Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Anže Sršen

Mobilna aplikacija za pregledovanje slik visokih ločljivosti

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Peter Peer
Asistent: as. Bojan Klemenc

Ljubljana 2012

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00202/2012
Datum: 07.03.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **ANŽE SRŠEN**

Naslov: **MOBILNA APLIKACIJA ZA PREGLEDOVANJE SLIK VISOKIH LOČLJIVOSTI**

HIGH RESOLUTION IMAGE VIEWER FOR MOBILE DEVICES

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Preučite možnost prikaza slik visoke ločljivosti z več kot 10 milijoni slikovnih točk na mobilnih napravah z operacijskim sistemom Android. Razvijte aplikacijo, ki bo omogočala predoglede in prikaze različnih pogosto uporabljenih zapisov slik visoke ločljivosti. Aplikacija naj podpira delno nalaganje slik, kar omogoča prikaz slik na napravah, ki nimajo dovolj pomnilnika za prikaz slik v celoti.

Mentor:
doc. dr. Peter Peer

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Nikolaj Zimic

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Anže Sršen,
z vpisno številko **63070154**,

sem avtor diplomskega dela z naslovom:

Mobilna aplikacija za pregledovanje slik visokih ločljivosti

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Petra Peera in as. Bojana Klemenca
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 15. junija 2012 Podpis avtorja:

Zahvala

Zahvaljujem se svoji zaročenki Maji in svoji družini za izkazano podporo pri študiju. Še posebej se zahvaljujem mentorju doc. dr. Petru Peeru in as. Bojanu Klemencu za mentorstvo in nasvete pri izdelavi diplomske naloge.

Kazalo

Povzetek

V okviru diplomskega dela smo razvili mobilno aplikacijo za pregledovanje slik visokih ločljivosti na mobilni platformi Android. Vsaka aplikacija na Androidu ima svoje omejitve pri uporabi pomnilnika. Velikost razpoložljivega pomnilnika na aplikacijo niha od naprave do naprave. Pri nekaterih napravah je samo 16 MB, medtem ko je pri ostalih lahko tudi 24 MB ali več. Če bi na primer želeli prikazati 32-bitno sliko v velikosti 4000 $\times$ 3000 v razmerju 1:1, bomo porabili približno 48 MB pomnilnika. V primeru, ko imamo na voljo samo 16 MB na aplikacijo, bi bil naš poskus prikaza neuspešen. Za projekt smo se odločili, ker smo želeli ponuditi uporabnikom novo mobilno okolje za pregledovanje slikovnih datotek. S tem smo želeli postaviti osnovo za nadaljnji razvoj, kjer bi končna rešitev bila aplikacija sposobna procesiranja in pregledovanja slikovnih datotek ne glede na velikost. Za zgled smo imeli programa IrfanView in Adobe Photoshop. V prvem delu smo predstavili tehnologije, ki smo jih uporabili pri razvoju aplikacije – operacijski sistem Android in različni tipi slikovnih formatov. V osrednjem delu je pojasnjena sama arhitektura aplikacije in uporabljene razvojne rešitve. V zaključku pa so predstavljene možnosti nadaljnjega razvoja mobilne aplikacije.

Ključne besede: Android, Java, mobilne naprave, operacijski sistem, slikovni formati, slike visokih ločljivosti

Abstract

In the thesis we have developed a mobile application for viewing high resolution images for the Android mobile platform. Each Android application has its limits when it comes to memory usage. The size of the available memory for an application varies from device to device. Some devices have only 16 MB, while others can have 24 MB or more. If we would like to display a 32-bit image of size 4000×3000 in a 1:1 ratio, we will spend about 48 MB of memory. In cases where we have only 16 MB per application, our attempt of displaying the image would be unsuccessful. We decided to do this project to offer users a new mobile enviroment for image viewing. We wanted to create a basis for further development, where the end product would be an application for processing and viewing images regardless of their size. For reference we used IrfanView and Adobe Photoshop. In the first part of the thesis we describe the technology used in developing the application, such as the Android operating system and various types of image formats. The central part of the thesis describes an application architecture and development solutions. Other problems and possible further development of the mobile application will be discussed in the conclusions.

Key words: Android, Java, mobile devices, operating system, image formats, high resolution images

Poglavje 1

Uvod

Pred leti so bili mobilni telefoni sposobni samo klicati in pošiljati kratka tekstovna sporočila. Z razcvetom "pametnih" mobilnih telefonov so se možnosti uporabe razširile. Sedaj nam mobilna naprava ponuja pregledovanje elektronske pošte, brskanje po internetu, igranje 3D iger, itd. Mobilna tehnologija je presegla vsa pričakovanja in dosegla nove ravni razvoja in uporabe. Navkljub temu, da strojna oprema mobilnih naprav še vedno ne dosega standardov njihovih večjih sorodnikov, so postale mobilne naprave za mnoge prijetna tehnološka rešitev.

V okviru diplomskega dela smo razvili mobilno aplikacijo imenovano AweView za pregledovanje slik visokih ločljivosti, ki ponuja sledeče:

- Enostaven in hiter pregled vseh slikovnih datotek, ne glede na število datotek.
- Celozaslonski prikaz slik (visokih ločljivosti) v razmerju 1:1.
- Sinhronizacija podatkov s pomočjo storitev v oblaku.

Ciljni uporabniki so lahko nadzorniki proizvodnje v tkalnicah, kjer bi jim naša aplikacija omogočala enostaven in učinkovit pregled vseh vzorcev tkanin. Nadzorniki potrebujejo celozaslonski prikaz slike v razmerju 1:1, ker lahko samo s tem razberejo vezavo in druge podrobnosti iz posamezne slike.

Poleg njih bi lahko bili ciljni uporabniki tudi fotografi, katerim bi aplikacija omogočala enostaven pregled vseh fotografij v razmerju 1:1 in bi lahko s tem izločili vse slabe in neprimerne posnetke. Nenazadnje bi aplikacija lahko bila v pomoč podjetjem v tekstilni industriji za lažjo demonstracijo vzorcev tkanin njihovim strankam. Skratka možnosti uporabe je veliko, sama aplikacija bi koristila vsem, ki si želijo enostaven pregled nad slikami visokih ločljivosti.

Diplomsko delo bo lahko služilo kot vzvod za nadaljni razvoj aplikacije, ki bi v končni obliki lahko bila novo delovno okolje za vse, ki želijo imeti možnost procesiranja slik visokih ločljivosti na svoji mobilni napravi.

Vsaka aplikacija na programski platformi Android ima svoje omejitve pri uporabi pomnilnika. Velikost razpoložljivega pomnilnika na aplikacijo niha od naprave do naprave. Pri nekaterih starejših napravah je meja postavljena zelo nizko. Prvi mobilni telefon (HTC G1) z Androidom je imel 16 MB razpoložljivega pomnilnika na aplikacijo in kamero, ki lahko zajame sliko v velikosti 3.2 miljona slikovnih pik. To pomeni, da zajema slike velike 2048×1536 slikovnih pik. Če želimo prikazati 32-bitno sliko takšne velikosti bomo potrebovali približno 13MB pomnilnika. V tem primeru nam za preostale objekte ostane samo še 3 MB. Čeprav to ne zagotavlja, da bo naši aplikaciji zmanjkalo pomnilnika, bo to zagotovo prispevalo k večji verjetnosti [1].

Res je, da imajo novejše naprave sedaj na voljo tudi 24 MB ali več na posamezno aplikacijo, ampak moramo upoštevati, da razvijalci za Android ne razvijamo samo za eno mobilno napravo. Vredno je omeniti, da je zgoraj opisana omejitev postavljena zato, da zagotavlja, kar se da nemoteno večopravilnost na mobilnih napravah. Seveda lahko za rezerviranje pomnilnika uporabljamo tudi C-kodo, kjer se naši objekti ne štejejo v zgornjo omejitev, vendar s tem izgubimo večji del funkcionalnosti iz Jave.

Zgoraj opisani problem postane očiten tudi pri bolj preprostih stvareh, kot je recimo sistem za upravljanje grafičnih predogledov slikovnih datotek.

V operacijskem sistemu Android je učinkovito upravljanje pomnilnika za slike občutljiva tema in mnogokrat velik izziv za vse razvijalce.

Splošna ideja je, da vsako sliko, ki jo želimo prikazati, v bistvu pomanjšamo, da prihranimo na pomnilniku, seveda s tem izgubimo na podrobnostih slike. Rešitev deluje, če delamo na aplikaciji, ki služi samo kot klasičen pregledovalnik slik. Mi želimo točno videti posamezno slikovno točko in tudi vedeti, kje se nahaja v naši sliki, zato zgoraj opisani koncept ne bi deloval za naš problem.

Aplikacija, ki smo jo ravili za prikaz slik visoke ločljivosti omogoča učinkovito sinhronizacijo podatkov s pomočjo storitve v oblaku, ki ne uporablja relacijske podatkovne baze na uporabniški strani in omogoča tudi selektivno osveževanje podatkov. Sinhronizacijo bi lahko v bodoče še dodatno pospešili z novimi iskalnimi algoritmi.

Implementirali smo tudi sistem za upravljanje grafičnih predogledov slikovnih datotek. Ponuja hitro in učinkovito prikazovanje predogledov ter ima minimalno porabo notranjega in zunanjega pomnilnika.

Celozaslonsko prikazovanje slik visokih ločljivosti v razmerju 1:1 je trenutno še v prototipnem stanju. Razvili smo konceptno rešitev, ki uporablja primeren pristop k zgoraj opisanemu problemu, vendar trenutno podpira samo slikovna formata JPEG in PNG. Sliko obravnavamo po delih in sledimo koordinatam s pomočjo transformacijske matrike.

V poglavju 2 bomo naredili splošni opis uporabljenih tehnologij, kjer bomo predstavili 5 slikovnih formatov, ki jih mobilna platforma Android trenutno podpira in osnovno strukturo operacijskega sistema Android. Na koncu 2. poglavja bo sledil še kratek opis storitve Dropbox. V poglavju 3 bomo opisali podobne rešitve za Android in osebne računalnike. Poglavje 4 bo dalje razdeljeno na 2 podpoglavji. Prvo bo vsebovalo opis izgleda naše aplikacije, medtem ko bomo v drugem opisali naše razvojne rešitve zgoraj opisanih problemov in klasični pristop pri prikazovanju slikovnih datotek, ki

lahko služi kot vodilo za ostale, ki se podajo na razvoj aplikacij za procesiranje slik. V zadnjem poglavju bomo povzeli naše delo in spoznanja iz izkušenj, ki smo jih pridobili tekom razvoja aplikacije.

Poglavje 2

Uporabljene tehnologije

Preden se lotimo dejanske problematike naše aplikacije, moramo pojasniti tehnologijo v ozadju. Za delo smo uporabljali razvojno okolje Eclipse, vtičnike podjetja Google za Eclipse in razvojni paket Android SDK.

2.1 Tipi slikovnih formatov

Potrebno je omeniti, da bomo pri slikovnih formatih pogosto uporabljali izraz stiskanje slikovnih datotek ali kompresijski algoritem. Kaj je cilj kompresije ali stiskanja? Cilj je, da predstavimo sliko z manj bajti, kot pa bi jo morali in s tem prihranimo na velikosti samih datotek ter času prenosa med posameznimi napravami. Najbolj učinkovita kompresija je dosežena s približki originalne slike in ne z natačnim poustvarjanjem [4].

Uporabljali bomo dva izraza:

- **Brezizgubna kompresija** (angl. Lossless compression):
  To pomeni, da pri stiskanju ne izgubljamo podatkov.
- **Izgubna kompresija** (angl. Lossy compression):
  To pomeni, da pri stiskanju izgubljamo podatke.

Obstaja veliko različnih slikovnih formatov, vendar se je Google odločil, da jih bo Android podpiral samo 5 [11]. Trenutno podpira sledeče formate:

- *Graphic Interchange Format* (GIF).
- *Bitmap* (BMP).
- *Joint Photographic Experts Group* (JPEG).
- *Portable Network Graphics* (PNG).
- *WebP*.

2.1.1 Graphic Interchange Format

Format GIF je eden izmed starejših podprtih slikovnih formatov. Vse se je začelo leta 1987, ko je CompuServe izdal prvo specifikacijo poimenovano GIF87a. Specifikacija je bila prosto dostopna in s tem so praktično vse aplikacije za procesiranje slik posvojile GIF [3]. Bil je eden izmed prvih rešitev, ki so se spopadle s takrat novodobnim problemom učinkovitega elektronskega shranjevanja slik.

Primarna moč je njegov brezizguben kompresijski algoritem, poznan tudi kot linearna kompresijska metoda Lempel-Ziv-Welch (LZW). Algoritem ponuja do 4:1 kompresijo slik brez izgube s podobnim kompresijskim in dekompresijskim časom. GIF ponuja tudi možnost prikaza grobe verzije slike, preden je slika prenešena oziroma prikazana. Nenazadnje format podpira več slik na posamezno datoteko, kar je odprlo vrata preprostim animacijam, ki jih pogosto srečamo na svetovnem spletu. Vendar pa mnogi pregledovalniki slik prikažejo samo prvo sliko v datoteki GIF. Format GIF pa ima tudi slabosti. Prva najbolj očitna je ta, da je omejen na največ 256 barv oziroma odtenkov sive v posamezni sliki. Zaradi te omejitve izgubijo slike prenesene v GIF precej podrobnosti. Druga slabost je bila pri prvotni specifikaciji GIF87a, ki prvotno ni dovoljeval slik s prosojnimi plastmi, vendar so problem kasneje delno popravili [7]. V zatonu lahko dandanes še vedno srečamo datoteke formata GIF, vendar pogosteje v obliki kratkih preprostih animacij, ker je trenutno njegova edina prednost samo ta, da podpira več slik na posamezno datoteko.

2.1.2 Bitmap

Čeprav je bil format BMP prvotno mišljen samo za operacijski sistem Windows, je sedaj dostopen tudi na drugih platformah. Oblikovan je bil tako, da je njegova notranja struktura ustrezala načinu hranjenja slik v pomnilniku, ki ga uporablja Windows. V osnovi je zelo enostaven format, ki podpira slike z 1, 4, 8, 16, 24 in 32 bitov na slikovno točko, čeprav bomo težko našli datoteke BMP, ki podpirajo 16 ali 32 bitov. Format podpira tudi enostavno kompresijo slik za 4 in 8 bitov na slikovno točko. Njegova kompresija je praktična samo v primeru, da so v sliki veliki bloki istih barv. Skratka težko najdemo stisnjeno datoteko BMP[3].

2.1.3 Joint Photographic Experts Group

JPEG so ustvarili v zgodnjih 90 letih prejšnega stoletja. Ni klasični datotečni format, temveč je ime kompresijskega algoritma, ki ga je razvila skupina Independent JPEG Group. Ustvarjen je bil samo za shrambo in prenos fotografij. Njegova moč leži v dobri kompresiji slik. Slika, ki bi zasedla 1 MB kot datoteka BMP, bi bila s pomočjo formata JPEG velika samo 50 KB. JPEG lahko tako uporabi za prikaz do 24-bitno barvno globino (16,7 miljona barv, medtem ko format GIF podpira 256 barv). Navkljub svojim dobrim lastnostim ni primeren za vse aplikacije. Vse njegove kompresijske metode so na splošno izgubne, zato je neprimeren za vmesno shranjevanje podatkov, ko urejamo slikovno datoteko. Z vsakim stiskanjem podatkov bi slika izgubila na podrobnostih. Med drugim JPEG ni tako dober pri stiskanju besedila in risb kot je dober pri fotografijah. Navkljub vsemu je še vedno eden izmed najbolj razširjenih slikovnih formatov [3, 4, 7].

2.1.4 Portable Network Graphics

Format PNG je bil ustvarjen leta 1996 v odgovor problemu s patenti pri formatu GIF. Težave za takrat splošno razširjen GIF so se začele, ko je podjetje Unisys (lastnik patenta za kompresijski algoritem LZW) začelo zahte-

vati pristojbine za uporabo njihovega patenta. Odločitev je razjezila mnoge razvijalce programske opreme, zato je uporaba formata GIF upadla po tem dogodku.

Nekateri razvijalci so videli rešitev v formatu JPEG. Vendar JPEG ne stiska dobro določenih tipov slik, zato ni mogel v celoti zamenjati formata GIF. Thomas Boutell je kmalu po zgoraj opisanih težavah organiziral skupino z imenom PNG Development Group. 01.10.1996 je skupina uspešno dokončala različico 1.0 slikovnega formata PNG, ki uporablja brezizgubno kompresijo in podpira sledeče:

- Do 48 bitov na slikovno točko v barvnih slikah.
- 1-, 2-, 4-, 8- in 16-bitno natančnost vzorčenja.
- Alfa-kanal za polno kontrolo nad prosojnostjo.
- Gama-korekcija za skladnost v svetlosti na različnih platformah.
- Kompresijski algoritem, ki ni patentiran.

PNG je še vedno zelo razširjen slikovni format in tudi eden izmed bolj prepoznavnih [3, 6, 7].

2.1.5 WebP

Zadnji format je mogoče najbolj nepoznan od vseh zgoraj naštetih, je pa tudi najmlajši. Avtor formata je podjetje Google, ki ga je ustvaril v želji, da bi pospešil prenose slikovnih datotek s še dodatnim zmanjševanjem velikosti datotek. Glavni lastnosti formata WebP so sledeče:

- Brezizgubna in izgubna kompresija.
- Animacija slikovnih datotek.

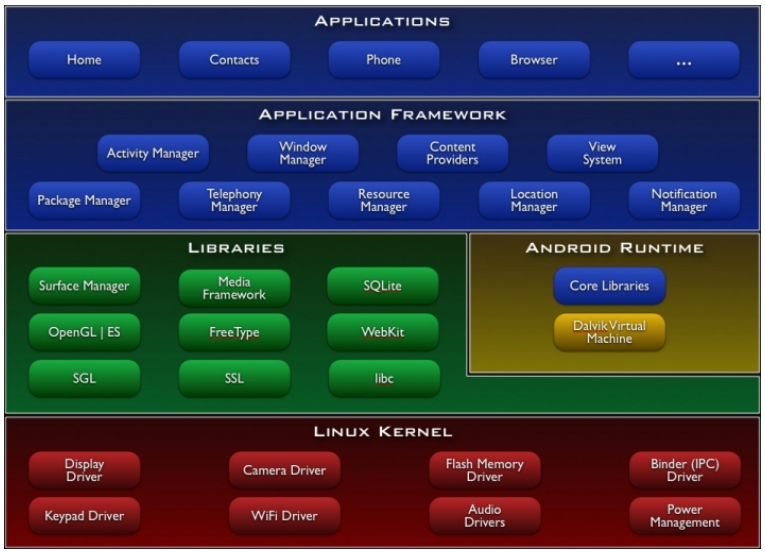
WebP uporablja kompresijski algoritem, ki je bil osnovan na VP8 video kodeku. VP8 so spremenili v odprtokodni kodek leta 2010, tako je na voljo

vsem razvijalcem za uporabo. Sam format trenutno še ni tako razširjen, vendar se bomo lahko mogoče v prihodnosti zaradi njega poslovili od JPEG, PNG in ostalih [5].

2.2 Android

Android je operacijski sistem za mobilne naprave, kot so tablični računalniki in pametni telefoni. Razvija ga konzorcij z imenom Open Handset Alliance. Trenutno je najhitreje razvijajoča se programska platforma, ker je bilo v nekaj letih razvoja preko 7 različic. Začetki segajo v leto 2005, ko je Google kupil podjetje Android Inc., ki je razvijalo takrat še nepoznan mobilni operacijski sistem. Leta 2007 so izdali distribucijo Android in ustanovili konzorcij Open Handset Alliance. V tem konzorciju lahko najdemo podjetja (Google, Broadcom, HTC, LG, Nvidia, itd.) za strojno in programsko opremo ter telekomunikacije. Zatem je Google izdal vso izvorno kodo Androida kot odprtokodno in jo licenciral pod licenco Apache [2].

Najnovejša raziskava podjetja Gartner Inc. (v nadaljevanju Gartner) [19] o tržnem deležu mobilnih naprav (podatki so za 3. četrtletje leta 2011) je pokazala, da glede na operacijski sistem mobilne naprave vodi programska platforma Android podjetja Google z 52.5 %, za tem sledi sistem Symbian podjetja Symbian Ltd. s 16.9 % in na tretjem mestu je iOS podjetja Apple Inc. (v nadaljevanju Apple) s 15.0 %. Če zgornje podatke primerjamo s podatki o tržnem deležu za 3. četrtletje 2010, lahko hitro ugotovimo, da je delež programske platforme Android narastel za 27.2 %. Ostala podjetja so v primerjavi s podatki iz leta 2010 izgubila odstotke tržnega deleža. Symbian je izgubil 19,4 % in iOS pa samo 1,6 %. Glede na te podatke lahko sklepamo, da programska platforma Android doživlja razcvet.



Slika 2.1: Arhitektura programske platforme Android.

2.2.1 Struktura programske platforme

V tem razdelku bomo povzeli osnovno strukturo operacijskega sistema Android. Bolj podrobno o sami strukturi lahko najdemo na spletni strani [9]. Android temelji na Linuxu, zato bomo seveda našli na najnižji ravni (na sliki 2.1) jedro Linux-a. Vendar začnimo na začetku, na sliki 2.1 lahko vidimo različne ravni samega sistema. Na najvišji ravni lahko vidimo same **aplikacije** (angl. Applications). Nivo nižje bomo našli **aplikacijsko ogrodje** (angl. Application Framework), ki vsebuje sledeče elemente:

- Bogat nabor komponent z imenom **Sistem Pogledov** (angl. View System), ki nam omogočajo izgradnjo različnih elementov grafičnega vmesnika.
- **Ponudniki vsebine** (angl. Content Providers) omogočajo deljenje podatkov med aplikacijami.
- **Upravljalec sredstev** (angl. Resource Manager) omogoča dostop do sredstev, ki niso del programske kode (kot so XML datoteke za grafični vmesnik).

- **Upravljalec obvestil** (angl. Notification Manager) omogoča aplikacijam možnost prikazovanja obvestil v statusni vrstici (angl. status bar).
- **Upravljalec aktivnosti** (angl. Activity Manager) upravlja z življenjskim ciklom samih aplikacij in omogoča enostaven dostop do zgodovine zagnanih aktivnosti.

Pod aplikacijskem ogrodjem bomo našli **C/C++ knjižnice** (angl. C/C++ libraries). Knjižnice lahko razvijalci uporabljajo preko Android ogrodja za aplikacije. Spodaj je nekaj izmed temeljnih knjižnic:

- **Sistemska C knjižnica** (angl. System C library):
  Implementacija navadne C sistemske knjižice, ki je izpeljana iz operacijskega sistema BSD.
- **Večpredstavnostne knjižnice** (angl. Media Libraries):
  Knjižnice temeljijo na OpenCORE knjižinici od PacketVideo in omogočajo predvajanje in snemanje številnih priljubljenih avdio in video formatov, kot tudi statične slikovne formate.
- **Upravljalec površin** (angl. Surface Manager):
  Upravlja dostop do podsistema za prikaz in neopazno združuje 2D in 3D grafične plasti iz več aplikacij.
- **LibWebCore**:
  Sodoben pogon za brskanje po spletnih straneh, ki poganja privzeti spletni brskalnik na Androidu in vdelani spletni pogled (del sistema pogledov).
- **SGL**:
  Temeljni 2D grafični pogon.
- **3D knjižnice**:
  Knjižice uporabljajo strojno pospeševanje 3D-grafike (kjer je na voljo) ali optimiziran programski grafični pospeševalnik.

- **FreeType**:
  Bitno in vektorsko upodabljanje pisave.

- **SQLite**:
  Zmogljiv in lahek (angl. lightweight) pogon za relacijske baze, ki je na voljo vsem aplikacijam.

V istem nivoju na predelu označenem z rumeno barvo na sliki 2.1 bomo našli **"Android Runtime"**. Tam je nabor osnovnih knjižic, ki omogočajo večino funkcionalnosti, ki so na voljo v knjižicah programskega jezika Java.

Vsaka aplikacija na Androidu teče v svojem lastnem procesu s svojo lastno kopijo virtualnega stroja Dalvik. Dalvik so napisali tako, da naprava lahko učinkovito poganja več virtualnih strojev naenkrat. Stroj izvaja datoteke v formatu "Dalvik Executable" (.dex), ki je optimiziran za minimalno porabo spomina. Osnovan je na registrih in izvaja razrede, prevedene s pomočjo prevajalnika programskega jezika Java, ki so bili spremenjeni v format ".dex" s pomočjo orodja "dx".

Virtualni stroj Dalvik se zanaša na jedro Linuxa za funkcionalnosti, kot so večnitenje in nizko nivojsko upravljanje s spominom. Android uporablja Linux različice 2.6 za vse osnovne sistemske funkcije.

2.2.2 Struktura aplikacije

V tem razdelku bomo povzeli osnovno strukturo aplikacije. Bolj podrobno o sami strukturi lahko najdemo na spletni strani [10]. Pri pisanju aplikacij za Android sta razvijalcem na voljo programski jezik Java in C/C++. Na voljo za razvoj je tudi JNI (angl. Java Native Interface), ki nam omogoča sočasno uporabo obeh programskih jezikov. Velja tudi opozorilo, da uporaba programskega jezika C ne pomeni avtomatičnega dviga zmogljivosti in tudi same hitrosti aplikacije. Razvojni paket Android SDK (angl. Software Development Kit) prevede vso programsko kodo in vse ostale datoteke iz projekta v Android paket, ki je arhivska datoteka s končnico ".apk". To datoteko

dojema operacijski sistem kot eno aplikacijo.

Ko je aplikacija nameščena na mobilno napravo živi v svojem lastnem varnem t.i. peskovniku (angl. Sandbox). Android je večuporabniški operacijski sistem, ker vsaka aplikacija deluje kot en uporabnik. Sistem ji določi unikatno identifikacijsko številko uporabnika ali "user ID" (številka je poznana samo sistemu). Nato dodeli vsa primerna dovoljenja datotekam, tako da samo identifikacijska številka aplikacije uporablja dodeljenja dovoljenja.

Privzeto se vsaka aplikacija izvaja v svojem procesu. Vsak proces ima svoj lasten virtualen stroj in s tem je vsaka aplikacija izolirana ter nima vpliva na druge. Po potrebi Android zažene proces in ustavi, ko ni več potrebovan ali ko sistem želi pridobiti spomin za druge aplikacije.

Na tak način ima vsaka aplikacija dostop samo do komponent, ki jih potrebuje, in nič več. S tem dobimo zelo varno okolje, v katerem aplikacije ne morejo dostopati do nedovoljenih delov sistema, vendar obstajajo načini za deljenje podatkov med aplikacijami in dostopanje do sistemskih storitev:

- Lahko uredimo, da si dve aplikaciji delita isto identifikacijsko uporabniško številko. S tem pridobi prva aplikacija dostop do datotek druge in obratno. Aplikacije, ki imajo isto uporabniško številko, lahko priredimo tako, da se izvajajo v istem procesu in delijo isti virtualni stroj (aplikacije morajo biti podpisane z istim certifikatom).

- Aplikacija lahko zahteva dovoljenje za dostop do podatkov na mobilni napravi kot so kontakti uporabnika, SMS sporočila, itd. Vsa dovoljenja mora uporabnik odobriti, ko namesti aplikacijo.

Osnovne komponente aplikacije

Osnovne komponente predstavljajo različne točke, skozi katere lahko sistem vstopi v aplikacijo. Vse komponente niso dejanske vstopne točke za uporabnika in nekatere so odvisne druga od druge, ampak vsaka obstaja kot svoja

lastna entiteta in igra specifično vlogo v obnašanju aplikacije. Obstajajo 4 različni tipi komponent:

- **Aktivnosti** (angl. Activities):
  Aktivnost predstavlja en zaslon z uporabniškim grafičnim vmesnikom. Na primer, aplikacija za elektronsko pošto ima lahko eno aktivnost, ki prikaže seznam vseh novih sporočil in drugo za sestavljanje novega sporočila ter tretjo za branje sporočil. Vse aktivnosti delujejo tako, da tvorijo povezano uporabniško izkušnjo, vendar so neodvisne druga od druge. Če aplikacija za elektronsko pošto dovoljuje, lahko druga aplikacija zažene samo eno izmed zgoraj naštetih aktivnosti. Na primer, aplikacija za fotografiranje lahko zažene samo aktivnost, ki omogoča pisanje nove elektronske pošte, tako da uporabnik lahko deli sliko z drugimi.

- **Storitve** (angl. Services):
  Storitev je komponenta, ki teče v ozadju in opravlja dolgotrajne operacije ter nima nobenega uporabniškega vmesnika. Na primer, storitev igra glasbo, medtem ko je uporabnik v drugi aplikaciji. Aktivnost lahko zažene storitev in jo pusti teči ali se pa veže nanjo in s tem sodeluje z njo.

- **Ponudniki vsebine** (angl. Content providers):
  Ponudnik vsebin upravlja s skupnim naborom podatkov aplikacij. Mi lahko shranimo podatke v datotečni sistem, podatkovno bazo, na svetovni splet, ipd. Preko ponudnika vsebine lahko ostale aplikacije delajo poizvedbe ali celo spreminjajo podatke (če jim ponudnik dovoli). Na primer, operacijski sistem Android omogoča ponudnik vsebine, ki upravlja s podatki uporabniških kontaktov. Vsaka aplikacija lahko s primernimi dovoljenji dela poizvedbe po delu ponudnika (kot je “ContactsContract.Data”) in s tem bere ter zapisuje podatke o določeni osebi.

- **Prejemniki sporočil** (angl. Broadcast receivers):
  Prejemnik sporočil je komponenta, ki se odziva na sporočila sistema ali aplikacij. Veliko sporočil je sistemskih, ki sporočajo, da je bil zaslon izklopljen, baterija slaba, ipd. Medtem ko aplikacija lahko sporoča ostalim, da so bili določeni podatki prenešeni in so na voljo za uporabo. Čeprav prejemniki sporočil ne prikažejo uporabniškega vmesnika pa lahko ustvarijo obvestilo v statusni vrstici ter s tem opozorijo uporabnika, da se je zgodil določen dogodek. Običajno je prejemnik sporočil samo "prehod" do drugih komponent in opravlja minimalno delo. Na primer, lahko zažene določeno storitev na podlagi sporočila.

Posebnost Androida je, da vsaka aplikacija lahko zažene komponento druge aplikacije. Na primer, v naši aplikaciji želimo omogočiti uporabniku, da zajame sliko s kamero mobilne naprave. Namesto, da bi razvijali novo aplikacijo, lahko enostavno zaženemo samo aktivnost željene že obstoječe aplikacije. Ko se aktivnost zaključi, vrne fotografijo naši aplikaciji tako, da jo lahko uporabljamo dalje. Prehod med aktivnostmi je za uporabnika neopazen. Skratka Android nima ene same vhodne točke zato ni funkcije "main()".

Aktivnosti, storitve in prejemnike sporočil lahko aktiviramo s pomočjo asinhronega sporočila poimenovanega **Prožilec** (angl. Intent). Prožilci vežejo posamezne komponente med seboj med delovanjem sistema. Delujejo kot nekakšni sli, ki zahtevajo odzive od drugih komponent. Prožilec je ustvarjen s pomočjo objekta tipa Intent, ki definira sporočilo za aktivacijo določene vrste komponente. Ponudnik vsebin ni aktiviran s pomočjo zgoraj opisanega sporočila. Namesto tega se aktivira, ko dobi zahtevo od sporočila tipa ContentResolver. Ta obravnava vse transakcije s ponudnikom vsebine.

Manifest aplikacije (angl. Application Manifest)

Preden lahko Android zažene komponento aplikacije, mora preveriti, ali komponenta obstaja. To naredi z branjem datoteke z imenom "AndroidManifest.xml". Za pravilno delovanje morajo biti v zgoraj opisani datoteki prija-

vljene vse uporabljene osnovne komponente aplikacije. Manifest ni odgovoren samo za komponente aplikacije, ampak tudi za:

- Identifikacijo vseh uporabniških dovoljenj, ki jih aplikacija potrebuje za normalno delovanje.
- Opredelitev minimalne ravni aplikacijskega vmesnika.
- Opredelitev strojne in programske funkcije, ki jih aplikacija potrebuje kot so "bluetooth" storitve, kamera itd.

Skratka naloga manifesta je, da obvesti operacijski sistem, o zahtevah in uporabljenih komponentah naše aplikacije. Na primer, aplikacija ima manifest, ki navaja, da potrebuje za delovanje vsaj Android 2.1, če naša mobilna naprava ne ustreza, potem ne bomo mogli zagnati aplikacije. Na sliki 2.2 lahko vidimo primer osnovne oblike "AndroidManifest.xml" datoteke.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ...>
<application android:icon="@drawable/icon.png" ... >
<activity android:name="com.ExampleActivity"
android:label="@string/label" ... >
</activity>
...
</application>
</manifest>
```

Slika 2.2: Primer AndroidManifest.xml datoteke.

Sredstva aplikacije (angl. Application Resources)

Aplikacija ni samo programska koda, ampak vsebuje tudi vsa ostala potrebna sredstva, ki so ločena od kode, kot so slike, zvočne datoteke in vse kar se

nanaša na vizualno predstavitev aplikacije. Tudi sam uporabniški vmesnik lahko oblikujemo v datotekah XML, ki so vključene v našo aplikacijo kot sredstva.

Za vsako sredstvo, ki ga vključimo, ustvarijo razvojna orodja edinstveno število, ki ga lahko uporabimo za sklicevanje vira iz naše programske kode ali iz ostalih sredstev. Na primer, če naša aplikacija vsebuje sliko z imenom "logo.png"(shranjeno v direktoriju "res/drawable/"), bodo razvojna orodja ustvarila številko z imenom "R.drawable.logo". To bomo lahko uporabili v programski kodi za sklicevanje na sliko in jo vstavili v naš uporabniški vmesnik.

2.2.3 Načini prikazovanja slik

Android ponuja nekaj različnih aplikacijskih programskih vmesnikov za 2D risanje. V praksi lahko uporabljamo dva pristopa (povzeto iz spletne strani [8] za pomoč razvijalcem pri prikazovanju slik):

1. Slikovne elemente in animacije lahko rišemo neposredno v objekt tipa Pogled (angl. View). S tem prikaz obravnava sistemski proces risanja v hierarhiji Pogledov. Če želimo prikazati samo preproste animacije ali statične slike, je to najboljši pristop.

2. Lahko pa rišemo s pomočjo vmesnika, ki se imenuje Platno (angl. Canvas). Platno deluje kot vrsta pretveza za površino, na kateri bomo risali oziroma prikazovali slikovne elemente (vsebuje vse naše "draw" klice). Risanje se izvaja na pomožni objekt z imenom Bitmap, ki ga postavimo v okno. Takšen pristop je primeren, ko mora aplikacija pogosto ponavljati izrise slik. Obstaja več načinov za risanje na Platno:

    - Lahko rišemo v isti niti, kot je naša aktivnost za uporabniški vmesnik. Skratka ustvarimo prilagojeno komponento tipa Pogled v naši postavitvi in kličemo metodo `invalidate()` ter obravnavamo risanje v metodi `onDraw()`.

- Lahko pa rišemo v drugi niti. V tem primeru pa upravljamo s komponento tipa `SurfaceView`, ki riše na Platno tako hitro kot je nit sposobna (ni potrebe po metodi `invalidate()`).

Nalaganje slikovne datoteke

Android nam ponuja razred, poimenovan `BitmapFactory`, ki določa vrsto statičnih metod, ki nam omogočajo nalaganje slikovnih datotek iz različnih virov. Podprte so samo slike formatov JPEG, PNG, GIF, WEBP in BMP. Metode, ki so v `BitmapFactory` nam vzamejo kot argument razred `BitmapFactory.Options`, s katerim lahko določimo, kako je naša slika naložena v pomnilnik. Tam lahko na primer, natančneje nastavimo velikost vzorčenja, ki ga bo razred `BitmapFactory` uporabil, ko bo naložil sliko. Vrednost parametra razreda `BitmapFactory.Options` z imenom `inSampleSize` (velikost vzorčenja) lahko določi, da bo velikost končne slike samo delček originalne velikosti slikovne datoteke. Na primer, če nastavimo parameter `inSampleSize` na 8, bo končna velikost slike približno 1/8 originalne velikosti [1].

Transformacijska matrika

Aplikacijski vmesnik operacijskega sistema Android vsebuje razred `Matrix`. To je transformacijska matrika, ki nam omogoča dodajanje prostorske transformacije pri naši prikazani sliki. Transformacija nam lahko omogoča vrtenje, obrezovanje, povečevanje in druge podobne transformacije naše slikovne datoteke.

Razred `Matrix` predstavlja transformacije s poljem devetih številk, ki jih lahko po potrebi vstavljamo ročno. Da bi razumeli, kako matrika deluje, bomo pokazali ročno pretvorbo matrike.

Vsako število v matriki se nanaša na eno izmed treh koordinat (x, y ali z) za vsako točko v sliki. Za primer vzemimo spodnjo matriko 2.1.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.1}$$

Zgornja vrstica (1, 0, 0) določa, da bo začetna x koordinata transformirana po naslednji formuli: $x = 1x + 0y + 0z$. Kot lahko vidite, postavitev posameznih vrednosti v matriki določa končni rezultat. Zgornja vrstica bo vedno vplivala na x koordinato. Druga vrsta (0, 1, 0) pomeni, da bo y koordinata določena kot $y = 0x + 1y + 0z$. Tretja vrstica (0, 0, 1) pomeni, da bo z koordinata določena s formulo $z = 0x + 0y + 1z$. Z drugimi besedami, ta matrika ne bo naredila nobene transformacije, vse bo tako kot v originalni sliki [1].

2.3 Dropbox

Dropbox je brezplačna storitev, ki nam omogoča enostavno deljenje datotek preko spleta. Danes uporablja spletno storitev Dropbox preko 50 miljonov ljudi po celem svetu [20].

Za Dropbox smo se odločili zato, ker ponuja možnost implementacije učinkovite in hitre sinhronizacije podatkov s pomočjo njihovega aplikacijskega vmesnika, ki ga lahko dobite na sledeči spletni strani [21].

Poglavje 3

Podobne rešitve

V tem poglavju bomo pogledali podobne rešitve iz spletne trgovine Google Play in pregledovalnik slikovnih datotek IrfanView za osebne računalnike.

3.1 Adobe Photoshop Touch

Adobe Photoshop Touch je aplikacija podjetja Adobe Systems (v nadaljevanje Adobe). Podjetje je med drugim odgovorno za programsko opremo za grafično oblikovanje z imenom Adobe Photoshop, ki je namenjena za osebne računalnike. Adobe Photoshop Touch je njihov poskus prenosa programske opreme za grafično oblikovanje na mobilno napravo. Osnovna ideja je bila, da bi omogočili ustvarjalnim strokovnjakom vključitev tabličnih računalnikov v njihovo mobilno delovno okolje. Aplikacija je trenutno izključno za tablične računalnike in ponuja sledeče:

- Uporaba funkcij programa Adobe Photoshop, kot so plasti, filtri, itd.
- Uporaba kamere tabličnega računalnika za zapolnitev območja na izbrani plasti.
- Iskanje in pridobivanje slik s pomočjo iskalnika "Google Image Search".
- Deljenje slik preko socialnega omrežja Facebook.

- Orodja za lažje izbiranje posameznih slikovnih elementov.
- Sinhronizacija projektov s storitvijo Adobe Creative Cloud in nadaljevanje projektov (iz Adobe Photoshop Touch) v Adobe Photoshop.
- Največja ločljivost slike je 1600 × 1600 slikovnih točk.

Poleg običajnega nabora funkcij, ki so skoraj nujne za vse uporabnike Adobe Photoshopa, vidimo še eno omejitev. Vse slikovne datoteke v Adobe Photoshop Touch so omejene na največjo velikost 1600 × 1600 slikovnih točk. Navkljub vsemu, je zelo težko sprejeti takšno skrajno omejitev v svetu, kjer navadni digitalni fotoaparati lahko ustvarijo sliko v ločljivosti 4000 × 3000 ali več.

Iz zgoraj opisane omejitve lahko sklepamo, da je bil to odgovor Adobe-a na težave, ki bi se lahko pojavile s pomnilnikom zaradi prevelikih ločljivosti. Rešitev ni najboljša in ne bi smela biti končna. Ne smemo pozabiti upoštevati še tega, da Adobe Photoshop Touch uporablja takoimenovane plasti (angl. layers). Plasti so slike, ki so postavljene druga na drugo in omogočajo preproste spremembe kompozicije ter drugih podobnih stvari. Lahko domnevamo, da zasedajo več pomnilnika kot pa samo preprosta slika brez zgoraj omenjenih plasti. Aplikacija Adobe Photoshop Touch je na voljo na spletni trgovini Google Play za ceno 7.99 €[15].

3.2 QuickPic

Druga rešitev je aplikacija z imenom QuickPic podjetja Alen Software, ki ponuja sledeče:

- Hitro brskanje po velikem številu slik.
- Enostavno izključevanje/vključevanje direktorijev.
- Enostavno skrivanje izbranih fotografij in video posnetkov pred ostalimi pregledovalniki slik.

- Zaščita slik z geslom.
- Igranje animiranih GIF datotek in standardnih video posnetkov.
- Gladka izkušnja brez zatikanj tako kot pri iPhone.
- Vrtenje, pomanjševanje in obrezovanje slik, ki jih lahko nastavimo za ozadje na mobilni napravi.
- Dodatne funkcije za upravljanje datotek: razvrščanje, preimenovanje, ustvarjanje novih direktorijev, premikanje in kopiranje slik.
- Optimizirano za tablične računalnike z zasloni z visokimi ločljivostmi.

QuickPic je ena izmed bolj popularnih rešitev. Ponuja vse, kar bi lahko pričakovali od aplikacije za pregledovanje slik. Vendar ni nobene možnosti pregledovanja slik v razmerju 1:1. Po kratkem testiranju smo ugotovili, da aplikacija prikazuje samo pomanjšane slike. Sodeč po testiranju je sistem upravljanja s pomnilnikom za slike odporen na različne težave s pomnilnikom, ki bi običajno povzročile kritične napake pri drugih aplikacijah. Sama aplikacija je brezplačna in je dostopna na spletni trgovini Google Play [16].

3.3 Large Image Viewer

Tretja podobna rešitev, ki jo bomo opisali, je slabo poznana aplikacija z imenom Large Image Viewer Android razvijalca Mihaija Preda. Sodeč po opisu podpira formate JPEG, PNG in GIF, ampak na žalost ne moremo posameznih povečati do tega, da vidimo posamezne slikovne točke. Eden izmed problemov je tudi to, da je grafični vmesnik aplikacije neroden in nekonvencionalen. AweView ima uporabniku prijazno sinhronizacijo podatkov s storitvijo Dropbox, medtem ko to ni na voljo za Large Image Viewer. Aplikacija Large Image Viewer je na voljo na spletni trgovini Google Play za ceno 0.62 €[17].

3.4 Za osebne računalnike: IrfanView

IrfanView je delo razvijalca z imenom Irfan Škiljan. Program je zelo hiter, majhen in kompakten grafični pregledovalnik za Windows 9x, ME, NT, 2000, XP, 2003, 2008, Vista in Windows 7. Podpira večjo množico različnih grafičnih formatov [13] in je eden izmed bolj popularnih rešitev za prikazovanje slikovnih datotek na osebnih računalnikih. IrfanView je brezplačen in na voljo na spletni strani [14]. Nekaj lastnosti programa:

- Podpora za Adobe Photoshop filtre.
- Vgrajen iskalnik slikovnih datotek (po metapodatkih).
- Brezizgubna rotacija slikovnih datotek tipa JPEG.
- Možnost spreminjanja barvne globine slike.
- Večjezikovna podpora.

Težko je primerjati aplikacijo AweView z IrfanView zaradi tega, ker imata različni ciljni platformi. Naša aplikacija podpira samo 5 formatov za nizko ločljivost in samo 2 (JPEG, PNG) za prikaz slike v visoki ločljivosti. IrfanView podpira skoraj 80 različnih grafičnih formatov in vsi imajo na voljo prikaz slike v visoki ločljivosti, vendar z omejitvami. Program smo testirali z 2 slikama. Prva je 32-bitna slika, ki je velika 8000 $\times$ 6000 slikovnih točk in s tem zasede približno 183 MB pomnilnika, medtem ko je druga 24-bitna slika, ki je velika samo 1800 $\times$ 1196 slikovnih točk in zasede samo 6 MB pomnilnika. Ko smo poskusili s prvo sliko je IrfanView zasedel samo 137.33 MB in tudi zmanjšal barvno globino slike na 24 bitov. To pomeni, da nalaga celotno sliko v pomnilnik, vendar zmanšuje barvno globino zato, da porabi minimalno količino razpoložljivega pomnilnika. Pri drugi sliki je IrfanView zasedel približno 6 MB pomnilnika in ni spreminjal barvne globine. Sodeč po testiranju je postavljena meja za velikost prikazanih slik. Ko je meja dosežena začne program varčevati s pomnilnikom, drugače pa prikazuje slike

v razmerju 1:1. Če upoštevamo skoraj nepregledno število različnih nastavitev in drugih možnosti za pregledovanje slikovnih datotek, je IrfanView zelo dobra izbira za osebne računalnike.

Poglavje 4

Aplikacija AweView

V tem poglavju bomo podrobneje opisali aplikacijo AweView, ki smo jo razvili v okviru diplomskega dela. Prvo podpoglavje vsebuje splošni opis uporabniškega vmesnika, medtem ko drugo vsebuje rešitve posameznih problemov in tudi možnosti za nadaljnje delo.

4.1 Grafični vmesnik

Vsaka aplikacija takoj pusti boljši vtis na uporabnikih, če ima intuitivni grafični vmesnik, ki je enostaven za uporabo. Pri razvoju smo vedno strmeli v bolj minimalističen pristop pri vmesniku. Uporabnik naj vedno vidi samo tisto, kar lahko uporablja in kar bo pogosto uporabljal. Skratka brez nepotrebnih grafičnih gradnikov v postavitvi, ki med drugim lahko tudi upočasnjuje aplikacijo. Nepravilno uporabljeni grafični elementi (v XML datotekah za postavitev) v Androidu lahko povzročajo probleme s pomnilnikom in druge podobne težave.

4.1.1 Osnovna načela

Pri oblikovanju našega uporabniškega vmesnika smo se držali dveh načel, ki smo ju povzeli po spletni strani [18]:

- **Lepota**
  Aplikacijo oblikujemo tako, da deluje elegantno in estetsko na več ravneh. Prehodi med posameznimi deli so hitri in jasni. Tipografija in ureditev je jasna in smiselna. Vsi grafični gradniki naj se povezujejo v vmesnik, ki bo prijeten na pogled.

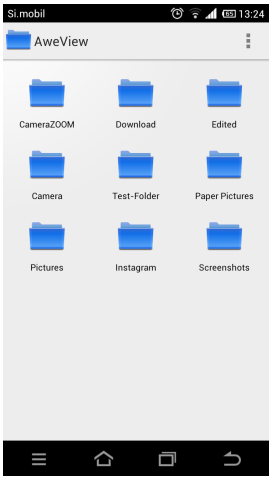
- **Preprostost**
  Aplikacijo razvijamo tako, da olajšamo življenje uporabnikom. Ko našo aplikacijo uporabljajo prvič, je zaželjeno, da takoj intuitivno razumejo najpomembnejše funkcije. Preprosta opravila naj nikoli ne zahtevajo zapletenih postopkov. Zapletena opravila pa naj bodo prilagojena za človeško roko in um. Nenazadnje moramo upoštevati, da so lahko uporabniki iz različnih starostnih skupin in kultur.

Zgoraj opisana načela lahko seveda še razširimo dalje na podrobnejše principe grafičnega oblikovanja, vendar bi se s tem odaljili od tematike diplomskega dela [18].

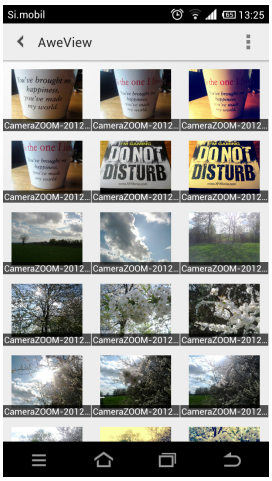
4.1.2 Vmesnik aplikacije AweView

Uporabniški vmesnik aplikacije AweView poskuša biti enostaven za uporabo in intuitiven, čeprav potrebuje še dodatno delo, da bo resnično estetsko prijeten. Na sliki 4.1 lahko vidimo glavni zaslon z albumi slik. V zgornjem desnem kotu je bližnjica do menija, ki vsebuje nadaljnje bližnjice do nastavitev, osvežitve podatkov s pomočjo storitve Dropbox in gumb za prijavljanje oziroma odjavljanje iz storitve Dropbox. S klikom na posamezni album preidemo v mrežni pogled slik iz izbranega albuma.

Na sliki 4.2 lahko vidimo mrežni pogled slik izbranega albuma. S kratkim pritiskom na posamezno datoteko jo lahko odpremo v polnozaslonskem načinu, medtem ko dolgi pritisk odpre okno z osnovnimi podatki izbrane datoteke. Logotip aplikacije služi kot gumb za premik nazaj na zaslon z albumi.



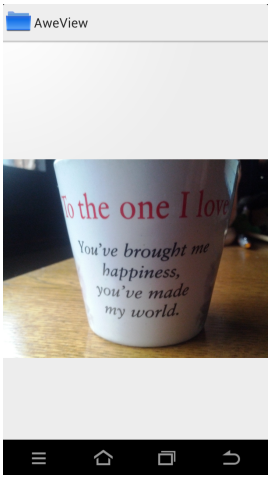
Slika 4.1: Mrežni pogled albumov pri naši aplikaciji.



Slika 4.2: Mrežni pogled slik pri naši aplikaciji.

Na sliki 4.3 lahko vidimo celozaslonski prikaz izbrane datoteke. Če s prstom zamahnemo oziroma potegnemo po zaslonu levo ali desno, preidemo na naslednjo ali prejšnjo sliko iz izbranega albuma. Sliko povečamo oziroma pomanjšamo s pomočjo "pinch-to-zoom" principa. To pomeni, da si s pomočjo dveh prstov izberemo območje in ga povečamo oziroma pomanjšamo. Če želimo povečati sliko postavimo dva prsta na zaslon in ju

razmaknemo, če pa želimo pomanjšati sliko pa zmanjšamo razdaljo med prstoma (ali "uščipnemo"). Na tem delu izgine tudi bližnjica do menija v desnem zgornjem kotu. Na prejšni zaslon lahko preidemo s sistemskim gumbom za nazaj.



Slika 4.3: Celozaslonski pogled slike pri naši aplikaciji.

Na slikah 4.1, 4.3 in 4.3 lahko vidimo na zgornjem delu tudi nov grafični element, ki se imenuje akcijska vrstica (angl. action bar). Element uradno podpirajo samo različice Androida 3.0 in kasnejše, vendar smo ga uspešno prenesli in usposobli tudi za starejše različice. S tem se ohranja enak izgled aplikacije ne glede na različico programske platforme.

Pri oblikovanju izgleda aplikacije smo se poskušali držati splošnih načel. Sam vmesnik je trenutno še osnoven, za splošno uporabo bi potreboval še dodatno delo. S tem imamo v mislih še dodatno izpopolnjevanje prehodov in vizualnih elementov (ikone, gumbi, drsniki itd.) v naši aplikaciji.

4.2 Razvojne rešitve

V tej točki razvoja je naša aplikacija še vedno na stopnji prototipa. Za uporabnike še ni na voljo zaradi mankajočih funkcij in občasnih nestabilnosti. Principi prikazovanja slik visokih ločljivosti so eksperimentalni.

Aplikacijo podpirajo Android različice 2.3 in naprej, s tem smo zajeli največje število uporabnikov. Po potrebi bi lahko končna različica aplikacije brez večjih težav podpirala tudi starejše platforme, ker smo se pri razvoju izogibali funkcij, ki bi nas omejile na določeno različico. To je tudi razlog, da uporabljamo Podporni Paket (angl. Support Package) za Android, ki omogoča uporabo novejših metod in razredov v starejših različicah.

Za prikaz mreže predogledov slikovnih datotek in celozaslonski prikaz posameznih slik uporabljamo novejši pristop z delci (angl. fragments). Delec predstavlja vedenje ali del uporabniškega vmesnika posamezne aktivnosti [12]. V eni aktivnosti lahko združimo več delcev skupaj ali pa lahko isti delec uporabimo v več aktivnostih. Lahko si predstavljamo posamezen delec kot modularen del aktivnosti, ki ima svoj življenski cikel, sprejema svoje vhodne argumente in jih lahko dodajamo ali odstranjujemo, medtem ko aktivnost deluje. S tem pridobimo uporabniški vmesnik, ki je sposoben prestati tudi zamenjavo orientacije zaslona.

Med razvojem smo spoznali, da je upravljanje pomnilnika za slikovne datoteke na operacijskem sistemu Android zelo težavno zaradi omejitve, ki smo jo opisali v uvodu. Navkljub vsemu lahko še vedno uporabimo nekaj različnih pristopov, ki v določenih primerih rešijo težave s pomnilnikom.

4.2.1 Učinkovit pregled slikovnih datotek

Prvi problem smo srečali pri oblikovanju sistema za učinkovit pregled slikovnih datotek. Sprva smo programsko pridobivali sezname datotek s pomočjo Javanske metode `listFiles()` na podlagi podanega objekta tipa `File`. Rešitev je bila zelo počasna in ni ponujala nobene fleksibilnosti za nadaljnji razvoj

(možnost implementacije iskalnika po slikah). Zato smo se odločili za uporabo ponudnika vsebine z imenom `MediaStore`, ki deluje kot vmesnik za že obstoječi datotečni sistem operacijskega sistema Android. Nova rešitev je bolj stabilna in robustnejša kot stara in ponuja fleksibilnost za nadaljnji razvoj sistema. Edina slabost je, da moramo sedaj ob vsaki osvežitvi podatkov skrbeti, da se ponudnik vsebine datotečnega sistema tudi osveži.

Drugi problem smo srečali pri oblikovanju sistema za ustvarjanje grafičnih predogledov slikovnih datotek. Od samega začetka smo uporabljali dodatne niti in s tem porazdelili delo in razbremenili glavno nit za uporabniški vmesnik. Sprva smo istočasno nalagali več datotek, vendar se je to kasneje izkazalo za neučinkovit pristop. Sedaj serijsko nalagamo grafične predoglede slikovnih datotek. Ena izmed razlik v pristopih je v principu upravljanja pomnilnika, vendar se pri obeh rešitvah poslužujemo LRU (angl. Least Recently Used) algoritma. To pomeni, da za vsak vstavljen podatek vodimo evidenco, kdaj je bil nazadnje uporabljen. V primeru, ko je struktura polna, bo nov podatek zamenjan s tistim, ki ga najdlje nismo uporabljali. Pri prvem pristopu smo uporabljali dve podatkovni strukturi:

1. podatkovna struktura tipa `LinkedHashMap` ⟨`String, Bitmap`⟩:
Predstavlja podatkovno strukturo, ki ima omejeno število shranjenih elementov in shranjuje pare ⟨niz, slikovna datoteka⟩. Niz je bil v našem primeru ime datoteke, slikovna datoteka pa je bila naša ikona. Za naš primer smo preoblikovali funkcijo za odstranjevanje najstarejših elementov v podatkovni strukturi. Nova funkcija odstranjene elemente dodaja v novo podatkovno strukturo, ki jo bomo opisali spodaj.

2. podatkovna struktura tipa `ConcurrentHashMap` ⟨`String, SoftReference` ⟨`BitmapDrawable`⟩⟩:
Predstavlja podatkovno strukturo, ki ima omejeno število shranjenih elementov (v našemu primeru je to polovična velikost prve strukture) in shranjuje pare ⟨niz, objekt tipa `SoftReference`⟩. Mehka referenca (angl. Soft Reference) hrani referenco na dejansko slikovno datoteko, ki ni dovolj močna,

da bi prisilila datoteko, da ostane v spominu. To daje vzvod Javanskemu smetarju (angl. Garbage Collector), da sam presodi o dosegljivosti objekta.

Obe strukturi sta vezani na časovnik, ki jih avtomatsko počisti po določenem časovnem intervalu. Z vsakim elementom, ki ga dodamo v strukturo ponastavimo časovnik in s tem ohranjamo minimalno porabo pomnilnika. Edini problem pri takemu pristopu je, da Javanski smetar preveč agresivno čisti šibke reference na naše slikovne datoteke. To pomeni, da se naše podatki ohranjajo za zelo kratek čas in so hitro funkcionalno neuporabni.

Problem smo rešili z uporabo strukture z imenom `LruCache⟨String, Bitmap⟩`. Še vedno uporabljamo podoben koncept kot zgoraj z edino razliko, da sedaj ne uporabljamo časovnikov za čiščenje podatkovne strukture. Struktura `LruCache` avtomatsko počisti podatke, ko je izbrani limit presežen. Razlika je, da imamo sedaj podatkovno strukturo, ki je bolj strogo omejena in stabilna. Enak pristop, vendar manj robusten smo uporabili za hranjenje podatkov na zunanjem pomnilniku.

Na zadnji problem smo naleteli pri ustvarjanju ikon. Ponudnik vsebine MediaStore ima svojo lastno funkcijo za pridobivanje ikone iz podatkovne baze, ki je pogosto neuspešno naložila ikone slikovnih datotek tipa GIF in WEBP. Zato smo razvili svojo lastno različico ustvarjanja ikon za zgornja slikovna formata, ki je bolj stabilna in hitrejša od funkcije ponudnika vsebin. Ko smo testirali čas nalaganja ikone, je naša rešitev prehitela funkcijo razreda MediaStore za približno 150 ms, pri nalaganju ikone slike tipa WEBP. Čas nalaganja metode razreda MediaStore je znašal približno 450 ms, ker je metoda pogosto spodletela so rezultati nenatančni, medtem ko je bil naš čas približno 300 ms. Povprečen uporabnik bi, po ocenah sodeč, bil pripravljen čakati največ 2 sekundi, tako da sta oba časa še vedno sprejemljiva. Pri datotekah slikovnega formata GIF je bil čas nalaganja isti za oba pristopa.

4.2.2 Celozaslonski prikaz slikovnih datotek

Klasična rešitev

Če želimo prikazati slikovne datoteke v operacijskem sistemu Android, imamo 2 možnosti. Prva možnost je uporaba že vnaprej pripravljenega razreda z imenom `ImageView`. Tak pristop ne ponuja nobene možnosti uporabe poslušalcev za zaznavanje dotikov. Res je, da mu lahko nastavljamo transformacijske matrike, vendar je njegova uporaba zelo omejena. Druga možnost leži v uporabi razreda `WebView`. S tem pristopom pridobimo tudi opcijo za uporabo poslušalcev, vendar moramo v manifestu aplikacije zahtevati dovoljenje za uporabo medmrežja, ker je tak pristop namenjen za interakcije s spletom. Oba zgoraj opisana pristopa sta seveda neprimerna, ker še vedno ohranjata celotno slikovno datoteko v spominu. Res je, da jima lahko podamo samo del slike ali pomanjšano sliko, vendar ne moremo aktivno spremljati dejanske operacije za povečavo in premikanje slike v samemu pogledu.

Problem smo rešili z razširjanjem razreda `View` in implementacijo potrebnih poslušalcev za zaznavanje dotikov. Za premikanje in povečevanje slik uporabljamo transformacijsko matriko, ki jo nastavimo, ko nam nit za nalaganje slikovnih datotek vrne predogled slike. Ker želimo imeti sliko poravnamo, uporabimo metodo razreda `Matrix` z imenom `setRectToRect`. Metodi podamo tri argumente:

- Prvi argument je objekt, ki vsebuje 4 števila, ki predstavljajo koordinate štirih robov pravokotnika oziroma naše slike.
- Drugi argument je podoben zgoraj opisanemu objektu, vendar ne predstavlja naše slike, ampak zaslon.
- Tretji argument nadzoruje, kako se bo naša slika poravnala na podanem zaslonu.

Seveda poskrbimo tudi za vse ostale vrednosti, ki jih bomo potrebovali v našemu razredu. Našo transformacijsko matriko zapišemo kot tabelo vrednosti tipa `float`. V tabeli se nahaja 9 vrednosti, vendar nas zanimajo

samo štiri. Faktorja povečave po x in y osi se nahajata na mestih 1 in 5, ker mi povečujemo enakovredno po višini in širini, sta oba enaka. Na mestih 3 in 6 se nahajata x in y koordinati zgornjega levega kota naše prikazane slike.

S pomočjo zgoraj opisanih vrednosti lahko pridobimo začetne točke naše slike in začetni faktor povečave. Funkcija `setRectToRect` je v tem primeru odgovorna za nastavitev zgoraj opisane vrednosti.

V poslušalcu za premike slike enostavno izračunamo razdaljo med dvema dotikoma in vstavimo nove vrednosti v našo matriko, tako kot je prikazano v kodi 4.1, ki zahteva dodatno pojasnilo. `mScale` je naš faktor povečave, `currPos.x` in `currPos.y` sta x ter y vrednosti trenutne točke naše slike. Metodi `getScaledSizeX()` in `getScaledSizeY()` vrneta širino in višino, katere je vsaka deljena z 2. Na koncu samo še kličemo metodo `invalidate()` in s tem prisilimo ponovni izris s spremenjeno matriko.

```
mMatrix.reset();
mMatrix.postTranslate(-imageHolder.getScaledSizeX(),
-imageHolder.getScaledSizeY());
mMatrix.postScale(mScale, mScale);
mMatrix.postTranslate(currPos.x, currPos.y);
```

Koda 4.1: Vstavljanje novih vrednosti v našo transformacijsko matriko.

V poslušalcu za povečavo uporabljamo isti pristop z edino razliko, da pozicijo izračunamo tako, kot je prikazano v kodi 4.2, ker moramo upoštevati faktor povečave pri naših premikih. `getFocusX()` in `getFocusy()` vrneta x- in y-koordinato točke, ki je točno med dvema prstoma. Na koncu samo ponovimo zgoraj opisani postopek.

```
float diffX = detector.getFocusX() - currPos.x;
float diffY = detector.getFocusY() - currPos.y;
diffX = diffX * scale - diffX;
diffY = diffY * scale - diffY;
currPos.x -= diffX;
currPos.y -= diffY;
```

Koda 4.2: Izračun nove pozicije naše slike v poslušalcu za povečavo.

Pri kodi 4.3 lahko vidimo metodo, ki skrbi za pravilne izrise naše slikovne datoteke. Celoten postopek je zelo preprost in samo doda novo transformacijsko matriko našemu objektu tipa `Canvas` in izriše podano sliko.

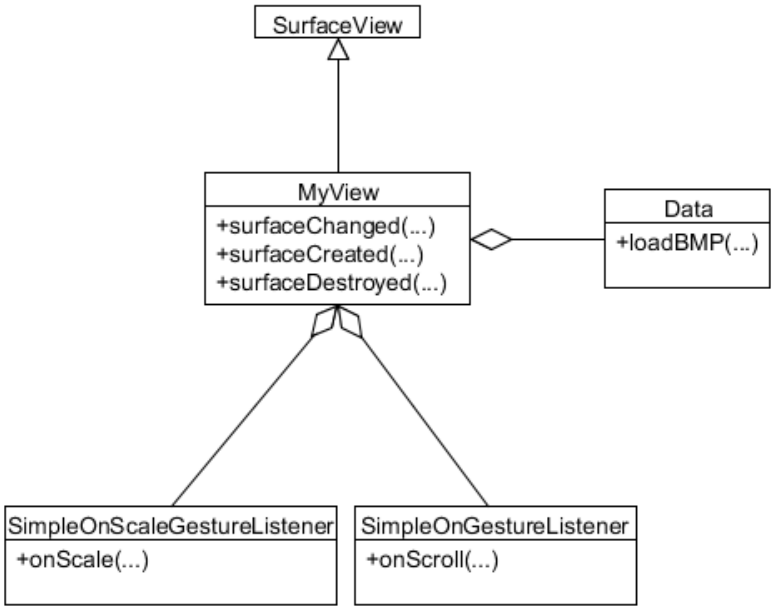
```
@Override
public void onDraw(Canvas canvas) {
int saveCount = canvas.getSaveCount();
canvas.save();
canvas.concat(mMatrix);
canvas.drawBitmap(imageHolder.getCurrentImage(),
0, 0, null);
canvas.restoreToCount(saveCount);
}
```

Koda 4.3: Metoda `onDraw` našega razširjenega razreda.

Prototipna rešitev

Naš koncept deluje tako, da v osnovi prikažemo pomanjšano sliko ali predogled slike, ko uporabnik poveča sliko pa začnemo z dekodiranjem posameznih delov izbranega območja. Predogled slike vedno hranimo v pomnilniku zaradi tega, ker je lahko pri nekaterih slikovnih formatih delno nalaganje zelo počasno in s tem med nalaganjem prikažemo pomanjšano sliko. Za branje datotek različnih slikovnih formatov sedaj uporabljamo C-kodo s

pomočjo ogrodja JNI in razred `BitmapRegionDecoder`, ki podpira samo formata JPEG in PNG. Večji del formatov lahko obravnavamo samo s pomočjo programskega jezika C, ker bi bil prenos v Javo težaven zaradi strukture slikovnega formata, ker Java ne omogoča neposrednega upravljanja s pomnilnikom. Za lažje razumevanje naše rešitve bomo prikazali postopek delovanja pri primeru branja 16-bitne slikovne datoteke formata BMP brez barvne palete. Podatki o slikovnih točkah v tem primeru niso stisnjeni. Za prikaz slikovne datotek uporabljamo prirejen razred imenovan `MyView`, ki je razširjen iz razreda `SurfaceView`, ki uporablja svojo nit za vse izrise naše slikovne datoteke. Res je, da bi v tem primeru lahko uporabili razred `View`, vendar bi to bil slab pristop, ker `View` izrisuje slikovne datoteke v glavni niti, ki skrbi za uporabniški vmesnik. To bi bil prevelik napor za glavno nit in bi se aplikacija v našem primeru odzivala zelo počasi.



Slika 4.4: Razredni diagram prikaza slik visokih ločljivosti.

Kot smo že zgoraj omenili, uporabljamo prirejen razred, ki je razširjen iz `SurfaceView`. Razred nima metode `onDraw`, ima pa metode `surfaceChanged`, `surfaceCreated` in `surfaceDestroyed`. Za lažje upravljanje naše slike in njenih metapodatkov uporabljamo prirejen razred imenovan `Data`, ki vsebuje pomembne podatke o trenutni sliki, metode za delno nalaganje slikovnih datotek in vse objekte tipa `Bitmap`. V metodi `surfaceChanged` nastavimo osnovne parametre, kot so velikost zaslona, velikost slikovne datoteke, ipd. V metodi `surfaceCreated` zaženemo našo nit za izrise, medtem ko v metodi `surfaceDestroyed` po potrebi ustavimo nit. Za poslušalce za zaznavo povečave in premikanja slike uporabljamo razreda `SimpleOnGestureListener` in `SimpleOnScaleGestureListener`. Za lažjo vizualizacijo imamo na sliki 4.4 diagram zgoraj opisanih razredov.

V niti za izrise lahko s pomočjo objekta tipa `SurfaceHolder` (razreda `SurfaceView`) urejamo slikovne točke na naši risalni površini. Tam tudi kličemo našo metodo za delno nalaganje slikovne datoteke, medtem pa prikažemo grobi predogled izbranega območja.

Za delno nalaganje slikovne datoteke potrebujemo koordinate, ki so relativne glede na našo slikovno datoteko. Relativne koordinate našega izbranega območja, dobimo s pomočjo metode `updateRelativePositions` 4.4, ki sprejme dva argumenta. Argumenta sta x in y koordinati mesta dotika ali središčne točke med dvema prstoma (če gre za povečavo). Metoda uporabi klasični pristop s transformacijsko matriko, potem pa nadaljuje z izračuni relativnih koordinat glede na informacije iz naše matrike. Metodo kličemo v poslušalcih za zaznavo dotikov, kjer ob vsaki spremembi prednastavimo vrednosti, ki nas zanimajo in prekinemo možno prejšnje delno nalaganje slikovne datoteke.

Za primer, če želimo naložiti datoteko formata BMP, kliče nit za izrise C funkcijo `loadBMP`, ki sprejme 7 argumentov in vrne izbrano območje kot objekt tipa `Bitmap`. Prvi argument je absolutna pot do slikovne datoteke na zunanjem pomnilniku. Argumenta 2 in 3 predstavljata relativne koordinate

```
private void updateRelativePositions(float x, float y){
imageMatrix.reset();
imageMatrix.postTranslate(-imageHolder.getSizeX(),
-imageHolder.getSizeY());
imageMatrix.postScale(mScale, mScale);
imageMatrix.postTranslate(currPos.x, currPos.y);
imageMatrix.getValues(matrixValues);
relative.x = (x - matrixValues[Matrix.MTRANS_X])
/ matrixValues[Matrix.MSCALE_X];
relative.y = (y - matrixValues[Matrix.MTRANS_Y])
/ matrixValues[Matrix.MSCALE_Y];
zoomedArea.left = Math.round(relative.x - span.x
/ (mScale * 2));
zoomedArea.top = Math.round(relative.y - span.y
/ (mScale * 2));
zoomedArea.right = Math.round(zoomedArea.left +
span.x / mScale);
zoomedArea.bottom = Math.round(zoomedArea.top +
span.y / mScale);
}
```

Koda 4.4: Metoda za izračun relativnih koordinat.

območja, ki ga želimo prikazati. Argumenta 4 in 5 predstavljata širino in višino izbranega območja. Zadnje 4 vrednosti smo dobili s pomočjo metode 4.4, ki smo jo pojasnili zgoraj. Zadnja dva argumenta predstavljata širino in višino našega zaslona.

Funkcija `loadBMP` prebere slikovno datoteko, po potrebi popravi velikost izbranega območja in primerno napolni naš prazen objekt tipa `Bitmap`. Pomembne informacije pri datotekah BMP najdemo na začetku datoteke. Prvi del je vedno velik 14 bajtov in vsebuje 5 polj. Nas zanima samo zadnje polje, ki je veliko 4 bajte. Tam najdemo vrednost odmika do začetka podatkov o slikovnih točkah. Zatem lahko sledi, glede na tip formata BMP, del imenovan `BITMAPINFOHEADER` ali del imenovan `BITMAPCOREHEADER`. Razlikujeta se v ve-

likosti in s tem tudi v številu podatkov, ki jih vsebujeta. Za naš primer bomo predpostavili, da datoteka BMP vsebuje del, imenovan `BITMAPINFOHEADER`, ki je bolj razširjen format kot `BITMAPCOREHEADER`. Iz dela `BITMAPINFOHEADER` bomo lahko pridobili podatke o velikosti slike, načinu stiskanja in številu bitov na slikovno točko. V primerih, ko je število bitov na slikovno točko manjše ali enako 8 bitov potrebujemo še barvno paleto, ki sledi po `BITMAPINFOHEADER`, čeprav je lahko barvna paleta prisotna tudi pri ostalih primerih. V primerih, ko je višina slikovne datoteke negativna, so podatki o slikovnih točkah urejeni od zgoraj do spodaj namesto od spodaj do zgoraj. Pri našemu primeru bomo nakazali branje 16 bitne slikovne datoteke tipa BMP. Višina ne bo negativna in s tem beremo podatke normalno. Podatki o slikovnih točkah ne bodo stisnjeni, kar bo omogočalo lažje razumevanje našega pristopa.

Formula 4.1 *Formula za izračun števila bajtov na posamezno vrstico [3].*

$$stevilo BitovNaVrstico = \frac{(\frac{sirina \times stevilo Bitov + 7}{8}) + 3}{4} \tag{4.1}$$

Število bajtev na posamezno vrstico (angl. bytes per row) dobimo s pomočjo formule 4.1. Velja omeniti, da računamo po celoštevilski aritmetiki, kar pomeni, da rezultat zaokrožujemo. Spremenljivka "steviloBitov" predstavlja število bitov na slikovno točko. Spremenljivka "sirina" predstavlja širino naše slikovne datoteke. To je prvi korak pri pridobivanju naše tabele vrednosti slikovnih točk. Zadnja stvar so seveda vrednosti posameznih slikovnih točk, ki so shranjene kot zaporedje vrednosti treh barv (rdeča, modra in zelena). Posamezne slike se s tem razlikujejo glede na število bitov na slikovno točko. To pomeni, koliko bajtov predstavlja posamezno barvo. V našem primeru imamo 16-bitno slikovno datoteko. Vsaka slikovna točka je predstavljena kot vrednost tipa `Integer` velika 2 bajta [3].

Sedaj, ko poznamo, kako je format BMP sestavljen, lahko preberemo izbrano območje v slikovni datoteki in razvrstimo posamezne vrednosti v našo razširjeno tabelo vrednosti. Slikovna točka je predstavljena kot trije zaporedni vnosi v tabeli. Na primer, prva slikovna točka ima barvne vrednosti

shranjene na mestih 1, 2 in 3 (rdeča, zelena, modra). V primeru barvne palete pa bi vsak vnos predstavljal mesto v barvni paleti, ki vsebuje primerno številsko zaporedje za naše vrednosti.

```
void fill_pixels(AndroidBitmapInfo *info, void *pixels,
unsigned char *imageData){
int i = 0, sum = 0;
for (i = 0; i < info->height; i++) {
        uint16_t*  line = (uint16_t*)pixels;
        uint16_t*  line_end = line + info->width;
        while (line + 1 <= line_end){
                *line ++=make565(imageData[sum],
                                 imageData[sum+1],
                                 imageData[sum+2]);
                sum+=3;
        }
        pixels = (char*)pixels + info->stride;
}
}
```

Koda 4.5: Funkcija za polnjenje podanega objekta tipa `Bitmap`.

Ko imamo razširjeno tabelo vrednosti zapolnjeno, lahko še preverimo, če velikost izbranega območja ni prevelika za razpoložljiv pomnilnik in jo po potrebi pomanjšamo. Izbrano področje naložimo s pomočjo relativnih koordinat, ki smo jih izračunali z metodo 4.4. V primeru, ko je velikost manjša od zaslona pa povečujemo, vendar vedno brez glajenja robov. Zadnji korak v naši C funkciji `loadBMP` je, da ustvarimo objekt tipa `Bitmap` in ga s pomočjo funkcije 4.5 napolnimo. Zgornja funkcija uporablja še sledečo funkcijo 4.6, ki skrbi za primerne bitne zamike vrednosti posameznih barv. V našem primeru imamo 16-bitno sliko, zato uporabljamo format `RGB_565`. To pomeni, da je 5 bitov namenjeno rdeči barvi, 6 bitov zeleni in ponovno 5 bitov modri.

```
uint16_t  make565(int red, int green, int blue)
{
return (uint16_t) ((red >> 3) << 11) |
                  ((green >> 2) << 5) |
                  ((blue >> 3));
}
```

Koda 4.6: Funkcija, ki s pomočjo bitnih pomikov nastavlja vrednosti slikovnih točk.

Preden kličemo funkcijo 4.5, moramo klicati še funkcijo `AndroidBitmap_lockPixels`, s katero "zaklenemo" trenutne podatke o slikovnih točkah. To nam omogoča, da lahko izvajamo operacije nad slikovnimi točkami v objektu tipa `Bitmap`.

Sedaj samo še vrnemo objekt razredu za prikaz. S tem grobi predogled slike izgine in prikaže se naša slikovna datoteka. Ob vsaki zaznavi premika slike ali povečave se zgoraj opisani postopek ponovi.

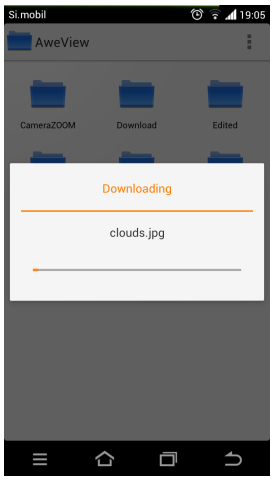
Za ostale formate bi se spremenila samo C funkcija za delno nalaganje slikovnih datotek. Potrebno bi bilo obravnavati vsak primer posebej in primerno ravnati. Že pri formatu BMP se lahko pojavijo težave, ko predstavimo različne oblike stiskanja podatkov, barvne globine in barvno paleto. Način, kako obravnamo podatke o slikovnih točkah se lahko spremeni, zato obravnavamo vsak primer posebej. Razred, ki je razširjen iz `SurfaceView` in poslušalci za zaznavo dotikov, so za ostale primere popolnoma isti.

Problemi pri naši konceptni rešitvi ležijo v nestabilnosti funkcij za branje različnih slikovnih formatov. Zato aplikacija še ni pripravljena za splošno uporabo. V svetu računalniške tehnologije obstaja ogromno različnih slikovnih formatov. Ker ima vsak format svoje posebnosti, kot npr. kompresijski algoritem, je zelo težko razviti univerzalno rešitev, ki bi delovala stabilno in učinkovito.

Z našo rešitvijo moramo obravnavati vsak slikovni format posebej, ker se med seboj običajno zelo razlikujejo. To pomeni, da bi morali razviti za vsak format funkcijo, ki bi bila sposobna naložiti samo del slike in ga prikazati. Problem ni v samem načinu prikaza slik, ampak v izdelavi učinkovitega ogrodja za dekodiranje slikovnih datotek.

Če postavimo število formatov v perspektivo, lahko navedemo popularni pregledovalnik slik IfranView. Na spletni strani [13] so navedeni vsi grafični formati, ki jih njihov pregledovalnik podpira. Če odštejemo nekaj formatov, ki so namenjeni bolj za predstavitve besedil, je podprtih približno 65 slikovnih formatov. Za naš primer bi to bilo 65 različnih implementacij funkcij za delno nalaganje slikovnih datotek.

4.2.3 Sinhronizacija podatkov



Slika 4.5: Pogovorno okno, ki kaže stanje pri sinhronizaciji podatkov.

V uvodu smo omenili tudi sinhronizacijo datotek s pomočjo storitve Dropbox. Trenutna sinhronizacija deluje, osveži vse naše datoteke in naloži nove. To smo lahko dosegli s pomočjo novega aplikacijskega vmesnika, ki doda tudi funkcijo z imenom `delta`, ki vrača samo spremenjene oziroma nove datoteke,

zaradi tega ni več potrebe po hranjenju informacij v podatkovni bazi o posameznih datotekah. Vse skupaj izvedemo v niti v ozadju ter s tem razbremenimo osnovno nit. Na sliki 4.5 lahko vidimo pogovorno okno, ki je prikazano med sinhronizacijo podatkov. Seveda je tudi tukaj možnost izboljšav. Hitrost prenosa in iskanja novih datotek bi lahko še dodatno pospešili ter s tem ponudili hitro sinhronizacijo.

Poglavje 5

Sklepne ugotovitve

V okviru diplomskega dela smo razvili aplikacijo za pregledovanje slikovnih datotek. Za aplikacijo smo razvili učinkovit sistem upravljanja s predogledi slik, ki hitro deluje ne glede na število datotek. S pomočjo aplikacijskega vmesnika za Dropbox smo razvili hitro in učinkovito selektivno sinhronizacijo podatkov. Nenazadnje smo začrtali konceptno rešitev problema celozaslonskega prikazovanja slik visokih ločljivosti, ki trenutno podpira samo formata BMP, JPEG in PNG, vendar je začrtan tudi koncept za preostale slikovne formate. Naš pristop odpre vrata nadaljnjemu delu na tem področju in bo morda nekoč pomagal mobilnim napravam prodreti tudi v običajna delovna mesta strokovnjakov v tekstilni industriji, grafičnim oblikovalcem in fotografom.

Pri razvoju aplikacije smo kar nekaj časa porabili za raziskavo področja za prikazovanje slikovnih datotek na programski platformi Android. Preizkušali smo različne pristope, ki so nas popeljali od slabo dokumentiranega razvojnega paketa "Android NDK" do rešitve, ki smo jo predstavili v našem diplomskem delu. Vredno je omeniti, da je največje probleme povzročala implementacija sinnhronizacije datotek s storitvijo Dropbox. Težave so bile običajno v slabo dokumentiranem aplikacijskem vmesniku. Obstajata dve metodi, ki po Dropboxovi dokumentaciji vračata objekta istega tipa, vendar se je to izkazalo za neresnično. Mnogokrat metode preprosto niso delovale.

Te težave so delno popravili v novejših različicah.

Ena izmed težjih nalog je bilo zagotovo samo oblikovanje izgleda aplikacije. Ker smo strmeli k dovršenosti pri izgledu, je po našem mnenju aplikacija zelo pomankljiva. Potrebno je še dodatno delo na vmesniku, ki bi lahko ponujal bolj unikatno in inovativno izkušnjo samemu uporabniku. S tem imamo v mislih še dodatno izpopolnjevanje vizualnih elementov (ikone, gumbi, drsniki itd.) v naši aplikaciji.

Preden smo se lotili razvoja naše aplikacije, nismo imeli nobenih izkušenj s programsko platformo Android. Srečali smo se s težavami, ki so nevidne za običajnega uporabnika, npr. učinkovito upravljanje pomnilnika, slaba dokumentacija (za uporabo C kode), ipd. Morali smo se tudi podučiti o sami strukturi platforme. Takšno delo je bilo koristno v mnogih pogledih. Dotaknili smo se vseh večjih delov razvoja, od oblikovanja izgleda aplikacije do postavljanja relacijskih podatkovnih baz. Res je, da aplikacija še ni pripravljena za množice uporabnikov, manjkajo še funkcije, ki so skoraj nujne za današnjo mobilno programsko opremo za pregledovanje ali procesiranje slik:

- Ni možnosti nadzora datotek (kopiranje, premikanje, preimenovanje), ki ga ponujajo ostale aplikacije.
- Ni filtrov ali drugih načinov preoblikovanja slik.
- Ni možnosti nastavljanja slik za ozadje mobilne naprave.
- Ni možnosti iskanja datotek (po metapodatkih).

Vendar ne smemo pozabiti, da trenutno ni nobene smiselne rešitve za prikazovanje slik visokih ločljivosti na programski platformi Android. Delo še ni končano, je pa zelo dobra smernica za nadaljnji razvoj.

Literatura

[1] S. V. Every. *Pro Android Media: Developing Graphics, Music, Video, and Rich Media Apps for Smartphones and Tablets*. Apress, 2010.

[2] S. Komatineni, D. MacLean, S. Hashimi. *Pro Android 3*. Apress, 2011, pogl. 1.

[3] J. Miano. *Compressed Image File Formats: JPEG, PNG, GIF, XBM, BMP*. Addison-Wesley Professional, 1999.

[4] W. B. Pennebaker, J. L. Mitchell. *JPEG still image data compression standard*. Springer, 1993, pogl. 1.

[5] R. Rabbat (2010) "WebP, a new image format for the Web". Dostopno na:
http://googlecode.blogspot.com/2010/09/webp-new-image-format-for-web.html

[6] G. Roelofs (1997) "History of the Portable Network Graphics (PNG) Format". Dostopno na:
http://www.libpng.org/pub/png/pnghist.html

[7] R. H. Wiggins II, H. C. Davidson, H. R. Harnsberger, J. R. Lauman, P. A. Goede. "Image File Formats: Past, Present, and Future", v reviji RadioGraphics, št. 21, str. 789-798, Maj 2001. Dostopno na:
http://radiographics.highwire.org/content/21/3/789.html

[8] (2012) Canvas and Drawables. Dostopno na:
http://developer.android.com/guide/topics/graphics/2d-graphics.html

[9] (2012) What is Android?. Dostopno na:
http://developer.android.com/guide/basics/what-is-android.html

[10] (2012) Application Fundamentals. Dostopno na:
http://developer.android.com/guide/topics/fundamentals.html

[11] (2012) Android Supported Media Formats. Dostopno na:
http://developer.android.com/guide/appendix/media-formats.html

[12] (2012) Fragments. Dostopno na:
http://developer.android.com/guide/topics/fundamentals/fragments.html

[13] (2012) Podprti formati programa IrfanView za pregledovanje slikovnih datotek. Dostopno na:
http://www.irfanview.com/main_formats.html

[14] (2012) IrfanView. Dostopno na:
http://www.irfanview.com/

[15] (2012) Adobe Photoshop Touch. Dostopno na:
https://play.google.com/store/apps/details?id=air.com.adobe.pstouch

[16] (2012) QuickPic. Dostopno na:
https://play.google.com/store/apps/details?id=com.alensw.PicFolder

[17] (2012) Large Image Viewer. Dostopno na:
https://play.google.com/store/apps/details?id=i.v

[18] (2011) Android Design. Dostopno na:
http://developer.android.com/design/index.html

[19] (2011) Gartner Says Sales of Mobile Devices Grew 5.6 Percent in Third Quarter of 2011; Smartphone Sales Increased 42 Percent.

Dostopno na:
http://www.gartner.com/it/page.jsp?id=1848514

[20] (2012) About Dropbox.
Dostopno na:
https://www.dropbox.com/about

[21] (2012) Development kits and documentation.
Dostopno na:
https://www.dropbox.com/developers/reference/sdk