořád svůj účel a dbají na vytyčené cíle.

Dalšími prvky uživatelského rozhraní jsou menu. Bylo potřeba definovat několik kontextových i aplikačních menu. Kontextová pro jednotlivé položky a aplikační menu v obou

[1] https://developer.android.com/reference/android/widget/ListView
[2] https://developer.android.com/guide/topics/ui/layout/linear

správcích odlišné. Menu obsahuje kořenový prvek, který ohraničuje samotné menu a jeho potomci jsou jednotlivé položky.

A konečně se mezi frontend dají zařadit dialogy, které slouží k prezentaci ostatních dat. Dialogy jsou typu `AletdDialog`, protože normální dialogy nemohou být v API úrovně 9 použity. Dialogy se zobrazí v popředí.

Pro zobrazování chyb nebo některých informací je použito API `Toast`. Zobrazí se jen na krátkou chvíli a pak zmizí. Uživatel se tak dozví, co potřebuje, ale zase ho zbytečně neobtěžují vyskakovací okna.

5.4.2 Propojení uživatelského rozhraní s funkcionalitou

K propojení backendu a frontendu bylo využito tříd dědících od tříd `Activity`, `Adapter`, `Listener`. V této sekci bude podrobněji popsána implementace těchto tříd u správce souborů. Každá jiná aktivita má svůj vlastní listener a adapter a fungují na podobném principu.

Hlavní třídou správce souborů je třída `MainActivity.java`. Je to třída, která dědí od třídy `Activity` a díky tomu se chová stejně jako aktivita. Má stejný životní cyklus jako aktivita (viz 3.1) a dokáže komunikovat s uživatelským rozhraním.

V metodě `onCreate()`, která je spuštěna po startu aktivity, se ustanoví komunikace s funkcionalitou. Nejprve bylo definováno, které rozložení má aktivita používat pomocí metody. Pro správce souborů je jím rozložení podrobně popsané v předchozí sekci. Toto rozložení je potom registrováno pro využití kontextového menu. Další krok je naplnění rozložení správnými položkami. Správce souborů tedy vyhledá potřebné vypisované položky, provede nad nimi filtrování a seřazování podle uživatelského nastavení a naplní jimi pole. Toto pole je následné předáno jako parametr pro konstruktor třídě `ItemAdapter`, která složí k naplnění uživatelského rozhraní těmito položkami.

Adapter kromě své inicializace obsahuje důležitou metodu `getView()`. Tato metoda slouží k dynamickému generování položek, které jsou potomci kořenového rozložení. Vytvoří se instance z rozložení v XML souboru do odpovídajícího objektu. Tato instance je rozložení druhého řádu a typu `LinearLayout`. Každá instance potom odpovídá jedné položce souborového systému. Položka se pak skládá ze dvou řádků. Na prvním řádku je název položky a na druhém řádku je podle typu položky buď počet potomků u adresářů nebo typ, čas a datum poslední změny a velikost u souboru. Adapter je potom přiřazen ke kořenovému rozložení.

Listener obsahuje metodu `onItemClick()`, která je zavolána při kliknutí na položku.

Aktivity správce souborů a správce aplikací mají definovány další metody týkající se propojení uživatelského rozhraní s funkcionalitou. Metody, které se starají o kontextové menu se nazývají `onCreateContextMenu()` a `onContextItemSelected()`. První z nich se zavolá při dlouhém kliknutí na položku a slouží k vytvoření menu s definovanou nabídkou. Jelikož menu je samotné rozložení, je nutné ho přiřadit ze zdrojů. Druhá je spuštěna při kliknutí některou z voleb v nabídce menu.

Na stejném principu funguje aplikační menu. Metody pro aplikační menu jsou `onCreateOptionsMenu()` a `onOptionsItemSelected()`. Rozdíl je takový, že není potřeba dlouze podržet na tlačítko, které toto menu vytváří, ale jen kliknout. Toto tlačítko se liší v různých verzích Androidu. U starších verzí nebyla podporována rozšířená horní lišta (jen titulek), takže se do něj nemohlo umístit tlačítko. Starší verze tedy buď mají hardwarové tlačítko nebo systémové tlačítko, pomocí kterého můžou menu otevřít. Na nových verzích se potom menu otevře tlačítkem umístěným v horní liště.

5.4.3 Funkcionalita

A konečně byla implementována samotná funkcionalita celé aplikace. Každá část aplikace byla implementovaná zvlášť.

Správce souborů

Správce souborů byl implementován jako první. Jak už bylo popsáno, jeho hlavní třídou je aktivita, která slouží k zobrazování informací na obrazovku. Implementace proběhla podle návrhu a snaží se dodržet vytyčené cíle. Základ celé funkcionality je statická proměnná `currentPath`, která zachovává informaci o tom, ve kterém adresáři se momentálně uživatel nachází.

První spuštění je kořenovém adresáři aktuálního zobrazovaného úložiště. Cesta k tomuto adresáři je uložena do další proměnné, která slouží k rozeznávání kořenového adresáře. Pokud je totiž aktuální adresář kořenový, nezobrazí se položka, která slouží ke kroku o jednu úroveň výše v souborovém systému. V titulní liště je vypsána aktuální zobrazovaná cesta. V kořenovém adresáři celá, v potomcích zkrácená.

Používané API je `java.io.File`, jelikož se implementuje v API 9, novější knihovny tak nejsou dostupné. Pomocí tohoto API jsou uloženi všichni potomci aktuálního adresáře a pomocí zmiňovaného adapteru vykresleny. Pokud uživatel klepne na položku krátce, buď se otevře nová aktivita s novou cestou (v případě položky reprezentující adresář) nebo se otevře možnost výběru, která aplikace má být použita k otevření souboru (v případě položky reprezentující soubor).

Při implementaci otevírání souborů musel být vyřešen problém s kompatibilitou s nejnovějšími verzemi Androidu, kde je potřeba počítat s tím, že pro otevření souboru jinou aplikací musí být implementován FileProvider (viz sekce 3.3.1). FileProvider byl nejprve definován v manifestu jako jedna z komponent typu content provider. Byl vytvořen také soubor **provider_paths.xml**, který obsahuje cestu, které má být umožněno otevírání souborů. Místo této cesty byl zde vložen znak tečky, což reprezentuje kořenový adresář a povolení se rozšíří i pro všechny potomky.

Pokud je na položku klepnuto dlouze, vytvoří se menu, ve kterém si může uživatel vybrat z následujících možností: kopírovat, přesunout, přidat do schránky, smazat, přejmenovat a vlastnosti. U souborů je zde navíc položka „otevřít v“. Všechny metody, které řídí tyto možnosti se nachází v souboru `ManagerActionsHelper.java`. Akce se týkají konkrétní položky.

Kopírovat a přesunout Kopírování a přesun fungují na stejném principu. Pokud se vybere jedna z těchto možností, pak je zkopírována cesta k vybranému souboru. Tato cesta je uložena do třídy `FileOperatorSingleton.java`, která se zde udržuje po celou dobu běhu aplikace, dokud nebyla položka zkopírována. Jestliže již zkopírovaná cesta existuje a znovu je vybrána tato možnost, jednoduše se přepíše. Spolu s uloženou cestou se také uloží informace o tom, zda byla vybrána možnost kopírovat nebo přesunout.

Přidat do schránky Tato možnost byla přidána až v rozšíření, které podporuje funkcionalitu schránky. Byla vytvořena z toho důvodu, aby bylo možné kopírovat nebo přesouvat více položek najednou. Schránka bude vysvětlena v samostatné sekci.

Smazat Jednoduché smazání položky. Pokud je položka typu adresáře, maže se rekurzivně prvně její obsah. Uživatel je upozorněn dialogem, jestli si opravdu přeje smazat soubor a při neprázdné složce také, že má složka potomky.

Přejmenovat Zobrazí se dialog, který obsahuje textové pole, do kterého uživatel zadá nové jméno. Položka je přepsána. V případě, že bylo zvolené neplatné jméno, je uživatel obeznámen a žádná akce se neprovede.

Vlastnosti Při výběru této možnosti se zobrazí výčet informací v dialogu. Informace jsou tyto: název, cesta, velikost, počet všech potomků (u adresářů) a doba poslední modifikace položky.

Když je otevřeno menu aplikace, dostane uživatel na výběr z této nabídky: nový – soubor/adresář, filtr – přidat/odebrat, schránka – kopírovat/přesunout/vymazat/zobrazit, vložit zde, nastavení, vybrat více a správce aplikací. Většina akcí pracuje s konkrétním zobrazovaným adresářem.

Nový Po vybrání této možnosti má uživatel možnost volby, jestli chce vytvořit nový adresář či soubor. Podobně jako u přejmenování je vytvořen a zobrazen dialog, pomocí kterého je specifikován název a poté je položka vytvořena

Filtr Rozšíření, které je popsáno dále v dokumentu. Je zde možnost přidání filtru nebo jeho odebrání.

Schránka Také rozšíření. Obsah stránky je možné zobrazit, nebo vymazat. Položky označené cestami ve schránce se přesunou nebo zkopírují při výběru odpovídající možnost.

Vložit zde Tato možnost zkopíruje či přesune dříve vybranou položku, jejíž cesta a typ přesunutí je specifikován v jedináčkovi. V případě, že nebyla vybrána žádná položka, uživatel se o tom dozví a nestane se nic.

Nastavení Pomocí této volby se uživatel naviguje do aktivity, která obsahuje nastavení správce souborů, konkrétně do třídy `SettingsManagerActivity.java`.

Vybrat více Celá aktivita správce souborů se přepne do módu „vybrat více“. V tomto módu se změní listener reagující na klepnutí a listener na podržení se odebere úplně. Položka na klepnutí zareaguje tím způsobem, že se zabarví. Barva tedy značí označenou položku. Jak bylo vysvětleno výše, prvky rozložení typu `ListView` se recyklují, když nejsou vidět na obrazovce. Není tedy možné čistě jen změnit barvu. Pro rozpoznání označené položky má každá položka proměnnou a při příštím zobrazení se zabarví, jestliže je tato proměnná pravdivá. Při dalším klepnutí na označenou položku se položka odznačí. Menu aplikace se také změní. Obsahuje možnosti: vybrat všechny položky, přidat vybrané položky do schránky, smazat vybrané položky, zobrazit vlastnosti o všech vybraných položkách a navrácení se zpět do normálního módu. Tyto možnosti fungují na stejném principu jako u normálního módu, ale provedou se na všech vybraných položkách. Zobrazení vlastností zobrazuje jen počet položek a jejich celkovou velikost.

Správce aplikací Aplikace se přesune do správce aplikací, konkrétně do třídy `AppLauncherActivity.java`.

Tato část aplikace vyžaduje oprávnění ke čtení a zápisu externích úložišť. Při spuštění této aktivity se tedy pokaždé zkontroluje, jestli jsou daná oprávnění povolena (u starých verzí Androidu není potřeba, viz 3.1.2).

Získání oprávnění

Správce souborů potřebuje pro svoji veškerou funkcionalitu oprávnění od uživatele s názvem `WRITE_EXTERNAL_STORAGE`. Toto oprávnění je považováno za nebezpečné, a proto se k jeho získávání musí přistupovat speciálně. Na zařízeních s verzí Androidu, ve které se ještě nerozlišovala oprávnění podle nebezpečnosti, není s oprávněním žádný problém a aplikace ho sama získá při instalaci od uživatele. Takto získané oprávnění je perzistentní a uživatel jej nemůže měnit.

Problém však nastává u novějších zařízení, kde aplikace nemůže získat oprávnění při instalaci, ale musí se dotazovat za běhu aplikace. Při každém spuštění hlavní aktivity je zkontrolováno, zda má aplikace zmíněná práva. Pokud má, funguje normálním způsobem. Pokud však práva nemá (první spuštění, nebo ho uživatel změnil v nastavení), zobrazí uživateli dialog, pomocí kterého o oprávnění žádá. Jestliže uživatel odmítne, aplikace zobrazí chybu a ukončí se. V opačném případě se volba zapamatuje a do příštího zrušení oprávnění mimo aplikaci není uživatel z tohoto důvodu vyrušován.

Jelikož je nastavena hlavní aktivita správce souborů jako hlavní aktivita celé aplikace, musí řešit několik dalších věcí při inicializaci. Metoda zahrnující tyto operace se nazývá `initFileManager()`. Provádí operace při inicializaci po každém spuštění a při úplně prvním spuštění aplikace po nainstalování.

Inicializace po každém spuštění

Tyto operace se provedou pokaždé po otevření aplikace a jen jednou při jednom sezení, tedy dokud není aplikace ukončena a znovu spuštěna. Pro zjištění prvního spuštění v jednom sezení byla vytvořena statická proměnná `firstTime`, která je přepsána právě po této inicializaci. Při prvním otevření aktivity má hodnotu `true` a při dalších `false`.

Po zjištění, že se jedná o první spuštění aktivity, se vytvoří instance jedináčka `FileObserverSingleton`. Tento jedináček zahájí sledování adresáře s konfiguračními soubory a jeho potomků – samotných konfiguračních souborů. Více v sekci o konfiguračních souborech a nastavení 5.4.3. Načtou se také první hodnoty do uživatelského nastavení ve třídě `Settings.java` a inicializuje se funkce schránky a oblíbených aplikací.

První spuštění po nainstalování

Tato operace se provede jen jednou za celou dobu, co je aplikace nainstalovaná. K tomu, aby se zjistilo, že se spouští aplikace poprvé po instalaci, slouží SharedPreferences (viz 3.2.3). Pomocí tohoto API se uloží soubor do výchozího adresáře pro to určeného v interní paměti. K tomuto souboru nemá za normálních okolností uživatel přístup, takže jej nemůže modifikovat. Do souboru je uložena jedna dvojice klíč-hodnota, která reprezentuje právě to, že už proběhlo první spuštění aplikace po nainstalování. Při každém dalším spuštění se přečte tato hodnota a operace neproběhnou. Jediná možnost, jak přimět aplikaci, aby

povedla prvotní inicializaci je přepsání této proměnné z hodnoty **false** na hodnotu **true**. Tohoto jsou schopni jen ti uživatelé, jenž mají větší zkušenosti a mají **rootnuté** zařízení a tedy přístup k interní paměti.

Prvotní inicializace zahrnuje vytvoření a inicializace souborů. Vytvoří se všechny konfigurační soubory ve výchozím adresáři aplikace v externí paměti (pokud je dostupná, pokud ne, tak v interní, viz 3.2). Těmi jsou ve finální verzi aplikace:

- **clipboard.txt** – soubor pro schránku
- **favoriteApps.txt** – soubor pro oblíbené aplikace
- **manager.properties** – soubor pro nastavení správce souborů
- **launcher.properties** – soubor pro nastavení správce aplikací

Po vytvoření se soubory s nastavením naplní výchozími hodnotami. Následuje inicializace po každém spuštění.

Správce aplikací

Druhou implementovanou částí aplikace byl správce aplikací. Ten byl implementován tak, aby mohl využít zdroje a některé části kódu správce souborů.

Používaná API při implementaci byla všechna popsaná v teoretické části (viz sekce 3.3.2). Nejprve byl vytvořen intent s takovými vlastnostmi, aby po aplikování na intent filtr pomocí třídy `PackageManager` a její metody `queryIntentActivities()` vypsal všechny nainstalované aplikace v zařízení (kromě této) a uložil do proměnné typu seznam objektů ze třídy `ResolveInfo`. Poté se projde celý tento seznam a vytvoří se odpovídající instance třídy `App`. Tato třída obsahuje všechny potřebné informace o dané aplikaci. Pro získání informací jsou použity třídy (kromě již zmíněných) `PackageInfo` na většinu z nich a `ApplicationInfo` na zbytek. Uchovávané informace o každé aplikaci jsou: označení (label), jméno balíku, je-li aplikace systémová nebo ne, jméno a kód verze, velikost APK, cílová SDK, datum a čas instalace a poslední aktualizace a nakonec oprávnění. V rozšíření byla ještě přidána proměnná na rozeznání oblíbených aplikací.

Na prvním řádku vypsané aplikace v rozhraní je zobrazeno její označení a na druhém řádku název balíku. Při kliknutí na aplikaci se aplikace otevře. Každá aplikace má kontextové menu, které obsahuje tyto volby: zobrazit informace, přidat do/odebrat z oblíbených aplikací a otevřít v systémové aplikaci.

Zobrazit informace Nejdůležitější funkce, která zobrazí všechny dříve zmíněné uchovávané informace o aplikaci. Zobrazí je pomocí dialogu.

Přidat do/odebrat z oblíbených aplikací Oblíbené aplikace jsou rozšíření, které je dokumentováno později v dokumentu v sekci 5.4.3.

Otevřít v systémové aplikaci Při výběru této možnosti se aplikace otevře v systémovém správci aplikací. Může sloužit například pro odinstalování nebo změny systémového nastavení.

Dále je zde implementováno i aplikační menu. Toto menu obsahuje volby: obnovit, nastavení, filtr a správce souborů.

Obnovit Ukončí a znovu načte aktivitu správce aplikaci. Slouží například k tomu, kdyby byla instalována aplikace na pozadí a po dokončení by si chtěl uživatel zobrazit novou aplikaci, použil by tuto možnost.

Nastavení Pomocí této volby se uživatel naviguje do aktivity, která obsahuje nastavení správce aplikací, konkrétně do třídy `SettingsLauncherActivity.java`.

Filtr Rozšíření, které je popsáno dále v dokumentu. Je zde možnost přidání filtru nebo jeho odebrání.

Správce souborů Aplikace se přesune do správce souborů, konkrétně do třídy `MainActivity.java`.

Nastavení

Už několikrát bylo v tomto textu zmíněno, že implementace nastavení souvisí s konfiguračními soubory. Zde bude implementace popsána. Každý ze správců má svoje odpovídající nastavení. Některá nastavení jsou však stejná.

Nastavení a konfigurační soubory

Jako serializační formát konfiguračních souborů pro nastavení byl podle návrhu použit JSON. Pro ostatní konfigurační soubory mimo nastavení byl použit formát .txt.

Operace se soubory s příponou .json jsou prováděny za využití odpovídajících JSON knihoven. Operace s klasickými textovými soubory jsou zpracovávány pomocí API jazyka Java. Při prvním spuštění po nainstalování aplikace jsou vytvořeny všechny potřebné soubory v adresáři systémem určeném pro tuto aplikaci. Pokud má aplikace práva a je přítomno externí úložiště, vytvoří se zde. V opačném případě je adresář vytvořen v interní paměti, kde nemůže být přepsán uživatelem (bez rootu). Soubory určené pro nastavení jsou naplněny výchozími hodnotami.

Po celou dobu běhu programu je udržována instance jedináčka, který se stará o pozorování změn konfiguračních souborů. Jedináček používá třídu `FileObserver` z API uvedené v kapitole 3.3.3. Observer pracuje tak, že pozoruje změny v určeném adresáři. Jakmile zaregistruje nějakou změnu v některém ze souborů v adresáři, aplikace spustí předefinovanou metodu `onEvent()`, kde se provede aktualizaci nastavení v odpovídající třídě. Veškerá nastavení se uchovávají ve statických atributech ve třídě `Settings.java`. Pro schránku a oblíbené aplikace jsou to odpovídající třídy. V nich se neaktualizuje nastavení, ale jejich obsah se synchronizuje s obsahem proměnných v programu.

Aktualizace nastavení probíhá tím způsobem, že se nejdříve načte celý konfigurační soubor a zkontroluje se každé nastavení. Pokud není nastavení validní (uživatel ho mohl přepsat mimo aplikaci), zobrazí se výchozí nastavení. Pokud je validní, přepíšou se staré hodnoty novými.

Nastavení mají své aktivity. V těchto aktivitách je jednoduché rozložení, které je na první pohled intuitivní a uživatel by neměl mít problém pochopit význam jednotlivých položek. Tyto aktivity slouží k tomu, aby si mohl uživatel vybrat nastavení aplikace podle svých potřeb.

Zapsání vybraného nastavení je vykonáno hned po zvolení možnosti. Soubor s nastavením je načten do programu, a všechny volby se aplikují znovu zapíší do odpovídajícího souboru.

Nastavení jsou různá pro každý ze správců. Nastavení je navrženo takovým způsobem, že přidat novou volbu je velmi jednoduché. Proto nebylo vytvořeno zbytečně mnoho možností, ale spíše jen ty důležité z pohledu autora a testerů. Další možnosti mohou být implementovány v rozšíření.

Nastavení správce souborů

U nastavení ve správci souborů je hlavní volbou prohlížené úložiště. Implementace poskytuje možnost vybrat si ze všech externích dostupných úložišť. Bylo potřeba použít metody z vyšších API než 9. Kvůli tomu se využilo kompatibilních knihoven. Sice to odporuje jednomu cíli, ale zase k dalšímu se to přiklání. Aplikace bude pro co nejvíce zařízení, ale bude zabírat nějaké místo navíc. Kromě externích úložišť jsou v nabídce vypsány i cesty v interním úložišti. Jelikož tato práce nepodporuje plné prohlížení systémových položek, nezobrazují se některé informace správě.

Aplikace tedy podporuje prohlížení i SD karty. Pro staré verze Androidu plně podporuje všechny operace s SD kartou. Pro vyšší verze Androidu než 5.0. se změnila práva k přístupu. Bohužel nebylo možné za pomocí API z úrovně 9 a nižších použít takové funkce, které by umožňovaly plný přístup k SD kartám a dalším sekundárním externím úložištím. SD karta má u těchto zařízení jen práva ke čtení a ne k zápisu. Při implementaci práv k zápisu by tak nebyly dodrženy cíle a aplikace by tak nebyla minimalistická.

Úplná podpora interního úložiště a SD karet by mohla být jako rozšíření této aplikace.

Další nastavení je řazení. Řadit se mohou položky podle jména, data poslední úpravy, velikosti a přípony. Řazení může být buď sestupně nebo vzestupně. Řazení má za úkol třída `ItemComparator` implementující rozhraní `Comparator` z API. Metoda pro porovnáváni se jmenuje `compare()`. Po načtení všech položek se pošle celé pole právě do této třídy, která seřadí položky podle uživatelského nastavení a vrátí seřazené pole. Uživatel má možnost v nastavení také vybrat, jestli mají od sebe být separovány soubory a složky ve výčtu všech položek. Pro některé uživatele pak může být aplikace přehlednější, a proto byla tato volba přidána.

Dále je možnost zobrazení či nezobrazení přípon a skrytých souborů a adresářů (začínajících znakem tečky). A poslední implementovanou volbou je zaokrouhlení a zkrácení vypisovaných velikostí. Místo 1520 B se vypíše 1,5 KB. Pro tuto aplikaci se použilo jednotek jako např. KB – kilobajt, který reprezentuje tisíc bajtů a ne KiB – kibibajt, který reprezentuje 1024 bajtů. Velikosti řeší třída `SizeTruncator`.

Nastavení správce aplikací

Správce aplikací podporuje řazení také, podle stejného princip s rozdílem, že pro řazení je určena třída `AppComparator`. Řadit se dá podle jména aplikace, jména balíku, velikosti APK a data instalace a poslední aktualizace. Stejně jak u správce souborů může být řazení sestupně i vzestupně. Velikosti APK mohou být také zaokrouhleny stejným způsobem.

Uživatel si může zvolit možnost zobrazení všech, jen systémových nebo jen nesystémových aplikací. U zobrazení skupin se dá vybrat mezi volbami: nezobrazovat skupiny, zobrazovat systémové aplikace nejdříve a zobrazovat oblíbené položky nejdříve.

Rozšíření

Po implementaci základní funkcionality bylo implementováno několik rozšíření. Rozšíření byla implementována tak, aby pořád dbala na to, aby byla aplikace minimalistická, ale zároveň přidala důležité funkce aplikaci.

Filtr Prvním implementovaným rozšířením nad rámec základní funkcionality správce souborů byla funkce filtru. Filtr funguje takovým způsobem, že si uživatel v aplikačním menu vybere položku přidat filtr. Otevře se dialog s textovým polem vyzývající uživatele k zadání řetězce, který má být obsažen v názvu položky. Řetězec se následně aplikuje na vypsané položky. Pokud položka obsahuje řetězec, zobrazí se. V opačném případě je položka vyfiltrována. Filtr je možné využít jak u správce souborů, kde se filtrují soubory a adresáře, tak ve správci aplikací pro filtrování názvů aplikací.

Schránka Jako další rozšíření byla implementována schránka pro správce souborů. Vznikla proto, aby bylo možné provádět základní operace správce souborů nad více položkami zároveň. Schránka je implementována podobným způsobem jako konfigurační soubory a je uložena ve stejném adresáři. Může být tedy změněna i přepsáním jejího obsahu než jen přes uživatelské rozhraní. Soubor reprezentující schránku není však napsán v žádném serializačním formátu, ale v základním textovém formátu s příponou .txt. Soubor se nazývá **clipboard.txt**. Každá položka, která má být přidána do schránky je přidána na nový řádek tohoto souboru. Při práci se schránkou se vždy načte tento soubor řádek po řádku, kde každý řádek reprezentuje jednu cestu. Proměnná typu pole řetězců je naplněna všemi cestami, které jsou tímto způsobem načteny a zároveň reprezentují validní soubor nebo adresář v souborovém systému. Pole cest je poté ještě důkladně zkontrolována, jestli neobsahuje duplicity či podřetězce tímto způsobem: pokud je nějaká cesta podřetězcem jiné cesty, pak je tato specifičtější cesta eliminována z pole proměnných. V samotném souboru zůstává původní obsah, jen se programově změní pole a dále se pracuje jen s touto upravenou proměnnou neobsahující nevalidní a nadbytečné cesty. Schránka může být zobrazena uživateli pomocí položky v menu. Zobrazí počet cest a vypíše takto minimalizované cesty pomocí dialogu. Schránka může také být celá vymazána skrze uživatelské rozhraní. Podporuje dvě operace – kopírování a přesun. Může buď kopírovat nebo přesunout položky reprezentované uloženými cestami v jedné várce.

Oblíbené aplikace Na podobném principu jako schránka pak bylo implementováno rozšíření pro správce aplikací „oblíbené aplikace“. Pro oblíbené aplikace se vytvoří soubor **favoriteApps.txt**, který má stejný formát jako schránka a stejným způsobem se i načítá. Každý řádek v tomto souboru představuje jednu aplikaci nainstalovanou v zařízení pomocí názvu balíku (např. com.example.hertl.app). Pokud jsou v souboru takové řádky, které neoznačují validní balík nainstalované aplikace, potom je tento řádek ignorován. Soubor s oblíbenými aplikacemi může být jednoduše přenositelný na jiná zařízení. Každá oblíbená aplikace má před svým názvem ve výpisu všech aplikací hvězdičku (*). Uživatel má možnost seřadit si aplikace tak, aby byly oblíbené aplikace vypsané na pozicích před ostatními aplikacemi.

Ostatní

Další jednoduchá funkcionalita je změna tlačítka „zpět“. Toto systémové tlačítko má různé významy. Ve správci souborů slouží jako další způsob ke kroku o úroveň výše, pokud se

uživatel nenachází v kořenovém adresáři právě procházeného úložného prostoru. V takovém případě se při prvním klepnutí zobrazí uživateli zpráva, že pokud chce opustit aplikace, ať klepne na tlačítko znovu. Jestliže tedy klepne dvakrát rychle po sobě, aplikace se ukončí. Stejnou funkci má ve správci aplikací. V aktivitách s nastavením a v módu vybrat více položek u správce souborů se naviguje uživatel zpět do aktivity, ze které se sem uživatel dostal.

Velmi důležitou funkcionalitu podporuje implementovaná metoda `refresh()`, která způsobí, že je daná aktivita ukončena a spuštěna znovu. Je využívaná velmi často. Například při jakékoli operaci, kde je možná změna položek (mazání, kopírování, ...). Také se volá v metodě aktivity `onRestart()` (viz 3.1). Uživatel totiž může aplikaci minimalizovat, změnit soubory v jiné aplikaci a otevřít znova tuto aplikaci. Případná změna se ihned zaregistruje a zobrazí. Ve správci aplikací je tato metoda jednou z možností v nabídce aplikačního menu.

Aplikace obsahuje metodu na automatické otevření systémové klávesnice při vytvoření a zobrazení dialogu s textovým vstupem, což přidává na přívětivosti.

Kapitola 6

Finalizace

V předposlední kapitole je popsána finalizace práce. Kapitola nejprve popisuje optimalizaci a naměřené výsledky a vysvětluje postup při testování. Dále zde jsou rozebrána a navrhnuta případná rozšíření a konečně uvedeno dostupné zveřejnění.

6.1 Optimalizace

Optimalizace proběhla po dokončení implementace. Jelikož už v návrhu a při implementaci byl brán ohled na vytyčené cíle, aplikace byla ještě před optimalizací velmi minimalistická. Po implementaci se přece jen ještě zmenšily nároky. Zde budou vyjmenovány a popsány části, které byly optimalizovány.

- U správce souborů při otevření nové aktivity se stará zavře. Nebude se tak udržovat v paměti a ušetří z dlouhodobého hlediska důležité zdroje. V důsledku to znamená, že se aktivita pokaždé zobrazí nahoře a není možné zachovávat stav, ve kterém byla aktivita zanechána. Také se při návratu k aktivitě nezobrazí dříve vykreslené dialogy a podobně.
- Byl minimalizován počet aktivit. Z původních pěti se staly čtyři, kde aktivita reprezentující mód pro vybrání více položek ve správci souborů není samostatná aktivita, ale část hlavní aktivity. Zmenšil se tak kód a odstranily zbytečné definice zdrojů.
- Některé prvky uživatelského rozhraní, které byly využívány často, byly vloženy do zvláštního souboru a poté pomocí `include` vloženy do rozložení. Nejsou tak definovány zbytečně víckrát.
- Některé operace, které trvají delší dobu, byly optimalizovány z pohledu uživatelé přívětivosti. Mezitím, co se provádí operace v pozadí, v hlavním vláknu aplikace (vlákno uživatelského rozhraní) je zobrazen odpovídající element reprezentující proces načítání, případně podrobnější informace. Na práci v pozadí je použito API `AsyncTask`[7]. Umožňuje průběžnou aktualizaci vlákna v popředí a po dokončení pokračuje v primárním vláknu aplikace.
- Formát konfiguračních souborů byl změněn z formátu JSON na formát properties. Formát properties je podporován přímo v jazyku Java a jeho API je velmi jednoduché na použití a je nenáročné na systém. JSON formát nebyl potřeba, protože nakonec ukládání nastavení do konfiguračních souborů nebylo prováděno serializací

celých struktur, ale jen jednoduchých hodnot. JSON API je mnohem větší a navíc bylo potřeba použít některých knihoven třetích stran, které zbytečně navyšovaly velikost. Tato změna má poměrně velký dopad na aplikaci. Soubory mají jinak stejné vlastnosti a jsou stále jednoduše editovatelné i uživatelem.

- Přesun položky ve stejném úložišti byl implementován pomocí jednoduchého přepsání cesty souboru místo složitého kopírování a mazání. Optimalizace proběhla i u přesunu jedné položky i u schránky. Přesun položky je po optimalizaci téměř stejně rychlý jako přejmenování souboru.

- Scrollování hlavního rozložení bylo optimalizováno tak, aby bylo rychlejší a lépe responzivní a aby se uchovávaly již zobrazené položky v paměti namísto jejich opakovaného načítání. Implementace proběhla pomocí vzoru „holder“, který právě k tomuto účelu slouží.

- Největší úspěch při optimalizaci bylo redukování velikost APK. Už jen díky zmíněným optimalizačním krokům se podařilo o poměrně značnou velikost aplikaci zmenšit. Největší dopad na velikost APK umožnila optimalizace po zkontrolování APK pomocí nástroje v Android Studiu. Po analýze bylo zřejmé, které části aplikace jsou velké a tyto části bylo potřeba redukovat. Zejména všechny kompatibilní knihovny bylo potřeba smazat nebo redukovat. Nepoužité metody a třídy byly vyloučeny. K tomu sloužila možnost `minifyEnabled` a `shringResources` při stavění APK nástrojem gradle. Původní velikost přibližně 3 MB byla zmenšena pod 100 KB. Celková velikost aplikace po nainstalování je samozřejmě vyšší viz obrázky 6.1 a 6.2.

Obrázek 6.1: Celková velikost nainstalované aplikace před optimalizací na zařízení s úrovní API 10.

6.2 Naměřené výsledky

Zde budou ukázky naměřených minimálních výsledků, kterých dosáhla aplikace. Většina měření probíhala v prostředí Android Studio. Bylo použito komponent analyzátor APK a Profiler. Profiling byl prováděn na reálném zařízení s verzí Androidu 7.0, velikostí RAM 3,0 GB a s procesorem o frekvenci 2,0 GHz. Rozlišení displeje u tohoto zařízení je 1920 x 1080.

Obrázek 6.2: Celková velikost nainstalované aplikace po optimalizaci na zařízení s úrovní API 10.

Tento telefon je poměrně výkonný na tuto aplikaci, ale bohužel Profiler nelze použít na starších zařízeních.

Konkrétní naměřené výsledky jsou v příloze A.

6.3 Průběh testování

Při této práci nebyly implementovány žádné automatické testy, jelikož by nebyly dobře využitelné kvůli velkému rozsahu cílových zařízení. Všechny testy probíhaly ručně. Po implementaci každé větší funkcionality byly provedeny testy zaměřené na danou funkcionalitu. Tedy když byla například implementována operace kopírování, tak byla ihned otestována.

Zařízení, které sloužilo pro testování po celou dobu implementace, bylo Xiami Redmi 4 Note s verzí Androidu 7.0. Všechna funkcionalita je tedy důkladně testována na tomto zařízení. Jedním z cílů aplikace bylo však vyvíjet pro co největší škálu verzí Androidu. Není jednoduché sehnat zařízení se všemi úrovněmi API od 9 do nynější a už vůbec na všech zařízeních aplikaci důkladně otestovat. Proto kromě reálných zařízení, která sloužila pro testovací účely (Android verze 2.3.6, 5.1, 8.0 a další okrajově), bylo využito emulátorů. Tyto emulátory umožní vytvoření obrazu reálného zařízení s jakoukoli verzí Androidu. Aplikace byla otestována na různých dalších verzích Androidu (včetně minimální úrovně API 9).

Testování v nereálných zařízeních probíhalo ve vývojovém prostředí Android Studio. Android Studio podporuje instalaci aplikace přímo do vytvořeného emulátoru.

Kromě autora pomáhalo testování provádět několik dalších lidí na svých zařízeních. Kompletní testování proběhlo až při dokončování celé práce a pomohlo objevit několik nedokonalostí či problémů aplikace. Po několika iteracích testování a opravování chyb už nebyl velký počet chyb nalezen. I přesto se však může objevit problém. Nejen z tohoto důvodu bude aplikace umístěna do volně přístupného repozitáře, aby bylo možné případné chyby opravit.

6.4 Možná rozšíření

Rozšíření by mohlo být podpora prohlížení interního úložiště při rootnutém zařízení. Tato práce takovou možnost nepodporuje. Implementace začala a možná by došla zdárnému

konci, ale je to velmi náročná práce a z časových důvodů by se nestihla implementace. Procházení některých interních cest, které vyžadují vyšší práva, nemůže být implementováno čistě za použití jazyka Java. Pro dosažení tohoto cíle se musí pracovat s konzolí a se systémovými nástroji. Například pro zobrazení všech složek se použije příkaz `ls` nebo pro zjištění informací o souborech a adresářích příkaz `file`.

Dalším problémem při této práci byla práce s SD kartou (a s dalšími sekundárními externími úložišti). U zařízení se starší verzí Androidu než 5.0 není se čtením a zápisem na SD kartu žádný problém. Bohužel novější zařízení mají mnohem přísnější přístup, není možné tedy na sekundární úložiště zapisovat. Prohlížet, otevírat a všechny další operace, které vyžadují jen práva ke čtení, je možné. Pro zápis musí být použito úplně jiného API (konkrétně SAF, viz 3.3.1). Toto API by už nedodržovalo cíle této práce, jelikož není minimalistické (velikost, výkon, ...). Možnost této implementace ale existuje a kdyby někomu nevadily důsledky, může to implementovat.

Aplikace by se samozřejmě mohla rozšiřovat v mnoha směrech. Například by mohla mít mnohem více nastavení v konfiguračních souborech. Koncept konfiguračních souborů by mohl být rozšířen a využit na další funkcionalitu (oblíbené soubory). Implementace je ale naprogramována tak, aby bylo jednoduché takovéto rozšíření přidat. V této práci jsou však implementovány hlavně základní operace, protože se cílí na co nejmenší systémové požadavky. S jakoukoli další funkcionalitou přibývá přinejmenším velikost aplikace, někdy se zvýší i nároky. Proto, při případném navázání na tuto práci, musí být pamatováno na tato upozornění.

6.5 Zveřejnění a využití

Aplikace je zveřejněna v repozitáři na platformě GitHub na odkazu `https://github.com/hertl/minimalistic_manager`. Repozitář je volně dostupný ke klonování a nebo k úpravě. Tímto se docílí možného navázání na práci nebo využití některými uživateli v praxi. Aplikace je unikátní tím, jak je minimalistická, takže by mohla být pro někoho zajímavá. Zejména uživatelé, kteří jsou pokročilí nebo kteří mají starší zařízení, by tuto aplikaci mohli uvítat.

Kapitola 7

Závěr

Cílem této práce bylo vytvořit aplikaci správce souborů a aplikací pro operační systém Android, která by měla mít minimální požadavky na systém.

Výsledek práce tento cíl splňuje. Po návrhu byla zdárně implementována aplikace, která umožňuje správu souborových systémů i správu nainstalovaných aplikací. Podporuje velké množství zařízení s různými verzemi operačního systému a s různou výkonností a je minimalistická. Požadavky na systém jsou také minimální.

Při práci byla nejdříve prozkoumána existující řešení a shrnuty jejich výhody a nevýhody. Poté byla nastudována potřebná literatura týkající se programování pro operační systém Android. Dále byla prostudována aplikační rozhraní související s tématem obsahující přístup k souborům a aplikacím a k optimalizaci. Následně byla navržena minimalistická aplikace typu správce souborů a aplikací pro operační systém Android. Návrh zahrnoval minimální požadavky na systém a nastavování pomocí editace konfiguračních souborů. Po konzultaci s vedoucím byla aplikace implementována a nakonec optimalizována. Výsledky byly změřeny a jsou součástí této práce v příloze A a je v vidět, že byla práce úspěšná. Výsledek byl popsán, vyhodnocen a zveřejněn jako open-source ve volně dostupném repositáři na platformě GitHub (`https://github.com/hertl/minimalistic_manager`).

Autor této práce byl obohacen důležitou schopností analyzovat komplexní problém, nastudovat potřebnou literaturu, navrhnout řešení a posléze provést implementaci.

Aplikace byla navrhnuta a okomentována takovým způsobem, aby bylo možné na tuto práci navázat bez větších obtížností. Případná rozšíření či úpravy mohou být implementovány díky volnému zveřejnění. Aplikace by mohla být rozšířena o další uživatelská nastavení podle vlastního zájmu. Dále je možné implementovat podporu prohlížení kořenových složek nebo zápis na sekundární externí úložiště. Optimalizace by mohla být prováděna nadále v několika iteracích, než by aplikace dosáhla dokonalosti. Alternativně by aplikace mohla mít ještě menší velikost snížením počtu podporovaných verzí Androidu, což by znamenalo méně kódu a žádné kompatibilní knihovny.

Literatura

[1] *Android*. [Online; navštíveno 08.05.2019].
URL https://www.techopedia.com/definition/5415/android

[2] *Android 7.0 Behavior Changes*. [Online; navštíveno 08.05.2019].
URL
https://developer.android.com/about/versions/nougat/android-7.0-changes

[3] *android.content.pm*. [Online; navštíveno 08.05.2019].
URL https:
//developer.android.com/reference/android/content/pm/package-summary

[4] *Android SDK*. [Online; navštíveno 08.05.2019].
URL https://www.techopedia.com/definition/4220/android-sdk

[5] *Application Fundamentals*. [Online; navštíveno 08.05.2019].
URL https://developer.android.com/guide/components/fundamentals

[6] *App Manifest Overview*. [Online; navštíveno 08.05.2019].
URL https://developer.android.com/guide/topics/manifest/manifest-intro

[7] *AsyncTask*. [Online; navštíveno 08.05.2019].
URL https://developer.android.com/reference/android/os/AsyncTask

[8] *Data and file storage overview*. [Online; navštíveno 08.05.2019].
URL https://developer.android.com/guide/topics/data/data-storage

[9] *ES File Explorer*. [Online; navštíveno 08.05.2019].
URL https://es-file-explorer.en.uptodown.com/android

[10] *File*. [Online; navštíveno 08.05.2019].
URL https://developer.android.com/reference/java/io/File.html

[11] *File Extension*. [Online; navštíveno 08.05.2019].
URL https://www.techopedia.com/definition/1831/file-extension

[12] *FileObserver*. [Online; navštíveno 08.05.2019].
URL https://developer.android.com/reference/android/os/FileObserver

[13] *FileProvider*. [Online; navštíveno 08.05.2019].
URL https:
//developer.android.com/reference/androidx/core/content/FileProvider

[14] *Files*. [Online; navštíveno 08.05.2019].
URL https://developer.android.com/reference/java/nio/file/Files

[15] *Input events overview*. [Online; navštíveno 08.05.2019].
URL https://developer.android.com/guide/topics/ui/ui-events

[16] *Introduction to Activities*. [Online; navštíveno 08.05.2019].
URL https:
//developer.android.com/guide/components/activities/intro-activities

[17] *Layouts*. [Online; navštíveno 08.05.2019].
URL https://developer.android.com/guide/topics/ui/declaring-layout

[18] *Multipurpose Internet Mail Extensions (MIME)*. [Online; navštíveno 08.05.2019].
URL https://www.techopedia.com/definition/1693/multipurpose-internet-mail-extensions-mime

[19] *Open files using storage access framework*. [Online; navštíveno 08.05.2019].
URL
https://developer.android.com/guide/topics/providers/document-provider

[20] *Performance tips*. [Online; navštíveno 08.05.2019].
URL https://developer.android.com/training/articles/perf-tips

[21] *Permissions overview*. [Online; navštíveno 08.05.2019].
URL https://developer.android.com/guide/topics/permissions/overview

[22] *Properties*. [Online; navštíveno 08.05.2019].
URL https://developer.android.com/reference/java/util/Properties

[23] *Request App Permissions*. [Online; navštíveno 08.05.2019].
URL https://developer.android.com/training/permissions/requesting

[24] *Save files on device storage*. [Online; navštíveno 08.05.2019].
URL https://developer.android.com/training/data-storage/files.html

[25] *Total Commander - file manager*. [Online; navštíveno 08.05.2019].
URL https://play.google.com/store/apps/details?id=
com.ghisler.android.TotalCommander

[26] *<uses-sdk>*. [Online; navštíveno 08.05.2019].
URL
https://developer.android.com/guide/topics/manifest/uses-sdk-element

[27] *Configuration file*. 2019, [Online; navštíveno 08.05.2019].
URL https://en.wikipedia.org/wiki/Configuration_file

[28] *File manager*. 2019, [Online; navštíveno 08.05.2019].
URL https://en.wikipedia.org/wiki/File_manager

[29] *file URI scheme*. 2019, [Online; navštíveno 08.05.2019].
URL https://en.wikipedia.org/wiki/File_URI_scheme

[30] *Mobile Operating System Market Share Worldwide*. 2019, [Online; navštíveno 08.05.2019].
URL http://gs.statcounter.com/os-market-share/mobile/worldwide/

[31] *.properties*. 2019, [Online; navštíveno 08.05.2019].
URL https://en.wikipedia.org/wiki/.properties

[32] Bloch, J.: *E ective Java*. Boston: Addison-Wesley, third edition vydání, [2018], ISBN 978-0-13-468599-1.

[33] Luboslav Lacko: *Vývoj aplikací pro Android*. Brno: Computer Press, 2015, ISBN 978-80-251-4347-6.

[34] Maheshwari, M.: *Top Programming Languages for Android App Development*. 2018, [Online; navštíveno 08.05.2019].
URL https:
//dzone.com/articles/most-used-programming-languages-for-android-app-de

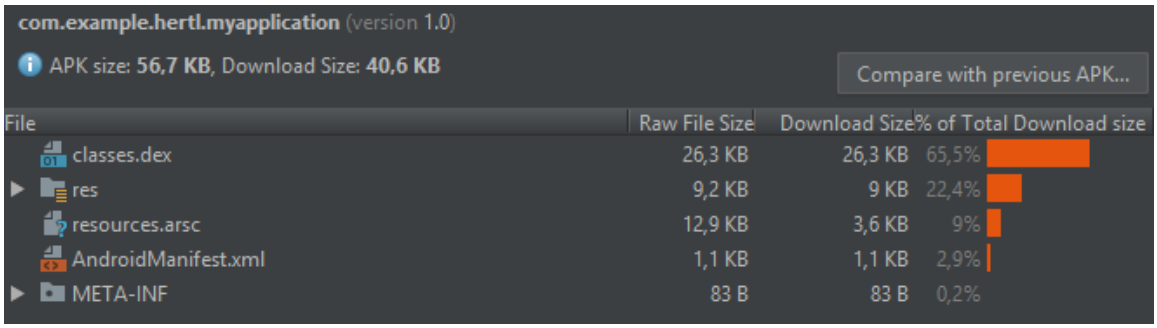
[35] Raphael, J.: *Android file management: An easy-to-follow guide*. 2018, [Online; navštíveno 08.05.2019].
URL https://www.computerworld.com/article/3221287/android-file-management-an-easy-to-follow-guide.html

[36] Sumaray, A.; Makki, S. K.: A comparison of data serialization formats for optimal efficiency on a mobile platform. *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication - ICUIMC '12*, 2012: s. 1–, doi:10.1145/2184751.2184810.
URL http://dl.acm.org/citation.cfm?doid=2184751.2184810

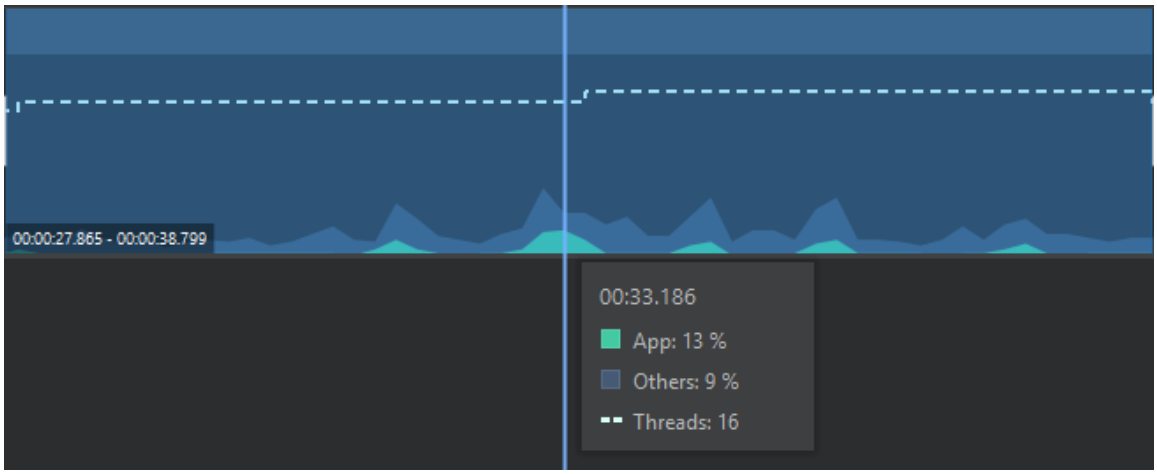
Příloha A

Naměřené výsledky

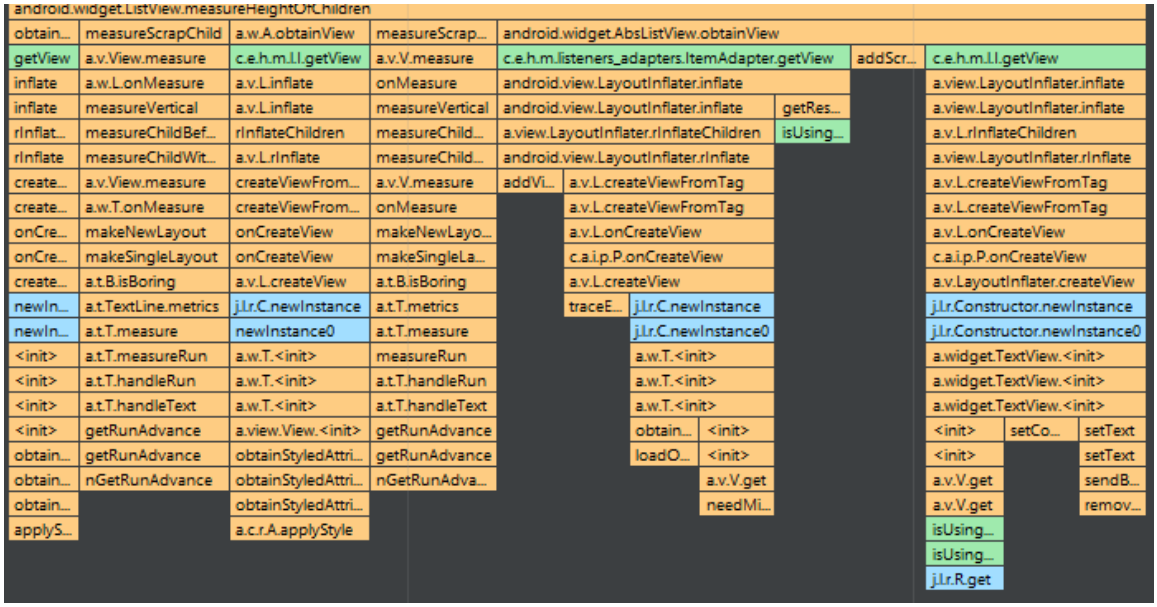
Tato příloha obsahuje naměřené výsledky ve finální verzi aplikace.



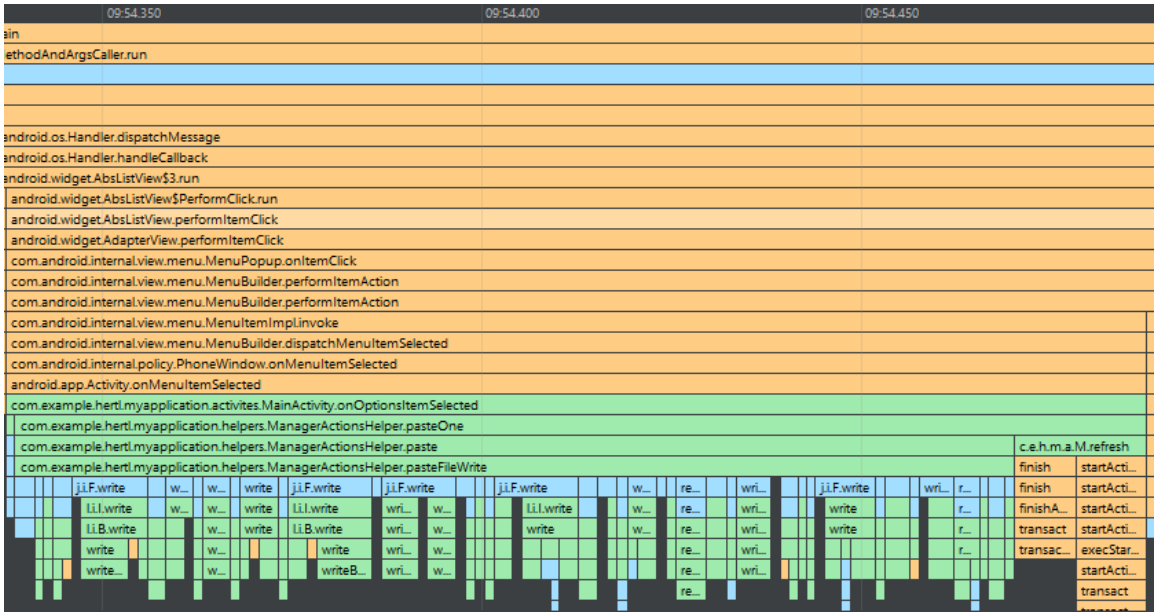
Obrázek A.1: Výsledek zobrazující velikost APK aplikace. Velikost je 56,7 KB, pokud by byla aplikace stahována z Google Play, pak by její velikost byla jen 40,6 KB. Největší podíl na velikosti má soubor **classes.dex**, který obsahuje kompilované třídy. Adresář **res** pak obsahuje definované zdroje.



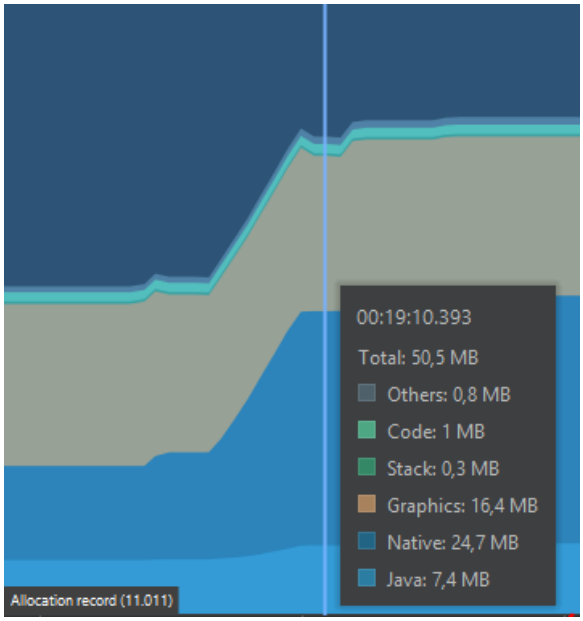
Obrázek A.2: Nejvyšší dosažené zatížení procesoru při práci s aplikací. Procesor byl aplikací zatížen nejvýše z třinácti procent.



Obrázek A.3: Podrobný výpis probíhajících operací při nejvyšším zatížení. Zelenou barvou označené části jsou metody aplikace. Na obrázku je vidět, že většinu nároků má metoda `getView()` u komponenty typu Adapter. Nejnáročnější na procesor je tedy vykreslování. Toto se může velmi lišit v každé verzi Androidu. Například starší zařízení nemají tak velká rozlišení, a proto není tak náročné vykreslování.



Obrázek A.4: Kopírování souboru o velikosti 3 MB trvalo aplikaci přibližně 100 milisekund.



Obrázek A.5: Naměřená velikost spotřebované paměti při zobrazení adresáře se 300 položkami. Nejvíce paměti zabírají datové typy `char`, které nejsou veliké, ale spíše jsou velmi často využívány. Spotřebovaná paměť se může také velmi lišit u různých zařízení. Položka **Graphics** reprezentuje paměť sloužící ke grafickým operacím. Položka **Native** v tomto případě také reprezentuje paměť zabíranou grafickými operacemi. A konečně položka **Java** je paměť alokovaná na samotnou funkcionalitou aplikace.

Obrázek A.6: Profiling grafické karty a rychlosti vykreslování. Pokud je graf pod zelenou čarou, uživatel by neměl zaregistrovat.

Příloha B

Obsah přiloženého paměťového média

- `FileManager` – zdrojové soubory aplikace
- `Profiling` – průběžné naměřené výsledky
- `Text` – zdrojové soubory textové části práce
- `FileManager.apk` – instalovatelný aplikační archív (přeložené řešení)
- `LICENSE` – licence
- `README.txt` – návod k instalaci a použití
- `xhertl04_text.pdf` – tento dokument