UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Sašo Mežnar

Razvoj mobilne aplikacije na platformi Android

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Ljubljana, 2012

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Sašo Mežnar

Razvoj mobilne aplikacije na platformi Android

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: viš. pred. dr. Igor Rožanc

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: 00226/2012
Datum: 03.04.2012

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

Kandidat: **SAŠO MEŽNAR**

Naslov: **RAZVOJ MOBILNE APLIKACIJE NA PLATFORMI ANDROID**

DEVELOPMENT OF A MOBILE APPLICATION FOR THE ANDROID PLATFORM

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

V diplomski nalogi celovito predstavite razvoj mobilnih aplikacij na platformi Android. V ta namen najprej predstavite specifične tehnologije in orodja s tega področja. V osrednjem delu podrobno opišite metodološki postopek razvoja primera aplikacije Avtodroid, ki na zelo uporaben način omogoča vodenje evidence stoškov osebnega avtomobila. Nalogo zaključite s predstavitvijo uporabe aplikacije in idejami za izboljšave le-te.

Mentor:

viš. pred. dr. Igor Rožanc

Dekan:

prof. dr. Nikolaj Zimic

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani **Sašo Mežnar**,
z vpisno številko **63070201**,

sem avtor diplomskega dela z naslovom:

Razvoj mobilne aplikacije na platformi Android

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom
  viš. pred. dr. Igorja Rožanca

- 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 22.06.2012 Podpis avtorja:

Zahvala

Zahvaljujem se mentorju viš. pred. dr. Igorju Rožancu za pomoč in nasvete pri izdelavi diplomske naloge.

Zahvaljujem se staršem in punci Tjaši, ki smo mi v času študija vseskozi stali ob strani ter me spodbujali.

Kazalo

Seznam uporabljenih kratic in simbolov

AAPT	angl. *Android Asset Packaging Tool* – orodje za izdelavo datoteke, ki predstavlja celotno Android aplikacijo
ADB	angl. *Android Debug Bridge* – razhroščevalnik Android aplikacij
ADT	angl. *Android Development Tools* – orodja za razvoj Android aplikacij
AOSP	angl. *Android Open Source Project* – odprtokodni Android projekt
API	angl. *Application Programming Interface* – vmesnik za programiranje aplikacij
AVD	angl. *Android Virtual Device* – Android navidezna naprava
BBM	angl. *BlackBerry Messenger* – družabno omrežje BlackBerry
C2DM	angl. *Cloud To Device Messaging* – sporočanje v smeri strežnik - naprava
DDMS	angl. *Dalvik Debug Monitor Service* - sistem za povezavo emulatorja ali fizične mobilne naprave z razvojnim okoljem
DTD	angl. *Document Type Definition* – jezik za opis strukture XML dokumenta
DVM	angl. *Dalvik Virtual Machine* – Dalvik navidezna naprava
EPL	angl. *Eclipse Public Licence* – javna licenca Eclipse
GPS	angl. *Global Positioning System* – globalni sistem določanja položaja
GSM	angl. *Global System for Mobile Communication* – globalni sistem mobilnih komunikacij
IDE	angl. *Integrated Development Environment* – integrirano razvojno okolje
IrDA	angl. *Infrared Data Association* – prenašanje podatkov na osnovi infrardeče povezave
ISM	angl. *Industrial, Scientific and Medical* – frekvenčno območje za uporabo na področju industrije, znanstvenega raziskovanja in medicine
J2ME	angl. *Java 2 Micro Edition* – platforma za razvoj javanskih mobilnih aplikacij
JDK	angl. *Java Development Kit* – paket za razvoj javanskih aplikacij

JDT	angl. *Java Developoment Tools* – orodja za razvoj javanskih aplikacij
JVM	angl. *Java Virtual Machine* – javanska navidezna naprava
MIDP	angl. *Mobile Information Device Profile* – API vmesnik za uporabniški vmesnik, življenjski cikel aplikacije, omrežno povezljivost, multimedijo, shrambo podatkov in brezžično nameščanje programske opreme
OTASL	angl. *Over The Air Software Loading* – brezžično nameščanje programske opreme
PDA	angl. Personal Digital Assistant - dlančnik
PIM	angl. *Personal Information Management* – upravljanje z osebnimi podatki
QEMU	angl. *Quick EMUlator* – emulator, ki temelji na dinamičnem binarnem prevajanju in lahko izvaja programe, prevedene za neko procesorsko platformo na drugi procesorski platoformi
RFID	angl. *Radio Frequency IDentification* – identifikacija preko radijskih valov
SDK	angl. *Software Development Kit* – paket za razvoj programske opreme
SQL	angl. *Structured Query Language* – strukturiran povpraševalni jezik
WEP	angl. *Wired Equvivalent Privacy* – zasebnost kot v žičnem omrežju
WLAN	angl. *Wireless Local Area Network* – brezžično lokalno omrežje
WPA	angl. *WiFi Protected Access* – zaščiten brezžičen dostop
XML	angl. *eXtenisble Markup Language* – razširljiv označevalni jezik

Povzetek

Cilj diplomskega dela je predstavitev primera razvoja mobilne aplikacije na platformi Android. Prvi del diplomskega dela je namenjen predstavitvi orodij in tehnologij, ki jih uporabljamo pri razvoju mobilnih aplikacij. Pri predstavitvi uporabljenih tehnologij smo se osredotočili predvsem na teoretični opis mobilnih tehnologij kot so mobilna omrežja, mobilne naprave, pametni mobilni telefoni, mobilni operacijski sistemi in mobilni operacijski sistem Android. V teoretičnem delu diplomskega dela so na kratko predstavljena tudi vsa orodja razvojnega okolja, pri čemer je večji poudarek na opisu manj poznanih orodij kot so recimo orodja Android SDK.

Drugi del diplomskega dela vsebuje predstavitev razvoja mobilne aplikacije AvtoDroid. Najprej sta opisani metodologija in ideja razvoja aplikacije, nato pa sledi podroben opis posameznih faz razvoja naše aplikacije (analiza in specifikacija zahtev, načrtovanje, implementacija ter testiranje). Na koncu sledi še analiza razvoja aplikacije, kjer so poleg analize razvoja predstavljene tudi možne izboljšave ter načrt nadaljnjega razvoja aplikacije. Rezultat praktičnega dela diplomske naloge je mobilna aplikacija AvtoDroid, ki omogoča številne funkcionalnosti, povezane predvsem s spremljanjem stroškov uporabe avtomobila. Aplikacija se lahko uporablja na vseh mobilnih napravah, ki imajo nameščen operacijski sistem Android.

Ključne besede: mobilne tehnologije, Android, Android aplikacije, AvtoDroid.

Abstract

The aim of the thesis is to present an example of mobile application development on the Android platform. The first part of the thesis is devoted to the presentation of the tools and technologies used in mobile applications development. While describing the technologies we are focused on the theoretical description of mobile technologies such as mobile networks, mobile devices, smart mobile phones, mobile operating systems and mobile operating system Android. In the theoretical part of the thesis we also present different development environment tools, with special emphasis on the description of less known tools such as Android SDK tools.

The second part of the thesis presents the development of the mobile application AvtoDroid. First we describe the methodology and the idea of application development, followed by a detailed description of the various stages of our application development (analysis and requirements specification, design, implementation and testing). In conclusion we analyze application development and we also present a plan for improvement and further development of the application. The result of the thesis is AvtoDroid mobile application that allows a number of features, related to the monitoring car costs. We can use this application on any mobile device which uses operating system Android.

Keywords: mobile technologies, Android, Android applications, AvtoDroid.

1 Uvod

Mobilne tehnologije so v zadnjem času postale pomemben del našega vsakodnevnega življenja. Skoraj vsak izmed uporablja vsaj eno mobilno napravo, pri čemer pa so vse bolj popularni pametni mobilni telefoni in tablični računalniki. Zmogljivosti teh dveh vrst naprav lahko tako po strojni kot tudi programski opremi že primerjamo z zmogljivostmi osebnih računalnikov. Uporaba mobilnih naprav nam skupaj z vse bolj naprednimi brezžičnimi omrežji, ki nam zagotavljajo tudi hiter prenos podatkov preko interneta, omogoča, da lahko v stanju mobilnosti opravljamo tako rekoč vse naloge, ki bi jih drugače lahko opravljali le z uporabo osebnega računalnika doma ali na delovnem mestu.

Trenutno najbolj popularen oziroma razširjen mobilni operacijski sistem Android nam omogoča nameščanje ter uporabo zelo velikega števila aplikacij. Android aplikacije so večinoma razvite z uporabo programskega jezika Java. Tudi sami uporabljamo pametni mobilni telefon, ki ima nameščen ta operacijski sistem. To je bil poleg tega, da nas zanima programiranje v programskem jeziku Java, eden glavnih razlogov za razvoj mobilne aplikacije na platformi Android. Razlog pa je bil tudi ta, da podobne aplikacije ne omogočajo nekaterih funkcionalnosti, ki si jih sami želimo, prav tako pa je uporaba nekaterih funkcionalnosti aplikacij pogosto omogočena samo pri polnih in običajno plačljivih verzijah. Čeprav smo imeli zelo malo predhodnih izkušenj s programiranjem Android aplikacij, smo se vseeno odločili, da bo glavni cilj diplomskega dela prikaz razvoja mobilne aplikacije na platformi Android, ker se bomo ob razvoju seznanili tudi z mobilnimi tehnologijami, katere smo do sedaj poznali le malo ali pa sploh ne.

Seznanili smo se tudi s potekom razvoja programske opreme, ki smo ga do sedaj spoznali le iz teoretičnega vidika. Še pred razvojem aplikacije smo preučili literaturo, ki obravnava operacijski sistem Android ter razvoj Android aplikacij. Nato smo si namestili ustrezna razvoja orodja ter pričeli z razvojem aplikacije po posameznih razvojnih fazah. Rezultat našega dela je delujoča mobilna aplikacija AvtoDroid, ki omogoča številne funkcionalnosti ter deluje na mobilnih napravah, ki imajo nameščen operacijski sistem Android. Aplikacija deluje popolnoma lokalno, saj so vsi podatki shranjeni v lokalni bazi podatkov.

2 Orodja in tehnologije, uporabljene pri razvoju mobilne aplikacije AvtoDroid

2.1 Mobilne tehnologije

V zadnjem času se izraz mobilne tehnologije [3] pojavlja vedno bolj pogosto. Možno ga je zaslediti tako rekoč povsod (v medijih, reklamah, šolah, fakultetah itd.). To kaže na izredno razširjenost mobilnih tehnologij tako v Sloveniji kot tudi drugod po svetu in na pomembno mesto, ki ga imajo v današnjem vsakodnevnem življenju. Po zadnjih podatkih imamo v Sloveniji 106 mobilnih telefonov na 100 prebivalcev *(*evropski trend je sicer nekoliko višji, saj ima povprečna evropska država približno 120 mobilnih telefonov na 100 prebivalcev). Toda večina ljudi izraza mobilne tehnologije verjetno sploh ne razume pravilno, misleč, da je izraz mobilne tehnologije enakovreden izrazu mobilni telefoni. Izraza mobilne tehnologije ne smemo jemati dobesedno, saj tehnologije same po sebi niso mobilne. Mobilni smo uporabniki, ki tehnologije uporabljamo. Z izrazom tehnologije imamo v mislih najrazličnejše mobilne naprave, kot so mobilni telefoni, tablični računalniki itd; te tehnologije pa pravzaprav niso mobilne, ampak so prenosne ter prirejene za mobilno uporabo. To pomeni, da so majhne, lahke in priročne, da delujejo na baterije in so enostavne za uporabo. Z izrazom mobilne tehnologije torej običajno opisujemo tehnologije, ki jih uporabljamo v stanju mobilnosti (recimo ko smo na poti in nimamo dostopa do osebnega računalnika). V grobem jih lahko razdelimo na mobilna oziroma brezžična omrežja, mobilne naprave in mobilne storitve.

Razvoj mobilnih tehnologij je iz leta v leto hitrejši. Vzrok je vsekakor v hudi konkurenčnosti izdelovalcev mobilnih naprav in ponudnikov storitev saj skoraj ne mine dan, da na tržišče ne pride kakšna novost na področju mobilnih tehnologij. Na razvoj mobilnih storitev vplivajo trije glavni dejavniki:

- razvoj novih omrežij in omrežnih tehnologij,
- razvoj novih mobilnih naprav,
- želje ter zahteve trga oziroma uporabnikov.

Ker ti dejavniki velikokrat vplivajo tudi drug na drugega, se na ta način še stopnjujejo hitrost razvoja mobilnih tehnologij. Lahko rečemo, da gre v bistvu za nek razvojni krog, kjer mobilna omrežja vplivajo na razvoj novih mobilnih storitev in novih mobilnih naprav, ki bodo omogočale uporabo teh novih storitev. Nove storitve pa so zahtevane tudi s strani uporabnikov oziroma zaradi potreb trga. Za uporabo novih storitev pa je potrebno razviti tudi nove mobilne naprave in nova mobilna omrežja.

2.1.1 Mobilna omrežja in brezžične komunikacije

Brezžična komunikacija [3, 12] pomeni brezžičen prenos signala od oddajnika do prejemnika. Omogočajo jo brezžična omrežja, ki zagotavljajo fizično mobilnost tako uporabnikom kot tudi mobilnim napravam. Mobilna omrežja za brezžično komunikacijo za svoje delovanje uporabljajo različne tehnologije, ki delujejo na osnovi svetlobe in radijskih valov. Najbolj razširjen sistem za brezžično komunikacijo na osnovi svetlobe so infrardeči (angl. *infrared*) vmesniki, ki za komunikacijo uporabljajo infrardečo svetlobo. Te vmesnike vse bolj izpodrinjajo na fizične motnje manj občutljive tehnologije na osnovi radijskih valov. Na geografskih območjih, ki jih pokriva omrežje z brezžičnim signalom (tako je zgrajena večina današnjih sistemov), se lahko z ustrezno mobilno napravo povežemo v to omrežje. Med brezžična omrežja se ne uvrščajo samo najbolj razširjena mobilna telefonska omrežja, ampak tudi druge povezave kot sta npr. Bluetooth in brezžična lokalna omrežja WLAN.

Razvoj mobilnih telefonskih omrežij opisujemo z generacijami. Poznamo naslednje generacije mobilnih telefonskih omrežij:

- **prva generacija (1G)** – razvit je bil sistem NMT, ki omogoča zelo počasen prenos podatkov (teoretično največ do 380 bitov/s);
- **druga generacija (2G)** – razvit je bil sistem GSM, ki za razliko od NMT omogoča nekoliko hitrejši (vendar za današnje čase zelo počasen) prenos podatkov (teoretično največ do 14400 bit/s);
- **druga in pol generacija (2,5 G)** – razviti sta bili tehnologiji hitrega prenosa podatkov HSCSD, ki teoretično omogoča prenose s hitrostjo največ 43,2 kbit/s, in GPRS, ki teoretično omogoča prenose s hitrostjo največ 115 kbit/s;
- **EDGE (2,75G)** – iz tehnologije GPRS se razvije tehnologija EDGE, ki ponuja hitrejše prenose podatkov (teoretično do največ 473,6 kbit/s);
- **tretja generacija (3G)** – razvita je bila tehnologija UMTS, ki ponuja že dokaj velike hitrosti paketnega prenosa podatkov (teoretično največ do 2 Mbit/s);
- **tretja in pol generacija (3,5G)** – razvita je bila tehnologija HSDPA, ki predstavlja nadgradnjo omrežja UMTS in omogoča še hitrejše prenose podatkov (teoretično do največ 14,4 Mbit/s);
- **HSUPA in HSPA+ (3,75G)** - HSUPA je nadgradnja tehnologije HSDPA in omogoča hitrejši prenosa podatkov od uporabnika v omrežje (teoretično do največ 1,4 Mbit/s). HSPA+ omogoča občutno večje hitrosti prenosa podatkov, in sicer teoretično največ 21,6 Mbit/s v smeri proti uporabniku in največ 5,76 Mbit/s v smeri od uporabnika. Tehnologiji sta že v uporabi, vendar sta zaenkrat še redkost;

- **četrta generacija (4G)** – razvita sta bili tehnologiji LTE oziroma LTE-Advanced, ki pa sta zaenkrat še v fazi testiranja in naj bi teoretično omogočali prenose podatkov s hitrostjo do 100 Mbit/s, pri nepremičnem mobilnem terminalu pa celo do 1 Gbit/s.

Brezžično lokalno omrežje WLAN [3, 34] je računalniško omrežje kratkega dosega, ki deluje na osnovi radijskih valov frekvence 2,4 GHz in 5 GHz. Ti frekvenci se uvrščata v frekvenčno območje, ki je bilo rezervirano za nekomercialno uporabo na področju industrije, znanstvenega raziskovanja in medicine in ga imenujemo tudi ISM. To pomeni, da za WLAN komunikacije ni potrebno pridobiti koncesije za uporabo frekvenčnega spektra. Omrežje sestavljajo dostopne točke, ki so navadno priključene na ožičeno lokalno omrežje (med seboj se lahko povezujejo tudi brezžično). Te dostopne točke pokrivajo določeno področje z brezžičnim signalom, preko katerega se uporabniki lahko povežejo v lokalno omrežje. Domet posamezne dostopne točke je do 30 metrov v notranjosti betonskih zgradb oziroma do 300 metrov na prostem. Domet oziroma pokritost določenega področja lahko zagotovimo s postavitvijo več dostopnih točk, katerih signal se med seboj prekriva. Najbolj pogosto brezžično lokalno omrežje WLAN 802.11g deluje pri hitrosti 54 Mbit/s. Za primerjavo: najpogostejše žično omrežje deluje pri hitrosti 100 Mbit/s, s tem da se čedalje pogosteje uporablja 1 Gbit/s omrežje, v razvoju pa so že omrežja s hitrostjo 10 Gbit/s. Najhitrejše lokalno brezžično omrežje WLAN 802.11n omogoča hitrosti celo do 300 Mbit/s. To močno zmanjša vrzel med brezžično in žično hitrostjo. Brezžična omrežja, ki temeljijo na standardih IEEE 802.11, imenujemo tudi WiFi omrežja. Uporaba je enostavna, saj uporabnik na področju, ki je pokrito z vsaj enim omrežjem WLAN, z vključeno napravo poišče brezžična omrežja, izbere želeno omrežje in se vanj poveže. Uporabnik mora imeti v brezžični napravi vgrajen brezžični modul, ki ga sestavljajo antena, oddajnik in sprejemnik. Brezžičnih omrežij zaenkrat ne moremo zaščititi tako dobro, kot lahko zaščitimo žična omrežja. Varnost v brezžičnih omrežij se zagotavlja:

- z overjanjem uporabnika, ki lahko poteka enostavno z geslom ali pa z uporabo digitalnih certifikatov,
- s šifriranjem samega prenosa podatkov.

Za overjanje uporabnikov se najbolj pogosto uporablja tehnologija WPA, za šifriranje pa tehnologija WEP, ki naj bi omogočala skoraj enako stopnjo varnosti, kot jo imamo pri žičnih komunikacijah.

Infrardeče brezžične povezave [3] za prenos podatkov na kratke razdalje uporabljajo infrardečo svetlobo, ki je uporabljena kot nosilec podatkov in zaradi tega tudi omejena z njenimi lastnostmi. Za uspešno komunikacijo med oddajnikom in sprejemnikom ne sme biti nikakršnih fizičnih ovir, prav tako pa morata biti napravi pravilno usmerjeni ena k drugi. Kakršnakoli fizična ovira predstavlja motnjo v komunikaciji in s tem zniža hitrost prenosa

podatkov, lahko pa celo prekine povezavo. Najbolj razširjen standard za brezžične komunikacije na osnovi infrardeče povezave je IrDA. Naprave z IrDA omogočajo komunikacijo s hitrostmi do 4Mbit/s, pri čemer razdalja med oddajnikom in sprejemnikom ne sme biti večja od dveh metrov. Na večjih razdaljah se hitrost hitro znižuje in kmalu komunikacija sploh ni več mogoča. Infrardeč vmesnik imajo dandanes vgrajene le še redke elektronske naprave, saj se za prenos podatkov na kratke razdalje večinoma uporablja tehnologija Bluetooth.

Tehnologija Bluetooth [3, 23] omogoča brezžično komunikacijo med elektronskimi napravami na krajše razdalje: do 10 metrov v verziji Bluetooth 1.0 pa vse do 60 metrov v verziji Bluetooth 4.0. Komunikacija poteka na osnovi radijskih valov nizke moči, zato ni potrebno, da sta napravi, ki komunicirata, v medsebojnem vidnem polju. Hitrost prenosa podatkov med napravama se giblje od okoli 1Mbit/s v verziji Bluetooth 1.0 do okoli 24 Mbit/s v verziji Bluetooth 3.0. Najnovejša verzija Bluetooth 4.0 omogoča enake hitrosti prenosa podatkov kot prejšnja verzija, vendar prinaša dve pomembni izboljšavi: nižjo porabo energije ter večji domet za brezžično komunikacijo med elektronskima napravama. Tehnologija je posebej primerna za mobilne telefone, prenosne računalnike in tablične računalnike, saj zagotavlja hitro ter enostavno medsebojno povezavo. Zelo so razširjene tudi brezžične slušalke in brezžični sprejemniki GPS, ki jih s pomočjo tehnologije Bluetooth povežemo z mobilnimi napravami. V zadnjem času se ta tehnologija vse pogosteje uporablja na področju avtomobilizma (npr. vmesniki za brezžično telefoniranje, predvajanje glasbe, glasovno upravljanje itd.). Na trgu je vedno več elektronskih naprav, ki omogočajo komunikacijo preko te zelo praktične tehnologije. V prihodnosti naj bi imela vsaka elektronska naprava in tudi druge naprave (recimo zabavna elektronika in gospodinjski aparati) vgrajeno podporo za brezžično komunikacijo Bluetooth, s pomočjo katere se bo lahko povezovala s katerokoli drugo elektronsko napravo.

2.1.2 Mobilne naprave

Mobilne naprave [3, 5] so elektronske naprave, ki jih uporabniki uporabljamo za brezžično komunikacijo z drugimi uporabniki, povezovanje preko mobilnih omrežij do storitev na svetovnem spletu in za nameščanje ter uporabo mobilnih aplikacij. Mobilne naprave so se z razvojem spreminjale in iz običajnega telefona postale večpredstavnostne komunikacijske naprave. Mobilne naprave ne omogočajo samo komuniciranja in povezovanja, temveč tudi uporabo storitev in aplikacij. V času 1. generacije mobilnih omrežij so bili telefoni veliki, težki in so omogočali le telefoniranje. Z razvojem novih elektronskih elementov, komponent in tehnologij pa so tudi mobilni telefoni postali bolj priročni, manjši in komunikacijsko veliko bolj zmogljivi. Omogočajo več različnih načinov komunikacije, boljšo kakovost storitev in hitrejše prenose podatkov. Za brezžično komunikacijo se danes ne uporabljajo samo mobilni

telefoni, ampak so trenutno v uporabi zelo različne mobilne naprave (slika 1), ki jih glede na obliko, vrsto ter način uporabe razdelimo na tablične računalnike (angl. *tablet computer*), dlančnike (angl. *PDA*), pametne mobilne telefone (angl. *smartphone*), običajne mobilne telefone (angl. *mobile phone*) in prenosne računalnike (angl. *laptop computer*). Prenosni računalniki niso tipični primer mobilne naprave, saj gre tu v bistvu za primer osebnega računalnika, kjer pa se ne odražajo posebnosti mobilnih aplikacij, ki se izvajajo na ostalih mobilnih napravah. Ugotovimo torej lahko, da predstavlja pri ostalih zvrsteh mobilnih naprav njihova velikost omejitev, ki pogojuje vse ostale omejitve mobilnih naprav in posledično tudi omejitve mobilnih aplikacij. Trenutno so med najbolj priljubljenimi mobilnimi napravami pametni mobilni telefoni, ki postajo vse bolj elegantni in jih sestavlja napredna strojna oprema kot so npr. GPS, merilec pospeškov, zaslon na dotik itd. Tudi programska oprema pametnih mobilnih telefonov je vse bolj napredna, kar kaže dejstvo, da imajo vsi pametni mobilni telefoni nameščenega enega od številnih mobilnih operacijskih sistemov, kar nam omogoča enostavno nameščanje poljubne programske opreme. Tudi rezultat našega dela, aplikacijo AvtoDroid, smo razvili za uporabo na pametnih mobilnih telefonih (lahko tudi na tabličnih računalnikih), ki imajo nameščen operacijski sistem Android. Pametne mobilne telefone bomo podrobno predstavili v poglavju 2.1.3.

Slika 1: Primeri mobilnih naprav.

2.1.3 Pametni mobilni telefoni

Pametni mobilni telefon [7, 29] je mobilna naprava, katere zmogljivosti lahko že primerjamo z zmogljivostmi osebnega računalnika tako po strojni opremi (opremljeni so npr. z zmogljivim dvojedrnim procesorjem, imajo nekaj GB delovnega pomnilnika itd.) kot po programski opremi (naložen imajo mobilni operacijski sistem, ki uporabniku dovoli namestitev in uporabo poljubne programske opreme). Sprva je bil pametni mobilni telefon kombinacija navadnega mobilnega telefona in dlančnika, medtem ko danes pametni mobilni telefon združuje več naprav (npr. digitalni fotoaparat, predvajalnik večpredstavnostnih vsebin, video kamero, modul GPS itd.) v eni. Njihove funkcionalnosti so združene in podprte z najsodobnejšimi tehnologijami. Sodobni pametni mobilni telefoni imajo zaslon na dotik z visoko ločljivostjo, spletni brskalnik za dostop in pravilno prikazovanje standardnih spletnih

strani (in ne samo spletnih mest, optimiziranih za mobilne naprave), odjemalec za elektronsko pošto ter možnost hitrega prenosa podatkov preko vseh sodobnih, v poglavju 2.1.1 omenjenih mobilnih omrežij. Čeprav se v zadnjem času vse bolj pogosto uporablja izraz pametni mobilni telefon, pa pravzaprav ne obstaja nobena povsem točna definicija, kaj pametni mobilni telefon je. Tudi določanje meje med navadnim ter pametnim mobilnim telefonom ni jasno, naj pa bi pametni mobilni telefon v primerjavi z navadnim mobilnim telefonom imel naslednje lastnosti in funkcionalnosti:

- **mobilni operacijski sistem** (angl. *mobile operating system*) – pametni mobilni telefon temelji na operacijskem sistemu, ki omogoča poljubno nameščanje in uporabo programske opreme oziroma aplikacij. Obstaja veliko različnih mobilnih operacijskih sistemov, katere bomo podrobno predstavili v poglavju 2.1.4;
- **zaslon na dotik** (angl. *touchscreen*) – zaslon na dotik je čedalje bolj priljubljen in je del skoraj vsakega pametnega mobilnega telefona. Diagonala zaslona pametnega telefona je običajno med 3,4 in 4,3 palca (angl. *inch*), pri tem pa velja poudariti, da je bistvenega pomena čim višja ločljivost, katere rezultat je jasna in ostra slika. Vse bolj pogosto se pojavljalo tudi večdotični (angl. *multitouch*) zasloni, ki lahko zaznavajo dotike na več mestih hkrati, kar omogoča izvajanje tudi zahtevnejših ukazov, kot so vrtenje, približevanje in obračanje;
- **aplikacije** (angl. *apps*) – vsi mobilni telefoni, tudi najosnovnejši modeli, vključujejo nekaj najbolj osnovne programske opreme (adresar, koledar, budilka itd.). Pametni mobilni telefoni poleg te programske opreme omogočajo uporabo aplikacij, ki jih uporabljamo na osebnem računalniku (npr. ustvarjanje in pregledovanje različnih vrst dokumentov, pregledovanje elektronske pošte, brskanje po svetovnem spletu, urejanje fotografij, poslušanje glasbe itd.). V zadnjem času se v mobilnih aplikacijah pojavljajo tudi napredne storitve, kot so pretvarjanje govora v besedilo (angl. *speech to text*), pretvarjanje besedila v govor (angl. *text to speech*), prepoznava obraza in glasu, mobilna internetna telefonija in televizija, identifikacija uporabnika z vgrajenim RFID, ki omogoča brezžično identifikacijo mobilne naprave itd. Pri razvoju mobilnih aplikacij pa moramo imeti v mislih vedno tudi omejitve mobilnih naprav (majhen zaslon, omejen delovni pomnilnik itd.);
- **dostop do spleta** (angl. *web access*) – večina pametnih mobilnih telefonov omogoča dostop do svetovnega spleta pri velikih hitrostih z uporabo 3G in 4G mobilnih omrežij kot tudi možnosti povezovanja preko WiFi;
- **QWERTY tipkovnica** (angl. *QWERTY keyboard*) – pametni mobilni telefon naj bi vseboval QWERTY tipkovnico. To pomeni, da je položaj tipk določen na enak način, kot je določen na tipkovnici pri osebnem računalniku. Pri tem ni nujno, da je

tipkovnica del strojne opreme (fizične tipke, na katere tipkamo) ampak je lahko tudi del programske opreme (prikaz tipkovnice na zaslonu na dotik);

- **sporočanje** (angl. *messaging*) – z vsemi mobilni telefoni lahko prejemamo in pošiljamo kratka besedilna sporočila. S pametnim mobilnim telefonom lahko dostopamo tudi do elektronske pošte, uporabljamo sinhronizacijo z osebnim računalnikom ter dostopamo do nekaterih priljubljenih storitev za neposredno sporočanje (npr. Messenger, Facebook, Gtalk itd.).

Zgodovina pametnih mobilnih telefonov

Začetek razvoja pametnih mobilnih telefonov [16, 25, 29] sega v daljno leto 1992, ko je podjetje IBM predstavilo prvi pametni mobilni telefon imenovan Simon (slika 2), ki pa ga glede na opisane lastnosti in funkcionalnosti pametnega mobilnega telefona težko uvrstimo med prave pametne mobilne telefone. Združeval je funkcije mobilnega telefona, pozivnika, dlančnika in faksa. To je bil prvi mobilni telefon nasploh, ki je poleg klicanja ter pošiljanja in sprejemanja kratkih sporočil ponujal tudi koledar, svetovni čas, beležnico, računalo, dostop do elektronske pošte itd. Telefon ni imel fizične tipkovnice, ampak se je uporabljal ekran na dotik (kar je bila redkost za tiste čase), kjer je bila na voljo navidezna QWERTY ali navadna številska tipkovnica. Kupiti ga je bilo mogoče izključno v ZDA.

Slika 2: Pametni mobilni telefon Simon.

V poznih 90-ih letih je podjetje Nokia razvila vrsto pametnih mobilnih telefonov, namenjenih poslovnim uporabnikom. Prvi pametni mobilni telefon, Nokia 9000, je bil križanec med mobilnim telefonom in dlančnikom, kateremu pa je hitro sledil razvoj še boljših modelov Nokia 9210, Nokia 9300 in Nokia 9500. Tu velja omeniti, da je bila Nokia 9210 eden prvih mobilnih telefonov, ki je uporabljal odprtokodni operacijski sistem in je imel na voljo barvni zaslon.

Leto 1997 je bilo leto rojstva prvega mobilnega telefona, ki se je dejansko imenoval pametni mobilni telefon. Šlo je za koncept pametnega mobilnega telefona Penelope, ki ga je razvilo podjetje Ericsson, izdelanih pa jih je bilo samo okoli 200. Prav razvoj te naprave je nekaj let kasneje utrl pot za razvoj pametnega mobilnega telefona Ericsson R380, ki je bil prvi komercialno dostopen pametni mobilni telefon z operacijskim sistemom Symbian.

Ericsson R380 je bil razvit leta 2000 in je bil prvi pametni mobilni telefon, ki se je tudi tržil pod imenom pametni mobilni telefon. Združeval je funkcije mobilnega telefona in dlančnika, imel pa je nameščen tudi mobilni operacijski sistem. Na voljo je imel zaslon na dotik, kjer je bila prikazana navidezna tipkovnica, prav tako pa je imel na voljo fizično številčno tipkovnico, ki se je uporabljala predvsem pri klicanju.

Leta 2001 je Microsoft predstavil pametni mobilni telefon, ki se je imenoval Windows Powered Smartphone 2002. Naprava ni dosegla velikega uspeha, saj ni imela zaslona na dotik, prav tako pa je imela zelo nizko ločljivost zaslona.

Leta 2002 je bil predstavljen Palm Treo, ki ga je sicer razvilo podjetje Handspring (podjetje je bilo nato kupljeno s strani podjetja Palm). Na voljo je imel zaslon na dotik in fizično QWERTY tipkovnico, na njem pa je bil nameščen operacijski sistem Palm. Istega leta je bil razvit tudi pametni mobilni telefon Sony Ericsson P800, ki je imel za tisti čas nove funkcionalnosti, kot so npr. MP3 predvajalnik in barvni zaslon na dotik. To leto pa je bil razvit tudi prvi BlackBerry telefon podjetja RIM.

Leta 2007 je podjetje Apple predstavilo njihov prvi pametni telefon, imenovan iPhone, ki ima nameščen operacijski sistem iOS. Celotna navigacija po operacijskem sistemu telefona (vključno s klicanjem) poteka s pomočjo na dotik občutljivega zaslona, saj telefon nima fizične tipkovnice. Do danes je bilo razvitih več novih modelov iPhone, kot so iPhone 3G, iPhone 3GS, iPhone 4 in najnovejši (predstavljen konec leta 2011) iPhone 4GS (slika 3). Za iOS je bilo do sedaj razvitih že preko pol milijona različnih (brezplačnih in plačljivih) aplikacij, ki jih lahko prenesemo s pomočjo spletne aplikacije App Store.

Leta 2008 je bila razvita mobilna platforma Android, ki je odprtokodni (angl. *open source*) mobilni operacijski sistem, podprt in razvit s strani podjetij Google, HTC, Intel in še nekaterih drugih vplivnih podjetij. Združenje teh podjetij se imenuje *Open Handset Alliance* (v nadaljevanju OHA). HTC Dream je bil prvi pametni mobilni telefon, ki je imel nameščen operacijski sistem Android. Android je hitro postal najbolj priljubljen mobilni operacijski sistem, za katerega je na voljo že preko pol milijona različnih (brezplačnih in plačljivih) aplikacij, ki jih lahko prenesemo s pomočjo spletne aplikacije Android Market.

Slika 3: Pametni mobilni telefon iPhone 4S.

Leta 2009 so tudi nekatera druga podjetja (že prej sta to storila Apple in OHA) odprla lastno prodajalno aplikacij (angl. *app stores*), kot so npr. Ovi Store (Nokia), Windows Marketplace (Windows Mobile in Windows Phone) in App World (BlackBerry), kjer za uporabnike ponujajo tako plačljive kot brezplačne aplikacije.

Leta 2010 je podjetje Google izdalo svoj prvi pametni mobilni telefon, imenovan Nexus One (izdeluje ga sicer podjetje HTC), ki ima nameščen operacijski sistem Android. Poganja ga dvojedrni procesor takta 1,2 GHz in ima 1GB delovnega pomnilnika. Istega leta je podjetje Microsoft predstavilo nov operacijski sistem, imenovan Windows Phone, ki je naslednik operacijskega sistema Windows Mobile.

Leta 2011 je podjetje HTC predstavilo pametni mobilni telefon HTC Evo 3D (slika 4), ki lahko na ekranu prikazuje 3D učinke, ne da bi za to potrebovali posebna očala.

V prihodnosti lahko pričakujemo, da bodo Apple, HTC, Samsung ter še nekateri drugi večji proizvajalci pametnih mobilnih telefonov ustvarili še bolj zmogljive in napredne pametne mobilne telefone, ki bodo vključevali še neznane oziroma zaenkrat še nepredstavljive funkcionalnosti. Prav tako se bodo verjetno v to smer razvijali tudi mobilni operacijski sistemi ter mobilne aplikacije.

Slika 4: Pametni mobilni telefon HTC Evo 3D.

2.1.4 Mobilni operacijski sistemi

Najpomembnejša programska oprema v vsaki sodobni mobilni napravi je mobilni operacijski sistem [8, 27]. Mobilni operacijski sistem je operacijski sistem, ki upravlja s sredstvi strojne in programske opreme ter nadzoruje pametne mobilne telefone, tablične računalnike, dlančnike in druge mobilne naprave. Sodobni mobilni operacijski sistemi združujejo funkcije operacijskega sistema osebnega računalnika s funkcijami zaslona na dotik, sodobnih brezžičnih povezav (npr. WiFi, Bluetooth itd.), GPS navigacije, fotoaparata, video kamere, prepoznavanja govora, predvajalnika glasbe itd., prav tako pa nam omogočajo uporabo širokega spektra različnih mobilnih aplikacij.

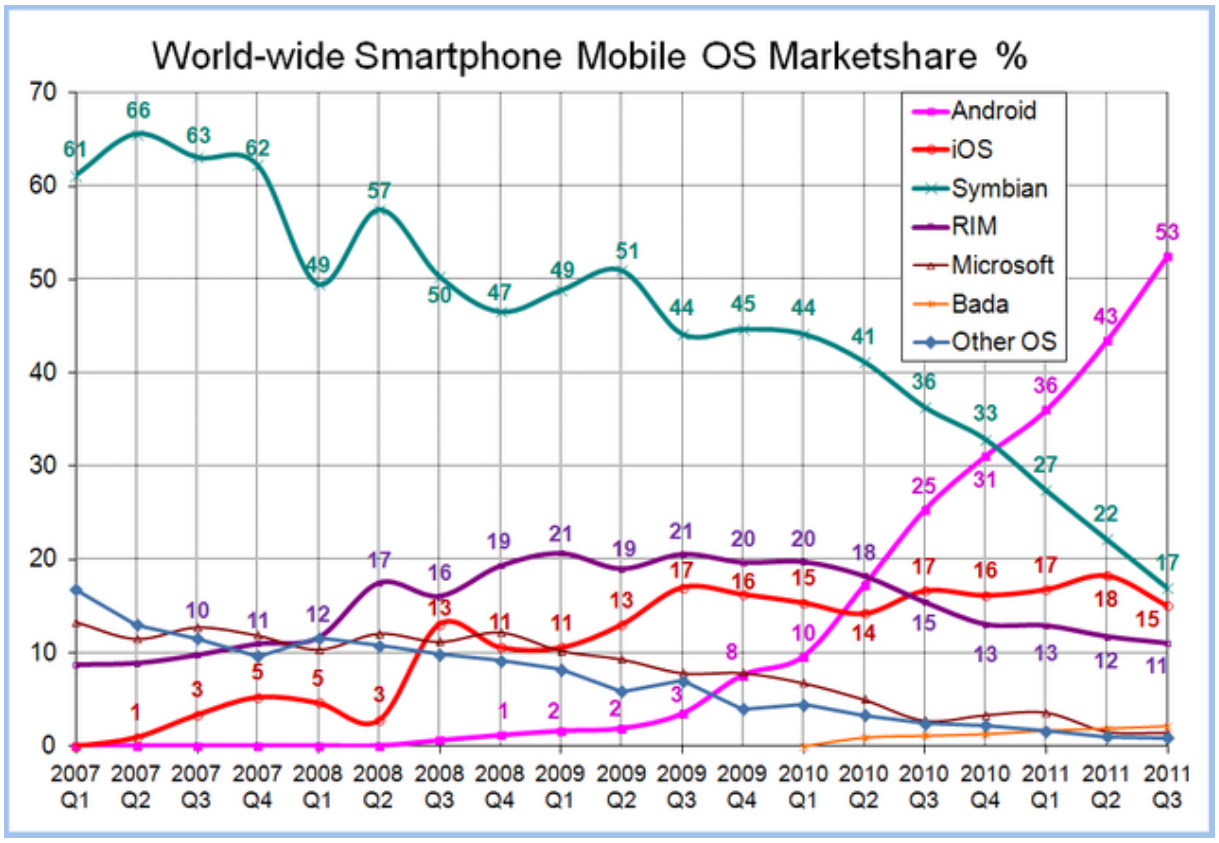
Osebni računalniki imajo lahko nameščene različne operacijske sisteme (npr. Windows, Linux itd.) oziroma različne verzije nekega operacijskega sistema (npr. Windows XP, Windows 7 itd.). Mobilne naprave pa običajno ne omogočajo nameščanja različnih mobilnih operacijskih sistemov, saj proizvajalci mobilne naprave določijo, kateri mobilni operacijski sistem bo nameščen na neki napravi. Kljub temu pa običajno omogočajo nameščanje različnih verzij nekega določenega mobilnega operacijskega sistema. To pomeni, da bodo na pametnem mobilnem telefonu iPhone lahko nameščene samo različne verzije mobilnega operacijskega sistema iOS, medtem ko bodo na Android pametnih mobilnih telefonih lahko nameščene

samo različne verzije mobilnega operacijskega sistema Android. Seveda pa obstajajo tudi izjeme, vendar je teh malo.

Obstajajo tri glavne kategorije mobilnih operacijskih sistemov, ki jih najdemo na mobilnih napravah različnih proizvajalcev:

- **proizvajalčev lastni operacijski sistem** (angl. *manufacturer-built proprietary operating systems)* – nekateri proizvajalci mobilnih naprav uporabljajo lasten operacijski sistem za mobilne naprave, ki jih proizvajajo. Primer je podjetje Apple z mobilnim operacijskim sistemom iOS, ki je nameščen na napravah iPod Touch, iPhone in iPad. Naslednji primer sta podjetji RIM, ki uporablja lasten operacijski sistem BlackBerry za vse BlackBerry mobilne naprave, in podjetje HP, ki uporablja lasten operacijski sistem Palm za svoje mobilne naprave. Značilnost teh operacijskih sistemov je, da imajo običajno zelo dosleden videz v vseh mobilnih napravah, ki jih poganjajo;
- **operacijskih sistemi zunanjih dobaviteljev** (angl. *third party proprietary operating systems)* – tu gre za podjetja, ki ne proizvajajo lastnih mobilnih naprav, ampak ponujajo samo lasten mobilni operacijski sistem, ki ga potem na svoje mobilne naprave nameščajo drugi proizvajalci mobilnih naprav. Primer sta operacijska sistema Windows Mobile in Windows Phone, ki ju razvija podjetje Microsoft, uporabljata pa se na mobilnih napravah znamk HTC, Samsung, LG itd. Tudi ti operacijski sistem imajo običajno zelo dosleden videz na vseh mobilnih napravah, ne glede na njihovo znamko;
- **prosti in odprtokodni operacijski sistemi** (angl. *free and open source operating systems*) - odprtokodni operacijski sistemi so izdelani s strani podjetja, skupine podjetij ali skupnosti razvijalcev in so brezplačno dostopni vsakomur, lahko pa jih tudi poljubno spreminjamo in nadgrajujemo. Primeri teh operacijskih sistemov so Symbian, MeeGo in najpomembnejši, Android. Proizvajalci mobilnih naprav, ki uporabljajo te mobilne operacijske sisteme, običajno nekoliko prilagodijo sam izgled operacijskega sistema ter dodajo nove funkcionalnosti in vmesnike, da kar najbolj ustreza njihovim napravam. Za primer lahko vzamemo podjetje HTC, ki je v prizadevanju za izboljšanje uporabniške izkušnje ustvarilo nov grafično izboljšan vmesnik, imenovan HTC Sense. Poleg tega nam ti operacijski sistemi ponujajo veliko več prilagajanja samega operacijskega sistema. V primerjavi z ostalimi operacijskimi sistemi lahko z namestitvijo ustrezne programske opreme spreminjamo videz in občutenje (angl. *look and feel*) ter obnašanje operacijskega sistema. S tem še izboljšamo uporabniško izkušnjo.

Leta 2006, ko še ni bilo mobilnih operacijskih sistemov kot so Android, iOS, Windows Phone in Bada, je bilo prodanih zgolj 64 milijonov pametnih mobilnih telefonov letno. Danes je vsako leto prodanih skoraj desetkrat več pametnih mobilnih telefonov, pri čemer imajo najpogosteje nameščene operacijske sisteme Android, Symbian, iOS, BlackBerry, Windows Phone in Bada. Na sliki 5 je prikazan tržni delež mobilnih operacijskih sistemov od začetka leta 2007 do konca leta 2011. Opazimo lahko, da je trenutno najbolj priljubljen operacijski sitem Android, ki ima že preko 50% tržni delež, medtem ko je v teh letih največji padec doživel operacijski sitem Symbian (iz preko 60% tržnega deleža na manj kot 20% tržnega deleža). Potrebno pa je upoštevati, da so tu zajeti samo tržni deleži operacijskih sistemov, ki tečejo na pametnih mobilnih telefonih, medtem ko druge mobilne naprave tu niso zajete.



Slika 5: Tržni deleži najpomembnejših mobilnih operacijskih sistemov.

V nadaljevanju bomo na kratko opisali nekatere najpomembnejše in najpogostejše mobilne operacijske sisteme, Symbian, iOS, BlackBerry, Windows (Mobile in Phone) ter Bada, medtem ko bomo operacijski sistem Android podrobno predstavili v poglavju 2.1.5.

Mobilni operacijski sistem iOS

Operacijski sistem iOS [26] (pred junijem 2010 se je imenoval iPhone OS) je lastniški (ni odprtokoden) mobilni operacijski sistem podjetja Apple. Prvotno je bil razvit za pametni mobilni telefon iPhone, kasneje pa je razširil svojo vlogo tudi na druge naprave, kot so iPod, iPad itd., vendar podjetje Apple ne dovoli nameščanja operacijskega sistema iOS na mobilne naprave drugih proizvajalcev. Njihova spletna prodajalna aplikacij App Store vsebuje že več kot pol milijona različnih (brezplačnih in plačljivih) iOS aplikacij, katerih skupno število prenosov je že preseglo 24 milijard. Operacijski sistem je bil prvič prikazan na mobilni napravi iPhone na Macworld konferenci (angl. *Macworld Conference & Expo*) januarja 2007, medtem ko je bil uradno izdan junija istega leta. Na začetku je podpiral nameščanje in uporabo samo tistih aplikacij, ki so bile izdelane pri njihovem podjetju, medtem ko ostale aplikacije niso bile podprte (zunanji razvijalci torej niso imeli možnosti razvoja iOS aplikacij). Tedanji direktor podjetja Apple, Steve Jobs, je to odločitev utemeljil z dejstvom, da lahko razvijalci razvijejo spletne aplikacije, ki se bodo obnašale kot prave aplikacije za iPhone. Oktobra 2007 je podjetje Apple izdalo obvestilo, da razvijajo SDK, ki so ga zunanji razvijalci iOS aplikacij lahko začeli uporabljati že naslednje leto. Junija 2010 je podjetje Apple spremenilo uradno ime operacijskega sistema iPhone OS v iOS. Da bi se izognili tožbi podjetja Cisco Systems, ki je to ime že prej registriralo za njihov operacijski sistem, je podjetje Apple moralo odkupiti licenčne pravice za to ime. Najnovejša različica operacijskega sistema je iOS 5 (slika 6).

Slika 6: Izgled mobilnega operacijskega sistema iOS 5.

Mobilni operacijski sistem Symbian

Symbian [30, 31] je mobilni operacijski sistem namenjen pametnim mobilnim telefonom, katerega trenutno vzdržuje podjetje Accenture. Najnovejša različica Symbian^3 je uradno izšla konec leta 2010 in je bila prvič nameščena na pametni mobilni telefon Nokia N8. Maja 2011 izšla nova posodobitev operacijskega sistema Symbian^3, imenovana Symbian Anna, avgusta pa še Symbian Belle (slika 7). Glavna prednost operacijskega sistema Symbian je njegova arhitektura, ki je načrtovana za prenosne naprave z majhnimi sistemskimi viri in točnimi zahtevami glede robustnosti. Programersko okolje je dogodkovno usmerjeno, zato se lahko procesor naprave izklaplja, ko ni v neposredni uporabi in s tem zmanjša porabo energije. Mobilni operacijski sistem Symbian je bil prvotno razvit v podjetju Symbian Ltd. Je neposredni naslednik operacijskega sistema EPOC (podjetja Psion) in deluje le na procesorjih ARM. Podjetje Nokia, proizvajalec pametnih mobilnih telefonov, ki imajo nameščen operacijski sistem Symbian, je decembra 2008 kupilo podjetje Symbian Ltd. in od takrat naprej se je redno nadgrajevalo in posodabljalo ta operacijski sitem. V začetku februarja 2010 je Symbian za pametne mobilne telefone postal povsem odprtokoden, saj je pridobil odprto licenco EPL. Februarja 2011 sta podjetji Microsoft in Nokia oznanila partnerstvo, s katerim bo Nokia ponujala pametne mobilne telefone, katere bo poganjal operacijski sistem Windows Phone, Ovi Store pa bo preimenovan v Windows Marketplace. Symbian bo glede na obete iz februarja 2011, v naslednjih dveh letih popolnoma nadomestil Microsoftov Windows Phone, vendar naj bi podjetje Accenture, ki trenutno vzdržuje operacijski sistem Symbian, obstoječim uporabnikom nudilo posodobitve in razvoj nove programske opreme vse do leta 2016.

Slika 7: Izgled mobilnega operacijskega sistema Symbian Belle.

Mobilni operacijski sistem BlackBerry

BlackBerry [22] je lastniški (ni odprtokoden) mobilni operacijski sistem, ki ga je razvilo podjetje RIM za njihovo linijo BlackBerry pametnih mobilnih naprav. Operacijski sistem omogoča večopravilnost in podpira nekatere specializirane vhodne naprave, ki jih je podjetje RIM prilagodilo za uporabo na njihovih napravah, kot so npr. drsnik (angl. *trackwheel*), sledilna kroglica (angl. *trackball*), drsna ploščica (angl. *trackpad*) in zaslon na dotik. Platforma BlackBerry je morda najbolj znana po podpori za delo z elektronsko pošto, saj s pomočjo Java MIDP 1.0 in v zadnjem času MIDP 2.0, omogoča popolno brezžično aktiviranje in sinhronizacijo s programi kot so npr. Microsoft Exchange, Lotus Domino itd. ter koledarjem, beležko in stiki. Predvsem med mlajšo generacijo je BlackBerry priljubljen zaradi programske opreme BBM, ki omogoča pošiljanje in sprejemanje šifriranih sporočil, snemanje glasovnih beležk, skeniranje črtnih kod itd., obenem pa omogoča tudi visoko stopnjo varnosti. Posodobitve operacijskega sistema so na voljo samodejno preko brezžičnih povezav, ki podpirajo BlackBerry OTASL storitev. Zunanji razvijalci lahko razvijajo BlackBerry aplikacije s pomočjo BlackBerry API razredov, čeprav je potrebno aplikacije, ki uporabljajo določene funkcionalnosti, digitalno podpisati. Trenutno preko spletne aplikacije App World ponujajo že preko 60.000 različnih aplikacij. Podjetje RIM je nedavno objavilo novico, da bo na operacijskem sistemu BlackBerry možno nameščati in uporabljati tudi Android aplikacije, saj se zavedajo moči Androida na trgu mobilnih operacijskih sistemov. S tem želijo preprečiti še nadaljnjo upadanje prodaje njihovih mobilnih naprav. Najnovejša različica operacijskega sistema je BlackBerry 7 (slika 8).

Slika 8: Izgled mobilnega operacijskega sistema BlackBerry 7.

Mobilna operacijska sistema Windows Mobile in Windows Phone

Mobilni operacijski sistem Windows Mobile [32] je razvilo podjetje Microsoft. Namenjen je pametnih mobilnim telefonom in t.i. žepnim računalnikom (angl. *Pocket PC*). Vsebuje zbirko osnovnih aplikacij, ki so razvite z Microsoft Windows API, ter ima podobne lastnosti in videz kot različice operacijskih sistemov Windows za osebne računalnike. Temelji na Windows CE jedru ter omogoča večopravilnost, namestitev aplikacij, ki uporabljajo `.cab` datoteke, poln dostop do datotečnega sistema, možnost zamenjave celotnega uporabniškega vmesnika z drugim, popolno sinhronizacijo z orodji Microsoft Office itd. Največja slabost je uporabniški vmesnik, ki ni dovolj optimiziran za upravljanje naprave preko zaslona na dotik s prsti, ampak je optimiziran za upravljanje s pisalom (angl. *stylus*). To je bil tudi eden glavnih razlogov za ukinitev razvoja tega operacijskega sistema. Zanimivo je, da je bil HTC Sense, ki je sedaj zelo popularen na operacijskem sistem Android, prvotno razvit prav za Windows Mobile, ki za razliko od operacijskega sistema Android ni odprtokoden. Zadnja izdana verzija je Windows Mobile 6.5.5. Leta 2010 je podjetje Microsoft predstavilo novo generacijo mobilnega operacijskega sistema, imenovanega Windows Phone [33], ki vključuje popolnoma prenovljen vmesnik Metro. S tem je omogočena popolna integracijo z storitvami kot so Zune, Xbox Live, Bing itd. ter tudi drugimi storitvami (ki niso izdelane s strani podjetja Microsoft), kot je npr. Facebook. Za razliko od predhodnika se ta operacijski sistem v prvi vrsti usmerja predvsem na potrošniški trg in ne toliko na podjetniški trg. Kljub temu, da je operacijski sistem Microsoft Phone naslednik operacijskega sistema Microsoft Mobile, pa uporabnikom Microsoft Mobile nadgradnja na Windows Phone ne bo na voljo, prav tako pa na novem operacijskem sistemu ne bo možno uporabljati aplikacij, ki so bile narejene za Windows Mobile. Microsoft sicer zunanjim razvijalcem aplikacij za operacijska sistema Windows Mobile in Windows Phone ne postavlja nobenih omejitev pri razvoju. Trenutno Windows Phone Marketplace ponuja več kot 70.000 različnih aplikacij. Operacijski sistem Windows Phone lahko najdem na mobilnih napravah proizvajalcev HTC, Samsung, Dell, LG, leta 2011 pa je postal tudi glavni operacijski sistem mobilnih naprav proizvajalca Nokia. Trenutna verzija se imenuje Windows Phone 7.5 Mango (slika 9), sicer pa je že najavljen prihod nove verzije, imenovane Windows Phone 8.

Mobilni operacijski sistem Bada

Mobilni operacijski sistem Bada [21] je razvilo podjetje Samsung in je v prvi vrsti namenjen pametnih mobilnim telefonom in tabličnim računalnikom. Ime *bada* izhaja iz istoimenske korejske besede, ki pomeni ocean. Čeprav operacijski sistem zaenkrat še ni odprtokoden, pa Samsung zaradi želje po še večjem tržnem deležu kmalu načrtuje objavo platforme pod odprtokodno licenco. Bada je bil prvič predstavljen na Mobilnem svetovnem kongresu (angl. *Mobile World Congress*) februarja 2010 v Barceloni na pametnem mobilnem telefonu

Slika 9: Izgled mobilnega operacijskega sistema Windows Phone 7.5 Mango.

Samsung Wave S8500, katerih je bilo v samo prvih štirih tednih po začetku prodaje prodanih več kot milijon po vsem svetu. Maja istega leta je podjetje Samsung izdalo beta verzijo SDK, s čimer je pritegnila veliko razvijalcev mobilnih aplikacij. Prav tako je podjetje organiziralo tekmovanje Bada Developer Challenge z nagradnim skladom skoraj tri milijone dolarjev. Avgusta 2010 je bila nato izdana SDK 1.0 verzija, avgusta 2011 pa SDK 2.0 verzija, ki ponuja številne izboljšave glede na prejšnjo verzijo. Od leta 2009 je na voljo tudi spletna mobilna aplikacija Samsung Apps, kjer uporabniki lahko dostopajo do več kot 13.000 različnih aplikacij. Trenutna verzija operacijskega sistema je Bada 2.0 (slika 10).

Slika 10: Izgled mobilnega operacijskega sistema Bada 2.0.

2.1.5 Mobilni operacijski sistem Android

Mobilni operacijski sistem Android [11, 20] je odprtokodni operacijski sistem, ki temelji na jedru operacijskega sistema Linux. Ime izhaja iz angleške besede *android*, kar pomeni robot, ki se obnaša in izgleda kot človek (logotip operacijskega sistema Android lahko vidimo na sliki 11). Android je trenutno najbolj razširjen mobilni operacijski sistem, saj je njegov tržni delež med vsemi mobilnimi operacijskimi sistemi že preko 50%. Najdemo ga nameščenega na mobilnih napravah proizvajalcev HTC, LG, Samsung, Sony Ericsson, Motorola itd., pri čemer proizvajalci običajno prilagodijo sam izgled operacijskega sistema ter dodajo nove funkcionalnosti in vmesnike, da kar najbolj ustreza njihovim napravam.

Slika 11: Logotip mobilnega operacijskega sistema Android.

Android je zasnovan tako, da lahko deluje na vseh vrstah mobilnih naprav, ne glede na njihovo velikost zaslona, resolucijo, zmogljivost procesorja itd., njegovo jedro pa je prenosljivo. Ločuje strojno opremo od programske opreme mobilne naprave, kar omogoča, da neko aplikacijo lahko poganja veliko število različnih mobilnih naprav, kar je prednost tako za razvijalce kot potrošnike.

Glavne prednosti mobilnega operacijskega sistema Android v primerjavi z nekaterimi drugimi mobilnimi operacijskimi sistemi so [17]:

- je brezplačen,
- je odprtokoden, kar razvijalcem omogoča cenejše in lažje razvijanje programov. Prednost tu občutijo tudi uporabniki, saj so programi za ta operacijski sistem večinoma zastonjski,

- omogoča cenejše in lažje razvijanje pametnih telefonov (proizvajalcem ni potrebno več razvijati operacijskih sistemov, lahko pa razvijajo posamezne komponente sistema),
- je enostaven, hitro odziven in omogoča večopravilnost,
- ima zelo enostaven in pregleden uporabniški vmesnik,
- se samodejno sinhronizira s storitvami Googla,
- omogoča hitro in enostavno nameščanje aplikacij preko Android Market,
- omogoča enostavno posodobitev tako samega operacijskega sistema kot tudi nameščenih aplikacij,
- itd.

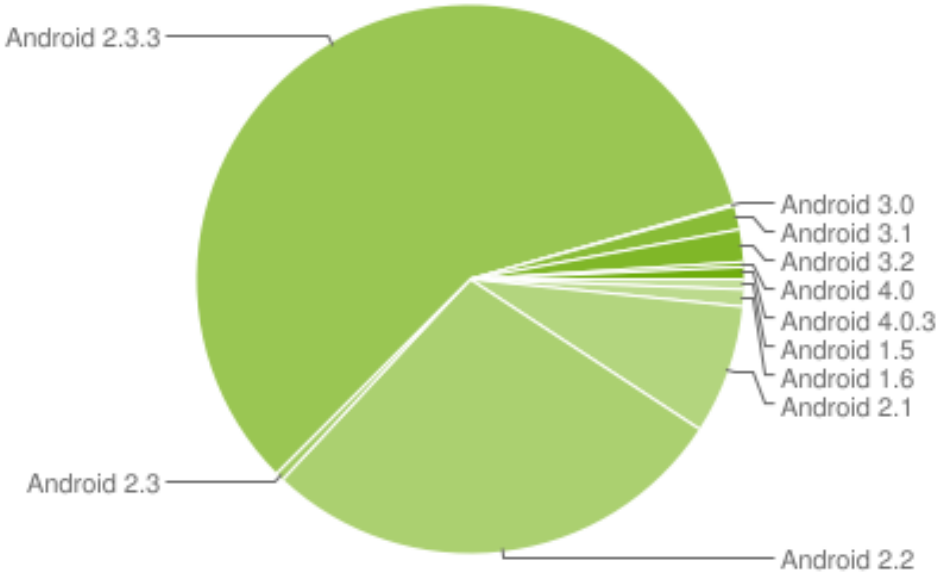
Android lahko okvirno predstavimo kot kombinacijo treh komponent [4] (njegovo podrobnejšo sestavo bomo predstavili v nadaljevanju poglavja):

- je brezplačen in odprtokodni mobilni operacijski sistem za mobilne naprave,
- je odprtokodna razvojna platforma za ustvarjanje mobilnih aplikacij,
- je mobilna naprava (običajno pametni mobilni telefon), ki ima nameščen operacijski sistem Android in Android aplikacije.

Z razvojem mobilnega operacijskega sistema Android [1, 20] je leta 2003 začelo podjetje Android Inc., ki ga je nato leta 2005 kupilo podjetje Google. Prva verzija Androida je bila predstavljena leta 2007 s strani konzorcija OHA, ki ga trenutno sestavlja 84 podjetij, med katerimi imajo glavno vlogo Google, HTC, Sony, Dell itd. Njihov glavni cilj je pospeševanje inovacij na področju mobilne telefonije in s tem potrošnikom omogočiti vsebinsko bogatejšo, boljšo in cenejšo uporabniško mobilno izkušnjo. Prva beta verzija operacijskega sistema Android je bila izdana novembra 2007, istega leta pa je bil najavljen tudi prvi SDK za Android. Prva komercialna verzija Android 1.0 je bila nato izdana šele septembra 2008 (prvi pametni telefon, ki je imel nameščen operacijski sistem Android, je bil HTC Dream), razvijalcem pa je bil istega leta predan tudi SDK 1.0. Oktobra 2008 je Google pod licenco Apache objavil celotno izvorno kodo, ki jo lahko spreminja in dopolnjuje vsakdo. Od prve izdaje leta 2008 je Android doživel že številne posodobitve. Posodobitve se po navadi usmerjajo na odpravljanje napak, uvedejo pa se tudi nove lastnosti. Kot zanimivost lahko povemo, da so vse verzije operacijskega sistema Android poimenovane po različnih sladicah. Do sedaj so bile razvite naslednje verzije operacijskega sistema Android (pri vsaki verziji je napisna trenutna razširjenost, kar prikazuje tudi graf na sliki 12):

- Cupcake (Android 1.5, API 3) – 0,6%,
- Donut (Android 1.6, API 4) – 1,0%,
- Eclair (Android 2.0 in 2.1, API 7) – 7,6%,

- Froyo (Android 2.2, API 8) – 27,8%,
- Gingerbread (Android 2.3.x, API 9-10)- 58,6%,
- Honeycomb (Android 3.x.x, API 11-13) – 3,4%,
- Ice Cream Sandwich (Android 4.0.x, API 14-15) – 1,0%.



Slika 12: Graf trenutne razširjenosti različnih verzij operacijskem sistema Android.

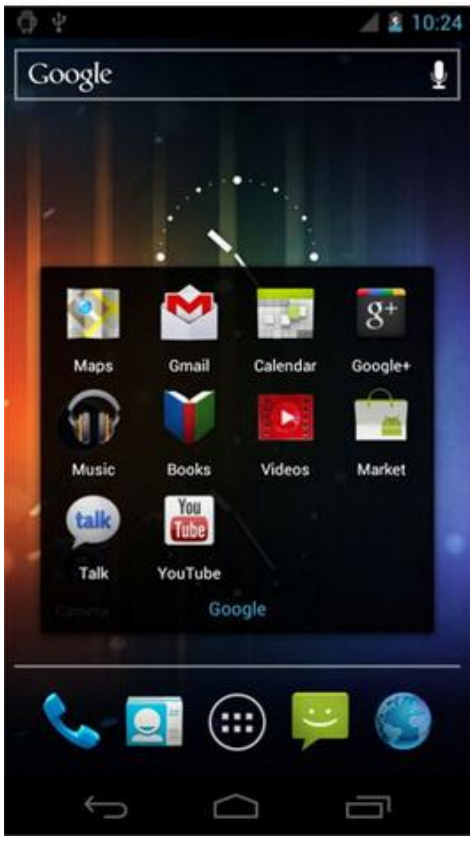
Mobilne naprave, ki imajo nameščen operacijski sistem Android, imajo običajno že privzeto nameščene aplikacije, ki so del AOSP, kot so npr.:

- odjemalec za elektronsko pošto,
- popolna PIM programska oprema, vključno s koledarjem in upravljavcem s stiki,
- zmogljiv spletni brskalnik,
- predvajalnik glasbe, video vsebin ter pregledovalnik slik,
- aplikacije za snemanje slik in video vsebin,
- itd.

Velikokrat pa imajo nameščene tudi mobilne aplikacije podjetja Google kot so npr.:

- Android Market,
- Google zemljevidi (angl. *Google Maps*),
- poštni odjemalec za Gmail,
- Google klepetalnica (angl. *Google Talk*),
- YouTube video predvajalnik,
- itd.

Android vse aplikacije (tudi aplikacije zunanjih razvijalcev) obravnava enako, saj so vse napisane z enakim API (obstajajo pa različne verzije API, ki določajo, na katerih verzijah Androida lahko te aplikacije nameščamo). Tako lahko naložimo, odstranimo in zamenjamo katerokoli aplikacijo, zamenjamo lahko npr. celo klicalnik (angl. *dialer*), katerega nato nadomestimo s tistim, ki smo ga razvili sami. Android od oktobra 2008 ponuja spletno mobilno aplikacijo Android Market, kjer se nahaja že preko pol milijona različnih (brezplačnih in plačljivih) aplikacij, katerih skupno število prenosov je že preseglo 10 milijard. Najnovejša različica operacijskega sistema je Android 4.0 (slika 13), ki se drugače imenuje tudi Ice Cream Sandwich.



Slika 13: Izgled mobilnega operacijskega sistema Android 4.0.

Lastnosti in funkcionalnosti mobilnega operacijskega sistema Android

Android ima že v osnovi naslednje lastnosti in funkcionalnosti [20] (proizvajalci mobilnih naprav, katere poganja operacijski sistem Android, običajno dodajo še druge funkcionalnosti, ki pa jih tu ne bomo omenili):

- **postavitev na zaslonu** (angl. *handset layouts*) – prilagodljiv je za optimiziran prikaz tako na večjih VGA (2D in 3D grafične knjižnice, ki temeljijo na OpenGL ES 2.0 specifikaciji), kot tudi na običajnih zaslonih pametnih mobilnih telefonov;

- **shranjevanje podatkov** (angl. *storage*) – za lokalno shranjevanje podatkov se uporablja SQLite podatkovna baza;
- **povezljivost** (angl. *connectivity*) – podprta so vsa sodobna mobilna omrežja in brezžične komunikacije, ki smo jih omenili v poglavju 2.1.1 (npr. GSM, EDGE, UMTS, HSDPA, Bluetooth, WiFi itd.);
- **sporočanje** (angl. *messaging*) – na voljo so tako klasični načini pošiljanja in sprejemanja sporočil (SMS in MMS) kot tudi napredni Android C2DM način sporočanja. Ta način sporočanja omogoča razvijalcem Android aplikacij pošiljanje sporočil njihovim aplikacijam oziroma napravam, ki imajo nameščene te aplikacije, kar preko strežnika;
- **večjezična podpora** (angl. *multiple language support*);
- **spletni brskalnik** (angl. *web browser*) – temelji na odprtokodni WebKit ter JavaScript (Chrome V8) zasnovi. Na Acid3 testu (to je spletna stran, kjer lahko preverimo, ali je spletni brskalnik skladen z različnimi spletnimi standardi) je dosegel vse možne točke (100/100);
- **podpora Javi** (angl. *Java support*) – čeprav je večina Android aplikacij napisanih v programskem jeziku Java, na Androidu ni nameščenega JVM, prav tako se ne izvaja prevedena bitna koda (angl. *byte code*). Vsi javanski razredi se namreč prevedejo v izvršljivo datoteko Dalvik (angl. *Dalvik executable*) s končnico `.dex`, ki jo poganja Dalvik navidezni stroj DVM. Dalvik navidezni stroj je zasnovan posebej za Android in optimiziran za naprave, ki imajo omejen delovni pomnilnik in moč procesorja. Podporo J2ME lahko zagotovimo preko aplikacij zunanjih razvijalcev;
- **multimedijska podpora** (angl. *media support*) – podprti so skoraj vsi sodobni avdio in video formati, kot so npr. H.263, H.264, MPEG-4, AAC, HE-AAC, MP3, MIDI, Ogg Vorbis, FLAC, WAV, JPEG, PNG itd.;
- **podpora za dodatno strojno opremo** (angl. *additional hardware support*) – podpira delovanje in upravljanje najrazličnejše strojne opreme kot so fotoaparat, video kamera, zaslon na dotik, GPS, merilec pospeškov, 3D grafični pospeševalnik itd.;
- **podpora večdotičnosti** (angl. *multitouch*) – omogoča zaznavanje dotikov na več mestih hkrati, kar omogoča izvajanje tudi zahtevnejših ukazov, kot so vrtenje, približevanje in obračanje. Funkcija je bila na ravni jedra sprva onemogočena (da bi se izognili kršenju patenta tehnologije zaslona na dotik podjetja Apple), danes pa to ni več potrebno;
- **večopravilnost** (angl. *multitasking*) – naenkrat lahko teče (v ospredju ali ozadju) več aplikacij hkrati;
- **glasovno upravljanje** (angl. *voice based features*) – že od vsega začetka je na voljo glasovno iskanje preko iskalnika Google, od verzije Android 2.2 pa so na voljo tudi glasovni ukazi za opravljanje klicev, pisanje SMS sporočil, upravljanje navigacije itd.;

- **deljenje podatkovne povezave** (angl. *tethering*) – omogoči mobilni napravi, da deluje kot WiFi dostopovna točka na katero se lahko priključimo, kar omogoča da lahko uporabljamo internetno povezavo na npr. prenosnem računalniku, četudi v bližini ni nobenega brezžičnega omrežja;
- **podpora zunanjim pomnilnikom** (angl. *external storage*) – večina mobilnih naprav ima microSD režo, tako da Android nudi podporo microSD karticam, ki so formatirane s pomočjo datotečnega sistema FAT32, Ext3fs ali Ext4fs. Drugi datotečni sistemi v osnovi niso podprti, vendar obstajajo rešitve zunanjih razvijalcev, ki to omogočajo.

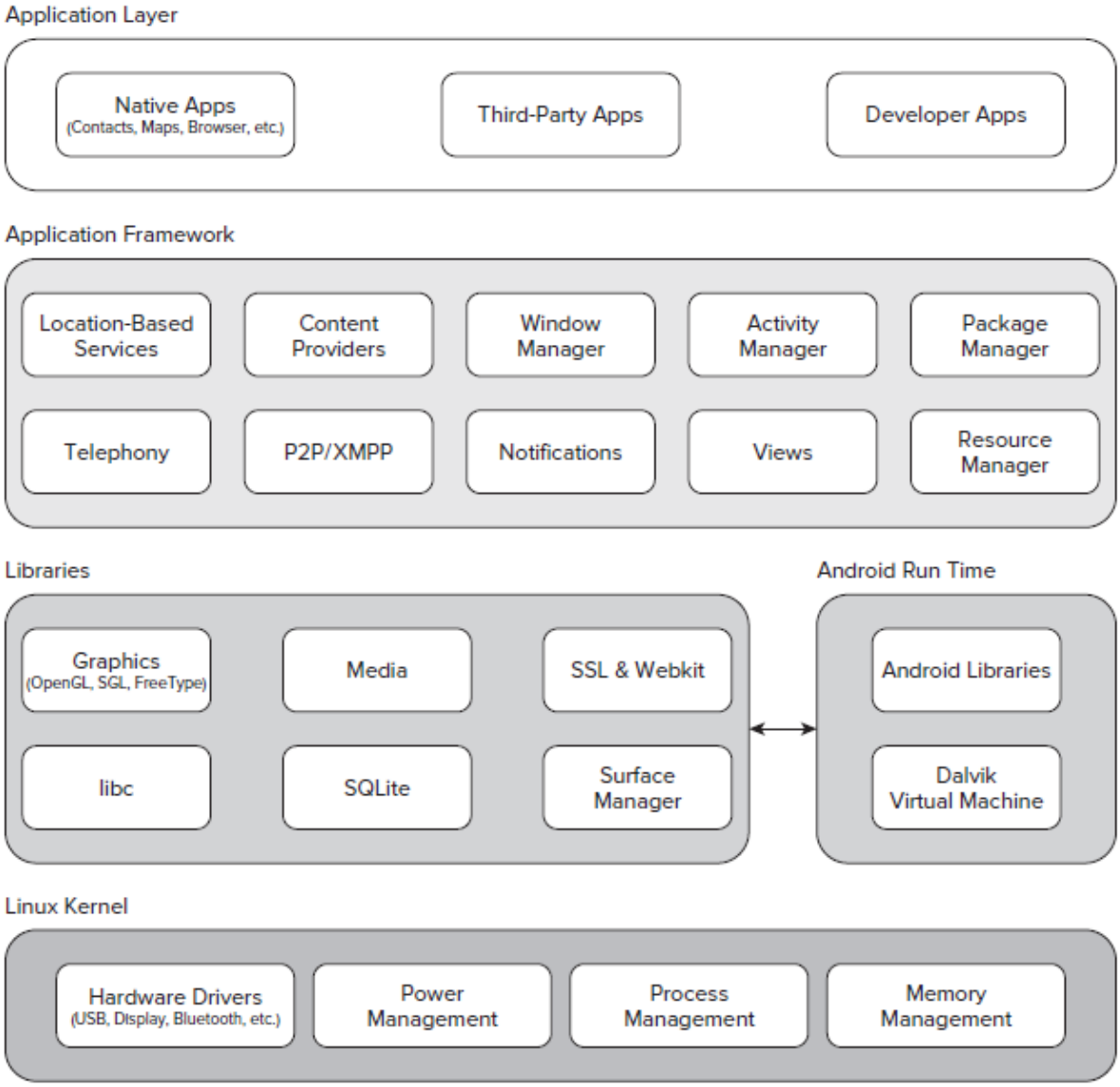
Zgradba mobilnega operacijskega sistema Android

Operacijski sistem Android je sestavljen iz naslednjih elementov (slika 14) [1, 2, 4]:

- **jedro operacijskega sistema Linux** (angl. *Linux kernel*) – glavne naloge jedra so upravljanje s pomnilnikom, zagotavljanje varnosti in omrežnih povezav, upravljanje s porabo energije itd. Prav tako jedro vsebuje gonilnike strojne opreme in predstavlja abstrakcijsko plast med strojno opremo ter ostalimi deli operacijskega sistema;
- **knjižnice** (angl. *libraries*) – napisane so v programskih jezikih C oziroma C++ in jih kot razvijalci uporabljamo za dostop do strojnih komponent mobilne naprave in do komponent samega operacijskega sistema (navaden uporabnik do njih nima dostopa). Pomembnejše knjižice so `libc` (standardna knjižnica programskega jezika C), `SSL` in `Webkit` (za internetno varnost in integriran spletni brskalnik), knjižnice za delo s podatkovno bazo SQLite itd.;
- **android izvajalno okolje** (angl. *Android run time*) – vključuje jedrne (javanske) Android knjižnice in navidezni stroj Dalvik, s čimer poskrbi za izvajanje aplikacij in skupaj s prej opisanimi knjižnicami tvori osnovo za aplikacijsko ogrodje. Jedrne Android knjižnice zagotavljajo večino funkcionalnosti, ki so na voljo v standardnih javanskih knjižnicah, prav tako pa zagotavljajo funkcionalnosti knjižnic, ki so specifične za Android. Dalvik uporablja posebno bitno kodo, tako da na Androidu standardne javanske bitne kode ne moremo pognati. Zato se uporablja orodje imenovano DX, ki omogoča pretvarjanje javanskih razrednih datotek (angl. *java class files*) v izvršljivo `.dex` datoteko, ki se potem izvaja na Dalvik navideznem stroju. Dalvik omogoča izvajanje več aplikacij hkrati, ki se izvajajo kot procesi oziroma primerki (angl. *instances*) virtualnega stroja (vsak proces ima svojo instanco navideznega stroja);
- **aplikacijsko ogrodje** (angl. *application framework*) – vsebuje razrede za ustvarjanje Android aplikacij (razrede lahko tudi poljubno spreminjamo), prav tako pa zagotavlja dostop do strojne opreme ter upravlja z uporabniškim vmesnikom ter aplikacijskimi

viri. S tem imamo zunanji razvijalci dostop do popolnoma enakega API-ja, kot je bil uporabljen pri razvoju privzeto nameščenih aplikacij, kar nam zagotavlja, da Android vse aplikacije obravnava enako. Bolj podrobno bomo aplikacijsko ogrodje predstavili v poglavju 2.1.6;

- **aplikacije** (angl. *application layer*) – na aplikacijskem nivoju se nahajajo že privzeto nameščene aplikacije ter tudi tiste, ki jih uporabniki namestimo sami. Vse aplikacije so napisane s pomočjo programskega jezika Java (za gradnjo grafičnih vmesnikov uporabljamo XML) in so, kot že rečeno, vse zgrajene s popolnoma enakim API. To je edini nivo, ki je viden navadnim uporabnikom, ostalih nivojev se jim ni potrebno zavedati oziroma se jih ne zavedajo. Android aplikacije bomo podrobneje predstavili v poglavju 2.1.6.



Slika 14: Zgradba mobilnega operacijskega sistema Andoid.

2.1.6 Zgradba in lastnosti Android aplikacije

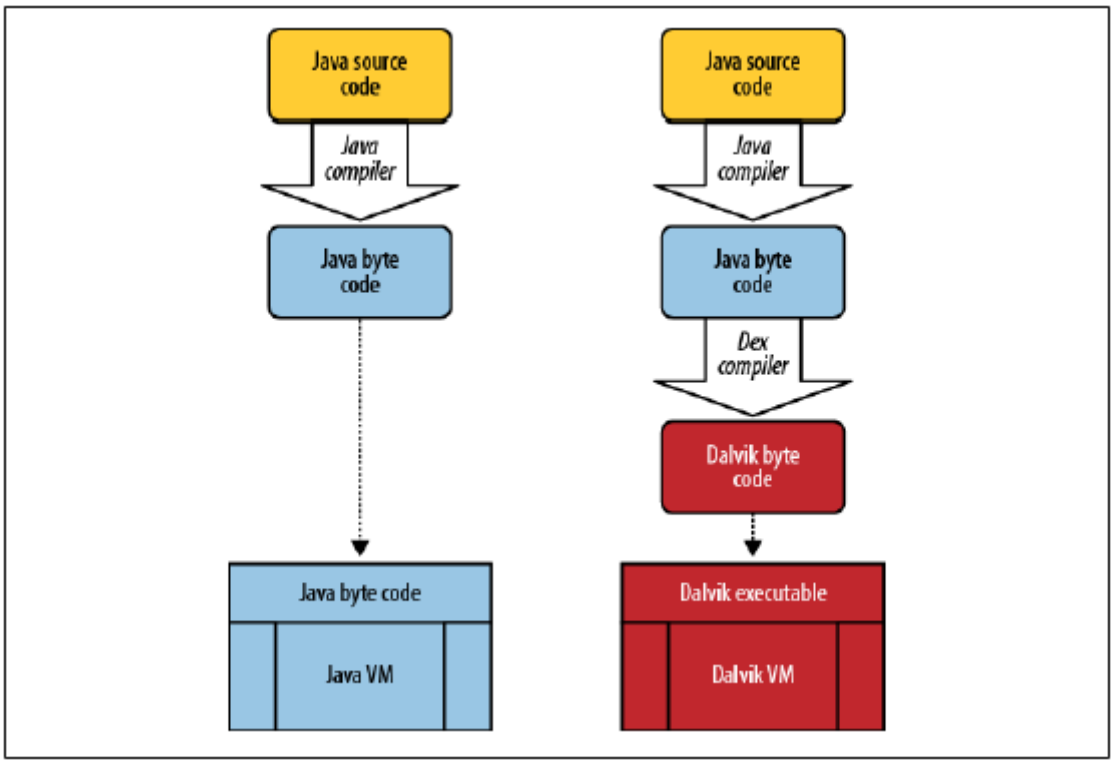
Android ponuja arhitekturo aplikacij [2, 4], ki spodbuja koncept ponovne uporabe že uporabljenih komponent, omogoča objavo in izmenjavo aktivnosti, storitev in podatkov z drugimi aplikacijami, pri čemer razvijalci sami določimo varnostne omejitve (lahko npr. dovolimo, da ima aplikacija dostop do podatkovne povezave, zunanjih pomnilnikov itd.). Naslednje aplikacijske storitve, ki se nahajajo v aplikacijskem ogrodju, so arhitekturni temelj vseh Android aplikacij, ki ga rabimo pri gradnji lastnih aplikacij:

- **upravitelj aktivnosti** (angl. *activity manager*) – nadzira celoten življenjski cikel aktivnosti, ki jih aplikacija uporablja;
- **pogledi** (angl. *views*) – uporabljajo se za gradnjo uporabniških vmesnikov za posamezne aktivnosti aplikacije. Urejeni so hierarhično in se znajo izrisati sami. Primer pogleda je npr. gumb (angl. *button*), oznaka (angl. *label*) itd.;
- **upravitelj obveščanja** (angl. *notification manager*) – zagotavlja dosleden in nevsiljiv mehanizem za obveščanje uporabnika pri uporabi aplikacije;
- **ponudnik vsebin** (angl. *content provider*) – aplikaciji omogoča izmenjavo podatkov z drugimi aplikacijami;
- **upravitelj z viri** (angl. *resource manager*) – ponuja dostop do virov, kot so npr. nizi, slike itd.

Velika večina Android aplikacij je napisanih v programskem jeziku Java. Orodja Android SDK prevedejo programsko kodo, skupaj z morebitnimi ostalimi viri in podatki, v arhivsko datoteko s končnico `.apk`. Pakiranje izvede program AAPT. Datoteka `.apk` je sestavljena iz treh glavnih komponent:

- **izvršljiva Dalvik datoteka** – tu se nahaja vsa izvorna koda, ki omogoča izvajanje aplikacije. Izvorne datoteke Java (angl. *Java source files*) so najprej s prevajalnikom Java (angl. *Java compiler*) prevedene v razredne datoteke Java (angl. *java class files*), ki pa so nato z orodjem, imenovanim DX, pretvorjene v eno izvršljivo `.dex` datoteko, ki se potem izvaja na Dalvik navideznem stroju. Primerjavo poteka prevajanja in izvajanja navadne Java aplikacije in Android aplikacije prikazuje slika 15;
- **viri** (angl. *resources*) – predstavlja vse vire, ki jih aplikacija uporablja, razen same kode aplikacije. Viri so lahko npr. slike, XML opisne datoteke, ki opisujejo uporabniške vmesnike aplikacije itd.;
- **osnove knjižnice** (angl. *native libraries*) – aplikacija lahko vključuje tudi nekaj osnovne kode, kot so npr. C/C++ knjižnice.

Ta datoteka predstavlja aplikacijo, ki jo nato lahko namestimo na katerokoli napravo, ki ima nameščen operacijski sistem Android. Še preden je neka aplikacija lahko nameščena, pa mora



Slika 15: Primerjava navadne Java aplikacije in Android aplikacije.

biti podpisana (angl. *signed*). Vsaka nameščena aplikacija živi v t.i. varnostnem peskovniku (angl. *security sandbox*) za katerega velja:

- operacijski sistem Android je večuporabniški operacijski sistem, v katerem je vsaka aplikacija različen uporabnik,
- operacijski sistem vsaki aplikaciji privzeto dodeli edinstveno uporabniško ime (angl. *Linux user ID*), ki je viden samo sistemu, ne pa tudi aplikaciji. Sistem prav tako nastavi dovoljenja vsem datotekam v tej aplikaciji, tako da do njih lahko dostopa samo uporabnik, ki ima enako uporabniško ime, kot je bilo dodeljeno aplikaciji,
- vsak proces uporablja lastno instanco navideznega stroja Dalvik, zato vsaka aplikacija teče v izolaciji od drugih aplikacij,
- vsaka aplikacija privzeto teče v lastnem procesu. Android začne nov proces, ko mora biti izvršena katerakoli komponenta neke aplikacije, prekine pa ga takrat, ko proces ni več potreben oziroma sistem potrebuje pomnilnik za pravilno izvajanje drugih aplikacij.

Na ta način operacijski sistem uveljavlja sistem najmanjšega privilegija. To pomeni, da ima vsaka aplikacija na voljo samo dostop do virov, ki jih potrebuje za delovanje, kar nam omogoča, da aplikacija ne more dostopati do delov sistema, za katere nima dovoljenja.

Obstajata dva načina, kako lahko aplikacija deli podatke z drugimi aplikacijami in kako lahko aplikacija dostopa do sistemskih storitev:

- lahko zagotovimo, da imata dve različni aplikaciji enako uporabniško ime ter tako omogočimo, da obe dostopata do datotek obeh aplikacij. V tem primeru lahko aplikaciji tečeta znotraj istega procesa in si delita isti navidezni stroj,
- aplikacija lahko zahteva dovoljenje za dostop do različnih sistemskih podatkov in virov (zunanji pomnilnik, podatkovna povezava itd.), vendar mora biti to potrjeno s strani uporabnika med nameščanjem aplikacije.

Aplikacijske komponente [2, 6, 10, 18], ki jih bomo v nadaljevanju podrobneje predstavili, so osnovni gradniki vsake Android aplikacije. Vsaka Android aplikacija je sestavljena iz ene ali več aplikacijskih komponent. Vsaka aplikacijska komponenta je nekakšna vstopna točka, preko katere lahko operacijski sistem vstopi v aplikacijo. Čisto vse komponente niso vstopna točka za uporabnika, saj so lahko nekatere odvisne ena od druge, vendar pa vsaka obstaja kot svoja entiteta in igra točno določeno vlogo. Lahko rečemo, da je vsaka komponenta v aplikaciji edinstven gradnik aplikacije, ki pomaga definirati splošno vedenje aplikacije. Vsako komponento se lahko aktivira posamično. Poznamo štiri osnove aplikacijske komponente. Vsaka izmed njih ima različen namen in življenjski cikel, ki definira, kako je neka komponenta ustvarjena in uničena. Osnovne aplikacijske komponente so:

- **aktivnosti** (angl. *activities*) – so najpogostejši gradniki vsake Android aplikacije. Aplikacijo lahko sestavlja ena ali več aktivnosti, ki so med seboj neodvisne, vsaka od njih pa običajno predstavlja zaslon z uporabniškim vmesnikom (aktivnosti so sicer lahko tudi transparentne oziroma prikazane kot dialogi). Za primer lahko vzamemo aplikacijo za elektronsko pošto, kjer ena od aktivnosti skrbi za prikazovanje novih elektronskih sporočil, medtem ko druga skrbi za sestavljanje oziroma pošiljanje novega elektronskega sporočila. Ker so aktivnosti med seboj neodvisne, lahko vsaka aplikacija aktivira katerokoli komponento druge aplikacije (seveda mora to druga aplikacija tudi dovoliti). Primer tega je, ko aplikacija za pregledovanje fotografij lahko začne aktivnost v aplikaciji za pošiljanje elektronske pošte, tako da se ustvari novo elektronsko sporočilo, ki že vsebuje pripeto fotografijo. Tipično je ena od aktivnosti glavna (angl. *main*) aktivnost, ki je pojavi takoj ob zagonu aplikacije. Vsaka od aktivnosti je razširitev razreda `Activity`, sestavljena pa je iz enega ali več pogledov, ki urejajo vizualno podobo okna oziroma sestavljajo uporabniški vmesnik. Aktivnosti so ena od najpomembnejših gradnikov aplikacije, saj določajo zgradbo in samo obnašanje aplikacije, prav tako pa so edina komponenta, s katero uporabnik aplikacije komunicira preko uporabniškega vmesnika;

- **storitve** (angl. *services*) – so komponente, ki tečejo v ozadju in običajno opravljajo dalj časa trajajoče operacije tako lokalnih (v primeru zagotavljanja storitve eni aplikaciji) kot tudi oddaljenih (v primeru zagotavljanja storitve več aplikacijam) procesov. Storitve nimajo nikakršnega uporabniškega vmesnika oziroma ga tudi ne potrebujejo. Primer storitev je npr. prenašanje podatkov iz omrežja, medtem ko uporabnik uporablja katero od aplikacij (prenašanje podatkov teče v ozadju, uporabnik se tega v bistvu sploh ne zaveda);
- **ponudniki vsebin** (angl. *content providers*) – upravljajo s skupno rabo podatkov različnih aplikacij, saj je deljenje podatkov med različnim mobilnimi aplikacijami zelo pogost pojav. Ponudniki vsebin nimajo nikakršnega uporabniškega vmesnika. Podatki, ki jih neka aplikacija potrebuje pri delovanju, so lahko shranjeni v datotečnem sistemu, SQLite podatkovni bazi, na svetovnem spletu itd. Preko ponudnika vsebin je drugim aplikacijam omogočen dostop do teh podatkov, včasih te podatke (če ponudnik vsebine to dovoli) lahko tudi spreminjajo. Za primer lahko vzamemo ponudnika vsebin, ki skrbi za upravljanje uporabnikovih osebnih stikov. Vsaka aplikacija, ki ima primerna dovoljenja, lahko pri svojem delovanju uporabi katerekoli podatke o določeni osebi;
- **sprejemniki napovedi** (angl. *broadcast receivers*) – gre za komponente, ki se odzivajo na ravni celotnega sistema v primeru napovedi nekega dogodka. Sprejemniki napovedi dogodkov nimajo nikakršnega uporabniškega vmesnika, lahko pa uporabnika o nekem dogodku opozorijo oziroma obvestijo preko statusne vrstice (angl. *status bar notification*). Veliko napovedanih dogodkov izvira iz sistema (recimo prazna baterija, izklopljen zaslon itd.), prav tako pa lahko napovedi dogodkov sprožijo same aplikacije kot npr. v primeru, da so bili neki podatki preneseni na napravo, kateri so sedaj na voljo za uporabo.

Android aplikacija je običajno sestavljena iz ene ali več osnovnih aplikacijskih komponent, ni pa nujno potrebno, da vsaka aplikacija vsebuje vse štiri vrste aplikacijskih komponent. Ko smo enkrat odločeni, katere komponente bomo v naši aplikaciji uporabili, jih navedemo v datoteki `AndroidManifest.xml` [10]. To je XML datoteka, ki poleg komponent naše aplikacije vsebuje tudi podatke o tem:

- katera uporabniška dovoljenja (angl. *user permissions*) aplikacija zahteva (npr. internetna povezava, dostop do zunanjega pomnilnika itd.),
- katera je minimalna API stopnja (angl. *API level*), ki jo zahteva aplikacija, glede na to, kateri API uporablja aplikacija,
- katera programsko in strojno oprema bo aplikacija uporabljala (npr. fotoaparat),
- katere zunanje API knjižnice so uporabljene v aplikaciji (npr. knjižnica Google Maps),
- itd.

Posebnost Android aplikacij je tudi v tem, da lahko vsaka aplikacija aktivira komponento neke druge aplikacije. Ko sistem aktivira neko komponento aplikacije, se ustvari nov proces za to aplikacijo (če seveda še ni ustvarjen). Za razliko od večine drugih operacijskih sistemov, Android aplikacije nimajo neke enotne vstopne točke (npr. nimajo funkcija `main()`, kot jo ima večina drugih aplikacij), ampak se začnejo tam, kjer se aktivira neka komponenta. Zaradi tega, ker v operacijskem sistem Android vsaka aplikacija teče kot svoj proces, aplikacija ne more direktno aktivirati komponente druge aplikacije, ampak to lahko naredi le operacijski sistem sam. Če želimo torej v neki aplikaciji aktivirati komponento druge aplikacije, moramo svojo namero sporočiti sistemu, ki nato poskrbi za aktiviranje te komponente.

Aktiviranje komponent

Komponente (aktivnosti, storitve in sprejemniki napovedi) so aktivirane s pomočjo asinhronskega sporočila, imenovanega namera (angl. *intent*) [10]. Namere vežejo posamezne komponente med seboj v času izvajanja, ne glede na to kateri aplikacijo pripadajo. Ustvarjena je s pomočjo objekta razreda `Intent`, ki definira sporočilo za aktiviranje specifične komponente (ali samo specifičnega tipa komponente), zato so lahko namere bodisi eksplicitne bodisi implicitne. Ponudnikov vsebin ne aktivirajo namere, ampak so aktivirani na zahtevo objekta razreda `ContentResolver`. Obstaja nekaj različnih metod za aktiviranje vsake izmed komponent:

- aktivnost lahko aktiviramo (oziroma ji dodelimo novo delo) tako, da se namera posreduje metodi `startActivity(Intent intent)` ali `startActivityForResult(Intent intent)`,
- storitev lahko aktiviramo oziroma obstoječi storitvi dodelimo nova navodila tako, da se namera posreduje metodi `startService(Intent intent)`,
- sprejemnik napovedi aktiviramo tako, da se nameri posreduje metodi `sendBroadcast(Intent intent)`, `sendOrderedBroadcast(Intent intent)` ali `sendStickyBroadcast(Intent intent)`,
- pri ponudnikih vsebin lahko izvedemo poizvedbo (angl. *query*) s pomočjo metode `query()` iz razreda `ContentResolver`.

Življenjski cikel aktivnosti

Operacijski sistem Android z aktivnostmi upravlja s pomočjo sklada aktivnosti (angl. *activity stack*) [2, 4, 9, 10]. Na novo ustvarjena aktivnost dobi mesto na vrhu sklada in postane trenutno izvajajoča se aktivnost, medtem ko se pred tem ustvarjene aktivnosti še vedno nahajajo nižje na skladu. Ko ta nova aktivnost preneha obstajati (ter če ni ustvarjena nova),

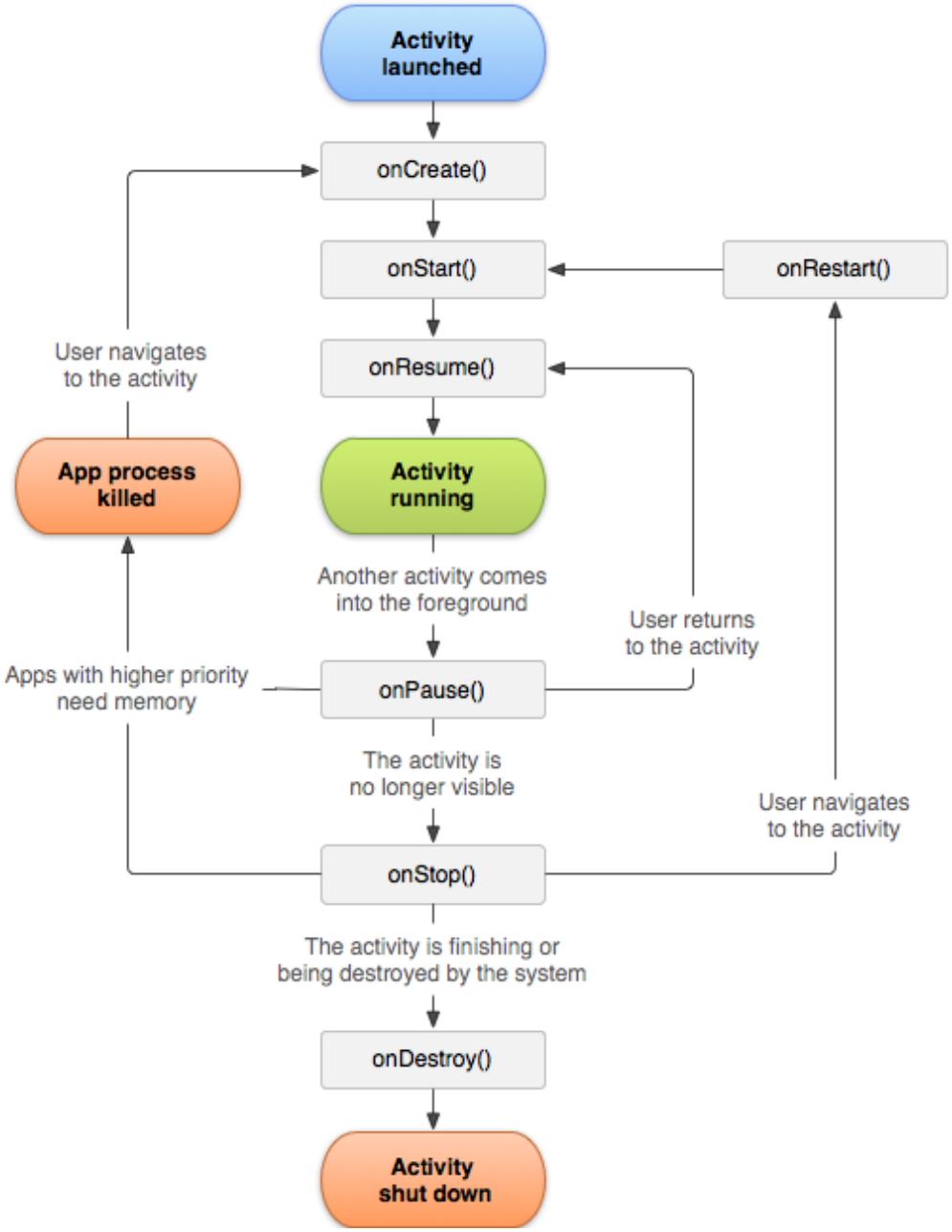
postane trenutno izvajajoča tista aktivnost, ki je bila do sedaj tik pod vrhom sklada (če seveda sklad aktivnosti ni bil prazen). Aktivnost se lahko nahaja v enem izmed štirih osnovnih stanj:

- če je aktivnost v ospredju (torej se nahaja na vrhu sklada aktivnosti), potem je v stanju izvajanja oziroma je aktivna (angl. *active*),
- če aktivnost izgubi fokus, vendar je še vedno vidna (ustvarjena je bila nepopolna oz. transparentna aktivnost, ko je npr. dialog), je v stanju mirovanja (angl. *pause*). Aktivnost v tem stanju je popolnoma živa (ohranja vsa stanja in informacije), vendar je vseeno lahko uničena (angl. *killed*) v primeru pomanjkanja spomina,
- če je aktivnost popolnoma zakrita z drugo aktivnostjo (ustvarjena je bila nova aktivnost), je ustavljena (angl. *stopped*). Čeprav ni več vidna, se ohranjajo vsa stanja in informacije, vendar je pogosto uničena v primeru pomanjkanja spomina,
- če je aktivnost v stanju mirovanja oziroma ustavljena, lahko sistem konča aktivnost, tako da zahteva zaključek aktivnosti, lahko pa jo preprosto ubije (aktivnosti ni več na skladu aktivnosti). Ko je končana aktivnost znova prikazana, jo je treba zagnati znova in povrniti v prejšnje stanje.

Slika 16 prikazuje prehajanja med vsemi možnimi stanji aktivnosti. V primeru, da je neka aktivnost na novo ustvarjena, se kliče metoda `onCreate()`. Uporabljamo jo za izvedbo enkratnih inicializacijskih prijemov, na primer določanje uporabniškega vmesnika. Klicu metode `onCreate()` vedno sledi klic metode `onStart()`, ne velja pa obratno (klic metode `onStart()` ni nujno posledica klica metode `onCreate()`, saj se lahko kliče tudi v primeru, ko je bila aktivnost ustavljena in jo želimo nadaljevati). Ob klicu metode `onStart()` aktivnost še ni vidna uporabniku, vendar bo kmalu. Metoda `onResume()` se kliče po klicu metode `onStart()`, ko pride aktivnost v ospredje in je vidna uporabniku (uporabnik komunicira z aktivnostjo preko uporabniškega vmesnika). Ko se uporabnik odloči začeti z neko drugo aktivnostjo, se bo za trenutno aktivnost uporabil klic metode `onPause()`. Klicu metode `onPause()` pa lahko nato sledi bodisi klic metode `onResume()` ali `onStop()`. Metoda `onResume()` se kliče takrat, ko se aktivnost znova vrne v ospredje (npr. ko je bila ustvarjena transparentna aktivnost), medtem ko se metoda `onStop()` kliče takrat, ko aktivnost uporabniku ni več vidna (npr. ustvarjena je bila nova polna aktivnost). Če v ospredje znova želimo postaviti aktivnost, na kateri se je že klicala metoda `onStop()`, se bo klicala metoda `onRestart()` v primeru, da aktivnost še ni bila uničena (aktivnost prehaja iz stanja, ko je bila ustavljena), v nasprotnem primeru pa metoda `onCreate()`. V primeru, da se aktivnost nahaja na skladu aktivnosti in je nevidna uporabniku, jo sistem lahko ubije s klicem metode `onDestroy()`.

Čeprav se model prehajanja stanj (slika 16) na prvi pogled zdi zahteven, se nam pri izdelavi aplikacije običajno ni potrebno ukvarjati z vsemi možnimi scenariji. Pri gradnji uporabniških

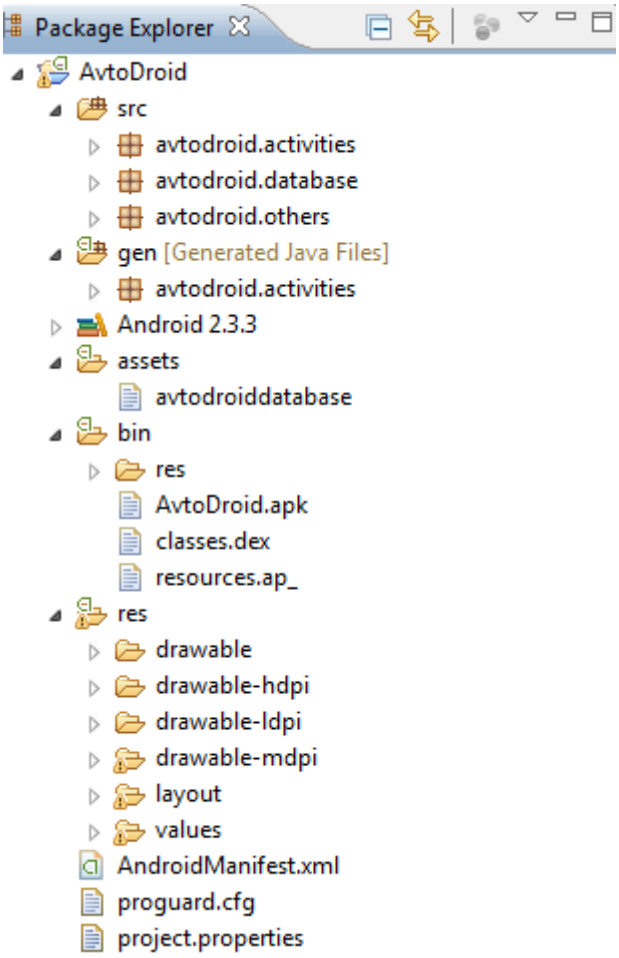
vmesnikov v aplikaciji se najpogosteje ukvarjamo z implementacijo metode `onCreate()`, kjer vsem gradnikom lahko določimo izgled, vsebino, rokovanje z dogodki itd. Pri metodi `onPause()` se ukvarjamo predvsem z podatki, ki jih naša aktivnost uporablja in še niso shranjeni, saj je to zadnja varna metoda, ki se izvede, preden sistem ubije aplikacijo (in s tem tudi vse neshranjene podatke).



Slika 16: Prehod med stanji aktivnosti.

Struktura Android aplikacije v razvojnem okolju

Kljub temu, da je velikost in zahtevnost Android aplikacij lahko zelo različna, je njihova struktura zelo podobna [2]. Na sliki 17 je prikazana struktura mobilne aplikacije AvtoDroid v razvojnem okolju Eclipse, katerega bomo podrobneje predstavili v poglavju 2.2.4.



Slika 17: Sestavni deli Android aplikacije.

- **`AndroidManifest.xml`** – v tej datoteki so definirane aktivnosti, ponudniki vsebin, storitve ter vse lastnosti teh aplikacijskih komponent. Prav tako lahko tu definiramo vsa dovoljenja, ki jih aplikacija potrebuje pri delovanju, ter tudi zagotovimo, da bodo druge aplikacije imele dostop do storitev naše aplikacije. Vsebuje tudi podatke o tem, katero strojno in programsko opremo bo aplikacija uporabljala, kakšni so pogoji za delovanje aplikacije itd.;
- **mapa `src`** – vsebuje vso izvorno koda aplikacije (tu so shranjeni vsi napisani razredi, ki so razporejeni po posameznih paketih);
- **mapa `gen`** – vsebuje vse avtomatsko generirane datoteke, katerih vsebine ne smemo spreminjati;

- **mapa `bin`** – vsebuje vse avtomatske generirane datoteke s strani prevajalnika. Tudi datotek v tej mapi ne smemo spreminjati;
- **mapa `assets`** – vsebuje poljubno zbirko datotek, ki jih želimo vključiti v aplikacijo (npr. datoteko, ki vsebuje podatke, s katerimi želimo napolniti neko tabelo v podatkovni bazi, ki jo aplikacija uporablja);
- **mapa `res`** – vsebuje vse glavne vire aplikacije, saj je to starševski direktorij map `drawable`, `anim`, `layout`, `menu`, `values`;
- **mapa `drawable`** – vsebuje zbirko vseh slik, ki jih uporablja aplikacija;
- **mapa `anim`** – vsebuje zbirko vseh animacij, ki jih uporablja aplikacija;
- **mapa `layout`** – vsebuje vse grafične uporabniške vmesnike aplikacije, ki so shranjeni kot XML opisne datoteke;
- **mapa `menu`** – vsebuje vse XML opisne datoteke za menije, ki so uporabljeni v aplikaciji;
- **mapa `values`** – vsebuje vse ostale vire (razen glavnih), ki jih aplikacija uporablja. Viri so definirani s pomočjo XML opisnih datotek. Primeri virov so barve (angl. *colors*), nizi (angl. *strings*) itd.

Opazimo lahko, da se označevalni jezik XML zelo pogosto uporablja v povezavi z Andorid aplikacijami. Uporabljamo ga npr. za izgradnjo oziroma definiranje uporabniških vmesnikov v aplikaciji, definiranje virov (barve, nizi itd.), ki jih aplikacija uporablja itd. V mobilni aplikaciji AvtoDroid bomo označevalni jezik XML poleg tega uporabljali tudi pri izvozu različnih zapisov podatkovnih tabel v datoteko XML, prav tako pa se bomo z XML datotekami srečali tudi pri pridobivanju nekaterih podatkov iz svetovnega spleta.

2.2 Razvojno okolje

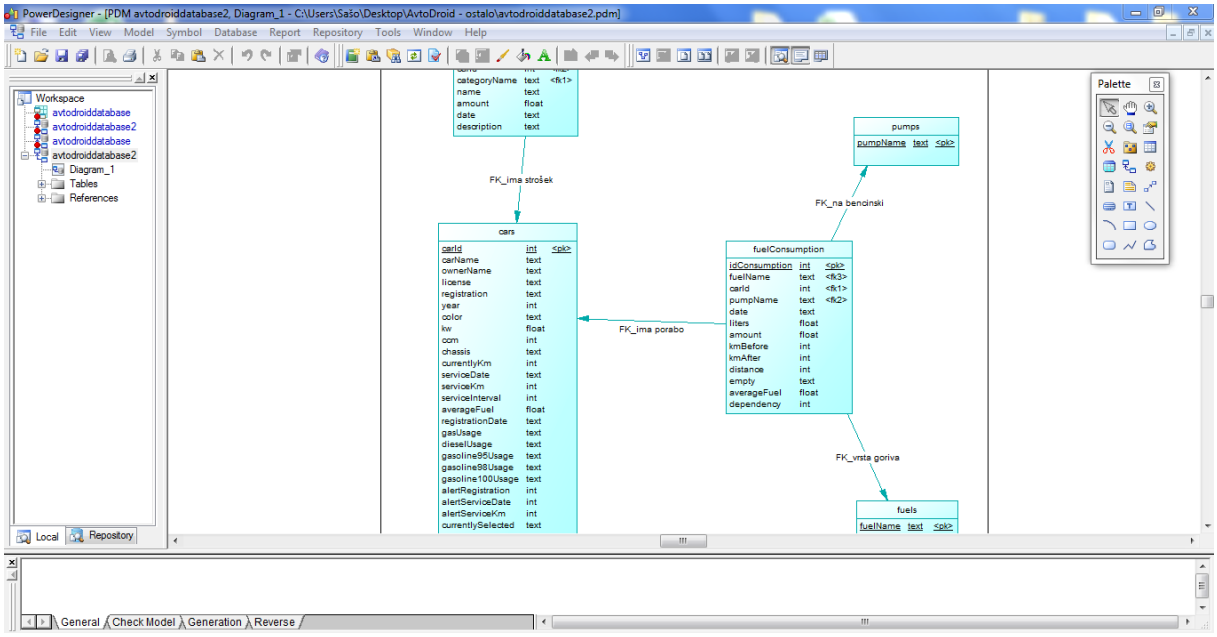
V tem poglavju bomo predstavili vsa orodja, ki smo jih uporabljali pri razvoju mobilne aplikacije AvtoDroid. Za načrtovanje podatkovne baze (konceptualni in logični podatkovni model) smo uporabili orodje PowerDesigner, izgradnje SQLite podatkovne baze (oziroma podatkovnih tabel v bazi) pa smo se lotili z uporabo orodja SQLite Database Browser. Za razvoj aplikacije smo uporabili skupek orodij in knjižnic Android SDK, integrirano razvojno okolje Eclipse ter vtičnik (angl. *plugin*) ADT. Za razvoj Android aplikacij pa je poleg tega potrebno imeti nameščeno tudi JDK in JDT, ki ju ne bomo podrobno predstavili.

2.2.1 PowerDesigner

Orodje PowerDesigner [28] je modelirno orodje, ki ga razvija podjetje Sybase. Kot samostojna aplikacija deluje v operacijskem sistemu Windows, obstaja pa tudi vtičnik za razvojno okolje Eclipse. Omogoča naslednje modelirne tehnike (angl. *modeling techniques*):

- poslovno modeliranje (angl. *business modeling*),
- podatkovno modeliranje (angl. *data modeling*), kamor spadata tudi konceptualni in logični podatkovni model,
- modeliranje podatkovnih skladišč (angl. *data warehouse modeling*),
- XML modeliranje (angl. *XML modeling*), podprto z XML shemami in DTD standardi,
- objektno modeliranje (angl. *object modeling*),
- itd.

Pri izdelavi konceptualnega in logičnega podatkovnega modela smo uporabljali verzijo PowerDesigner 12.5 (slika 18). Orodje lahko brezplačno uporabljamo 15 dni, nato pa je za uporabo potreben nakup licence.

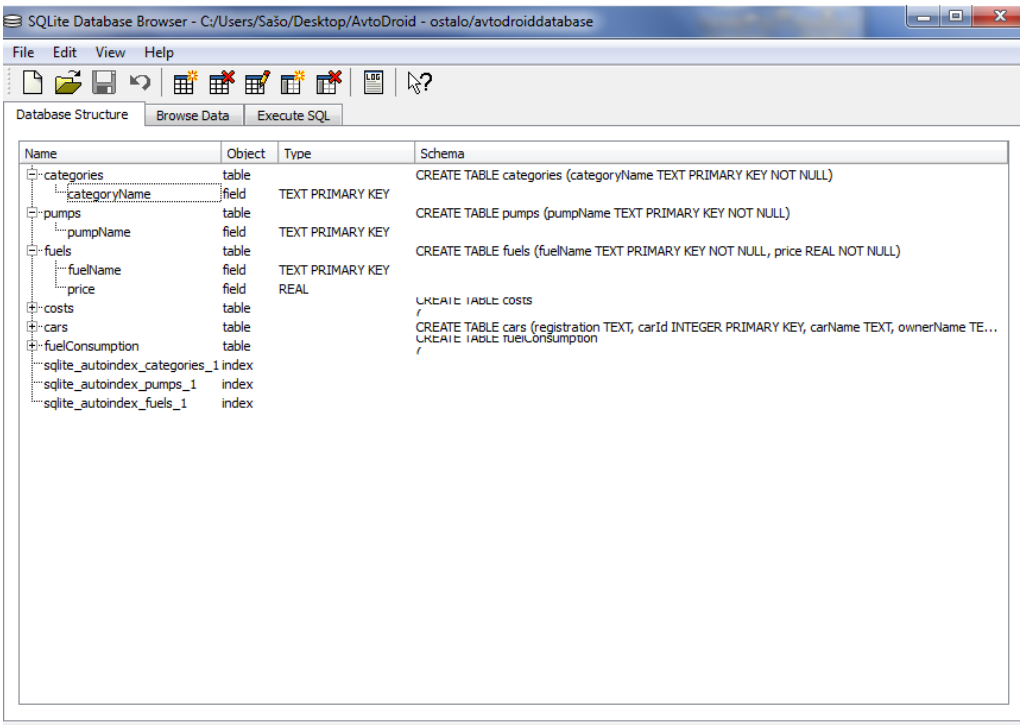


Slika 18: Orodje PowerDesigner.

2.2.2 SQLite Database Browser

Orodje SQLite Database Browser (slika 19) [19] je povsem brezplačno in odprtokodno orodje za ustvarjanje, oblikovanje ter urejanje SQLite podatkovnih baz (oz. podatkovnih tabel v bazi). To orodje je v prvi vrsti namenjeno razvijalcem različnih aplikacij, ki pri svojem delovanju uporabljajo SQLite podatkovno bazo. Omogoča naslednje funkcionalnosti:

- ustvarjanje podatkovnih baz,
- pregledovanje, ustvarjanje, urejanje in brisanje podatkovnih tabel,
- pregledovanje, dodajanje, urejanje in brisanje zapisov v tabelah,
- izvoz in uvoz zapisov tabel v tekstovno datoteko,
- izvajanje poizvedb (angl. *SQL query*).



Slika 19: Orodje SQLite Database Browser.

SQLite je na voljo na vsaki napravi, ki ima nameščen operacijski sistem Android, in ne zahteva nobenih nastavitev oziroma administrativnih posegov. Operacijski sistem Android med knjižnicami, ki so sestavni del operacijskega sistema Android, ponuja tudi knjižnico za delo z SQLite podatkovnimi bazami.

SQLite je odprtokodna podatkovna baza, ki podpira običajne lastnosti relacijskih podatkovnih baz, kot so npr. sintaksa SQL (angl. *SQL syntax*), transakcije itd., poleg tega pa med delovanjem porabi zelo malo delovnega pomnilnika (okoli 250 KB). Za razliko od ostalih relacijskih podatkovnih baz pa podpira nekoliko manj podatkovnih tipov za hranjenje podatkov, saj so podprti samo:

- **`TEXT`** – nizi,
- **`INTEGER`** – cela števila,
- **`REAL`**– decimalna števila,
- **`NULL`** – brez vrednosti,
- **`BLOB`** – poljubna binarna vsebina.

Cena majhnih zahtev SQLite podatkovnih baz je tudi v tem, da ni nobenega preverjanja, če so bili v polje zapisani podatki pravega tipa, torej lahko napišemo nek črkovni niz v polje tipa `INTEGER` in obratno, kar lahko privede do različnih napak med samim izvajanjem aplikacije.

2.2.3 Android SDK

Android SDK [2, 4] vsebuje vse kar potrebujemo, da začnemo razvijati, testirati in razhroščevati (angl. *debug*) Android aplikacije. Paket Android SDK lahko brezplačno prenesemo in namestimo v razvojnem okolju Eclipse. V paketu je vključeno:

- **Android API** – jedro Android SDK predstavljajo Android API knjižnice, ki razvijalcem Android aplikacij omogočajo dostop do komponent operacijskega sistema. Gre za enake knjižnice, kot jih uporabljajo tudi npr. pri podjetju Google za programiranje privzeto nameščenih aplikacij. Obstaja veliko različnih verzij API, ki določajo, na katerih verzijah Androida lahko te aplikacije nameščamo. Vsaka nova verzija API vključuje novosti ter popravke, zato moramo kot razvijalci aplikacije določiti, na kateri API verziji naša aplikacija deluje. Da bi naša aplikacija delovala na čim več napravah (kar je običajno naš cilj), moramo določiti najnižjo verzijo API, na kateri aplikacija še deluje;
- **razvojna orodja** (angl. *development tools*) – da lahko uspešno spremenimo izvorno kodo aplikacije v izvršljivo Android aplikacijo, so v Android SDK vključena številna orodja, ki nam omogočajo pomoč pri prevajanju in razhroščevanju aplikacije. Ta orodja bomo podrobneje predstavili v nadaljevanju poglavja;
- **polna dokumentacija** (angl. *full documentation*) – vključuje vse informacije o tem, kaj posameznimi paketi in razredi vključujejo ter kako jih uporabljati, prav tako pa so tu vključene vse potrebne informacije in navodila, ki jih potrebujemo pri razvijanju Android aplikacij;
- **primeri kode** (angl. *sample code*) – vključuje primere enostavnih Android aplikacij, ki prikazujejo nekatere funkcionalnosti operacijskega sistema Android, prav tako pa so tu vključene izvorne kode preprostih programov, ki nam pojasnijo uporabo posameznih API elementov;
- **spletna podpora** (angl. *online support*) – na voljo so različni forumi, Google skupine (angl. *Google groups*) itd., kjer najdemo odgovore na različna vprašanja, povezana z razvojem Android aplikacij.

Razvojna orodja

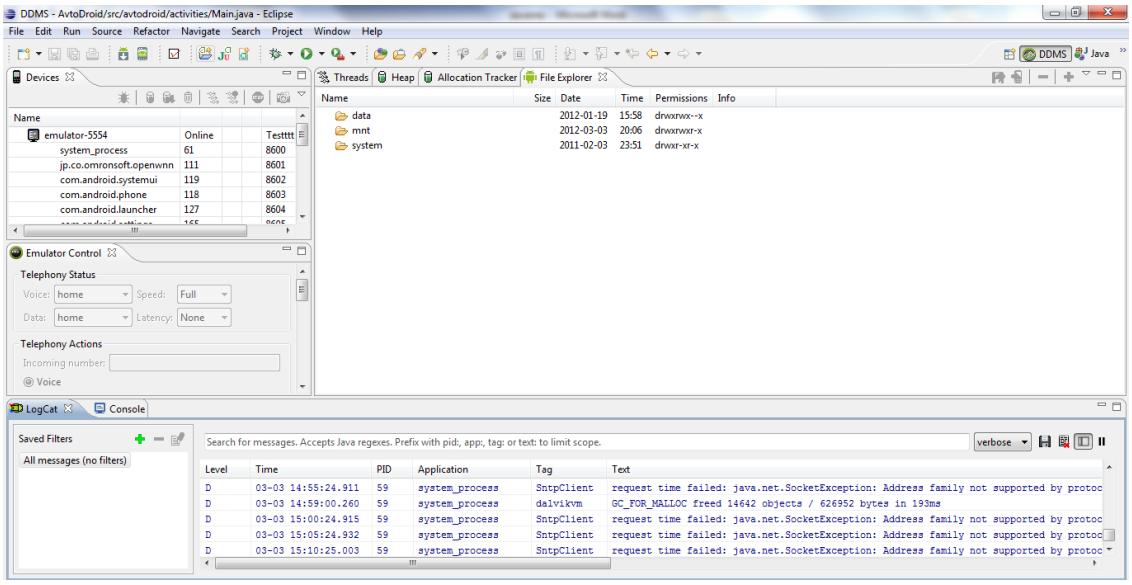
Android SDK vsebuje številna orodja in pripomočke, ki nam pomagajo ustvariti, testirati in razhroščevati Android aplikacije brez uporabe fizičnih oziroma resničnih Android naprav [2, 4]. Vtičnik ADT, ki ga bomo podrobno predstavili v poglavju 2.3.5, za ta orodja zagotavlja grafični uporabniški vmesnik (namesto uporabe orodij preko ukazne vrstice), najlažje pa jih najdemo in uporabljamo v DDMS perspektivi razvojnega okolja Eclipse. Med najpomembnejša orodja spadajo:

- **upravljalnik Android SDK in upravljalnik AVD** (angl. *Android SDK and AVD manager*) – omogočata ustvarjanje in upravljanje z (do sedaj ustvarjenimi) AVD napravami in SDK paketi - vidimo, katere pakte imamo nameščene in kateri so še na voljo;
- **AVD** – vsaki AVD lahko določimo ime, verzijo operacijskega sistema Android, katera SDK verzija je podprta, zaslonsko ločljivost, velikost zunanjega pomnilnika (SD kartice) ter zmogljivost strojne opreme. Tako lahko s pomočjo različnih nastavitev testiramo ogromno tipov mobilnih naprav, ne da bi jih morali kupiti. AVD omogoča izvajanje emulatorja ter določa njegovo konfiguracijo;
- **emulator** – je implementacija Dalvik navideznega stroja in je popolnoma enaka tistim navideznim strojem, ki so del operacijskega sistema fizičnih Android naprav. Deluje na osnovi tehnologije QEMU, ki omogoča emulacijo na ravni CPE. Zagotovljena je tudi polna omrežna povezljivost, pri čemer lahko sami določimo hitrost internetne povezave, lahko pa simuliramo tudi opravljanje klicev in pošiljanje SMS sporočil. Ker se emulator izvaja znotraj AVD, moramo ob zagonu emulatorja naprej ustvariti AVD, katero bo emulator zagnal in znotraj katere bo potem ustvarjen Dalvik navidezni stroj. Vtičnik ADT omogoča integracijo emulatorja v razvojno okolje Eclipse, tako da je zagnan avtomatsko znotraj izbrane AVD. Emulator se uporablja predvsem za testiranje in razhroščevanje aplikacij. Primer emulatorja, na katerem teče operacijski sistem Android 2.3, je prikazan na sliki 20;
- **DDMS** – je zmogljivo razhroščevalno orodje, s katerim lahko spremljamo trenutno aktivne procese, dogajanje na skladu in kopici, simuliramo prej omenjene dogodke (opravljanje klicev, pošiljanje SMS sporočil), raziskujemo datotečni sistem itd. katerekoli trenutno aktivne Android naprave ali emulatorja. V tem pogledu razvojnega okolja Eclipse (DDMS pogled je na voljo samo, če uporabljamo vtičnik ADT) lahko preprosto zajemamo zaslonske slike in dostopamo do zapisov dnevnikov (angl. *logs*) Android naprave ali emulatorja, ki so generirani s pomočjo orodja LogCat. LogCat omogoča pregled zapisov dnevnikov ter filtriranje zapisov po resnosti, informacijah, opozorilih, napakah itd. Slika 21 nam prikazuje DDMS perspektivo razvojnega okolja Eclipse, prav tako pa so spodaj na tej sliki vidni nekateri zapisi dnevnikov, ki jih prikazuje LogCat;
- **AAPT** – namenjen izdelavi arhivske `.apk` datoteke, ki predstavlja celotno aplikacijo;
- **ADB** – je aplikacija tipa odjemalec-strežnik (angl. *client-server*), ki nam omogoča, da se povežemo z Android napravo ali emulatorjem. Kot komunikacijski kanal med strojno opremo razvojnega okolja in Android emulatorjem/napravo, nam ADB omogoča namestitev aplikacij, prejemanje in pošiljanje datotek ter izvajanje ukazov na ciljni napravi. Prav tako pa nam omogoča opravljanje poizvedb in urejanje katerekoli SQLite podatkovne baze na ciljni napravi. Vtičnik ADT avtomatizira in poenostavi

komunikacijo z ADB, vključno z nameščanjem in posodabljanjem aplikacij ter prenašanjem datotek;

- **DX** – omogoča pretvarjanje javanskih razrednih datotek v izvršljivo `.dex` datoteko, ki se nato izvaja na DVM;

Slika 20: Emulator z operacijskim sistemom Android 2.3.



Slika 21: DDMS perspektiva v razvojnem okolju Eclipse.

2.2.4 Eclipse

Eclipse IDE [24] je odprtokodna in brezplačna zbirka razvojih programskih orodij, ki nam zelo olajšajo razvoj programske opreme skozi celoten življenjski cikel razvoja. Skoraj celotno razvojno okolje je razvito v programskem jeziku Java. Eclipse je v prvi vrsti namenjen razvoju programske opreme v programskem jeziku Java, sestavljajo pa ga platforma Eclipse, javanska razvojna orodja in razvojno okolje nameščenih vtičnikov. Platforma Eclipse sicer omogoča razvojno programsko okolje za več programskih jezikov, sestavljena pa je iz integriranega razvojnega okolja ter razširljivega sistema vtičnikov. Prav funkcija nameščanja vtičnikov nam omogoča večjezični razvoj programske opreme, kar pomeni, da poleg razvoja programske opreme v Javi, lahko programsko opremo razvijamo tudi v programskih jezikih Ada, C, C++, Cobol, Python, PHP itd. Razvojno okolje (poleg ostalih) sestavljajo razvojna orodja za programski jezik Java (JDT), C/C++ (CDT) in PHP (PDT). Pri razvoju mobilne aplikacije AvtoDroid smo uporabljali verzijo Eclipse 3.6 Helios (zadnja izdana verzija je sicer Eclipse 3.7 Indigo), ki smo jo brezplačno prenesli in namestili na našem osebnem računalniku. Naj na tem mestu omenimo še, da lahko Eclipse namestimo na različnih operacijskih sistemih (Linux, Windows itd.), kar je zagotovo velika prednost v primerjavi z drugimi razvojnimi okolji, ki tega ne omogočajo.

2.2.5 Vtičnik ADT

ADT [4, 10] je vtičnik za Eclipse, ki zagotavlja velik nabor razvojnih orodij, ki so združena znotraj Eclipse IDE. Ponuja dostop do številnih funkcionalnosti, ki nam olajšajo razvijanje Android aplikacij. Glavna prednost ADT je zagotovo zagotavljanje grafičnega uporabniškega vmesnika za večino Android SDK orodij (ki bi jih drugače morali uporabljati preko ukazne vrstice) in orodij za izdelavo grafičnih uporabniških vmesnikov aplikacije, ki jo razvijamo. Najpomembnejše funkcionalnosti vmesnika so:

- **ustvarjanje in gradnja Android projektov, pakiranje, nameščanje in razhroščevanje** (angl. *integrated Android project creation, building, packaging, installation, and debugging*) – ADT vključuje številna orodja za opravljanje delovnih nalog za hiter in učinkovit razvoj ter testiranje aplikacij;
- **integracija z Android SDK orodji** (angl. *SDK tools integration*) – večina SDK orodij z namestitvijo ADT postane del integriranega razvojnega okolja Eclipse (najdemo jih v različnih menijih, perspektivah itd.);
- **urejevalnik kode programskega jezika Java in urejevalnik XML** (angl. *Java programming language and XML editors*) – urejevalnik kode programskega jezika Java omogoča sprotno preverjanje sintakse programske kode, samodejno dokončevanje ukaznih stavkov pri kodiranju itd., XML urejevalnik pa poleg osnovnih

funkcij omogoča tudi urejanje XML datotek s pomočjo grafičnega uporabniškega vmesnika, ki deluje na principu povleci in spusti (angl. *drag and drop*);

- **integrirana dokumentacija za Android API** (angl. *integrated documentation for Android framework APIs*) – do prikaza dokumentacije razreda ali metode preprosto pridemo tako, da z miško označimo razred ali metodo, ki nas zanima. Odpre se nam podroben opis lastnosti in funkcij izbranega razreda ali metode;
- **izvoz podpisanih (ali nepodpisanih) `.apk` datotek** (angl. *export signed (or unsigned) `.apk` files*) – omogoča nameščanje in distribucijo (npr. preko Android Market) aplikacij, ki smo jih razvili.

3 Razvoj mobilne aplikacije AvtoDroid

3.1 Metodologija

V tem poglavju bomo podrobno predstavili primer razvoja mobilne aplikacije na platformi Android. Aplikacijo smo razvili na podlagi kaskadnega modela razvoja programske opreme, katerega navodila smo poizkušali upoštevati čim bolj natančno. Tako bodo predstavljene vse faze razvoja od analize in specifikacije zahtev, načrtovanja, implementacije pa vse do testiranja aplikacije. Analizo razvoja ter samo delovanje oziroma funkcionalnosti aplikacije bomo podrobneje predstavili v poglavju 4. Še pred samim začetkom razvoja aplikacije smo namestili vsa potrebna orodja, opisana v poglavju 2.2. Za razvoj Android aplikacije smo se odločili predvsem zato, ker nas zanima programiranje v programskem jeziku Java (imamo že nekaj predhodnih izkušenj) ter tudi zato, ker sami uporabljamo pametni mobilni telefon, ki ima nameščen operacijski sistem Android, za katerega pa ni na voljo nekaterih aplikacij, ki jih potrebujemo. Ker smo imeli z razvojem Android aplikacij zelo malo predhodnih izkušenj, smo še pred samimi planiranjem in načrtovanjem razvoja aplikacije poiskali literaturo, v kateri je opisano programiranje Android aplikacij (literature na to temo je sicer zelo veliko). Ko smo se seznanili z osnovnimi temelji operacijskega sistema Android ter razvoja Android aplikacij, smo naredili nekaj osnovnih primerov aplikacij, nato pa smo začeli lotiti razvoja mobilne aplikacije.

3.2 Ideja

Idejo za razvoj mobilne aplikacije AvtoDroid smo dobili na podlagi spletne aplikacije Poraba.com [13]. Za uporabo aplikacije se moramo najprej registrirati, tako da vpišemo želeno uporabniško ime, geslo, naš elektronski naslov ter sprejmemo pogoje uporabe. Na naš elektronski naslov nato prejmemo elektronsko sporočilo, kjer moramo registracijo potrditi. Ko potrdimo registracijo, lahko začnemo z uporabo aplikacije. Na glavni strani lahko naprej opazimo novice, ki se nanašajo predvsem na spremembe cen goriv. Če želimo začeti uporabljati funkcionalnosti aplikacije, moramo najprej dodati avtomobil, katerega uporabljamo oziroma za katerega želimo uporabljati funkcionalnosti te aplikacije (lahko dodamo tudi več avtomobilov). Na sliki 22 lahko vidimo glavno stran aplikacije po prijavi ter dodajanju avtomobila. Za vsak avtomobil posebej lahko spremljamo porabo goriva (omogočen nam je vpis nove porabe goriva ter pregled preteklih zapisov o porabi goriva) ter ostale stroške, ki so povezani z avtomobilom (omogočen nam je vpis novega stroška ter pregled preteklih zapisov o stroških). Za vsako točenje goriva nam sistem avtomatsko izračuna tudi povprečno poraba goriva, vendar samo v primeru, ko smo natočili poln rezervoar goriva (izračun povprečne porabe namreč deluje tako, da najprej natočimo poln rezervoar goriva, se potem vozimo, nato spet natočimo poln rezervoar goriva ter nato število

natočenih litrov pri zadnjem točenju delimo s številom prevoženih kilometrov med obema točenjema goriv in pomnožimo s 100). Prav tako lahko pregledujemo razne statistične podatke o porabi goriva ter ostalih stroških, letna poročila, omogočen pa je tudi izpis zapisov v `.pdf` ali `.csv` datoteko. Na voljo so sicer še nekatere druge funkcionalnosti, ki pa jih ne bomo opisali, saj za nas niso aktualne. Za uporabo na mobilnih napravah obstaja tudi spletna mobilna aplikacija Poraba.com [14], ki pa ima zelo okrnjen nabor funkcionalnosti, saj omogoča samo vpis porabe goriva ter pregled preteklih zapisov o porabi goriva.

Slika 22: Glavna stran spletne aplikacije Poraba.com

Idejo za razvoj mobilne aplikacije AvtoDroid smo torej dobili na podlagi spletne aplikacije Poraba.com, ki jo uporabljamo že kar nekaj časa. Pri uporabi te aplikacije nas moti predvsem dejstvo, da je za uporabo potrebna povezava z internetom, ki sicer danes zaradi razvoja mobilnih brezžičnih povezav tudi na mobilnih napravah ne predstavlja večjih težav. Kljub temu pa ta aplikacija deluje samo v primeru, če je zagotovljena povezava z internetom, kar pa je neugodno predvsem pri uporabi aplikacije v tujini (visoke cene prenosa podatkov). Resda lahko aplikacijo uporabljamo tudi doma na osebnem računalniku, kjer imamo zagotovljeno neprekinjeno internetno povezavo, vendar je aplikacijo veliko bolj praktično uporabljati na mobilni napravi, kjer si vsako porabo goriva oziroma ostale stroške lahko takoj zapišemo (drugače se lahko zgodi, da to pozabimo narediti). Pri spletni aplikaciji Poraba.com pa nas nekoliko moti tudi način izračuna povprečne porabe goriva. Če nikoli ne natočimo polnega rezervoarja goriva, ne moremo spremljati povprečne porabe goriva. Ta dva moteča dejavnika smo pri razvoju mobilne aplikacije AvtoDroid odpravili.

3.3 Analiza in specifikacije zahtev aplikacije

3.3.1 Zahteve oziroma želene funkcionalnosti aplikacije

Želimo razviti mobilno aplikacijo AvtoDroid, ki naj omogoča podobne funkcionalnosti ter način uporabe kot spletna aplikacija Poraba.com. Aplikacijo naj bo možno uporabljati na mobilnih napravah, ki imajo nameščen operacijski sistem Android (glavni cilj je, da deluje na čim nižji verziji, da bo aplikacijo lahko uporabljalo čim večje število uporabnikov). Pri načrtovanju aplikacije naj bodo upoštevane vse omejitve mobilnih naprav, tako da naj bodo funkcionalnosti ustrezno omejene oziroma prilagojene uporabi na mobilnih napravah. Za uporabo aplikacije naj ne bo potrebna registracija uporabnika, saj bo nameščena lokalno, torej bo v večini primerov aplikacijo uporabljala ena sama oseba. Aplikacija naj za svojo delovanje ne zahteva povezave z internetom, deluje naj povsem lokalno. Vsi podatki, ki jih aplikacija potrebuje za delovanje naj bodo shranjeni lokalno v lokalni bazi podatkov. Mobilna aplikacija AvtoDroid naj omogoča spremljanje porabe goriva ter ostalih stroškov povezanih z avtomobilom (omogočen naj bo vnos, sprememba ter izbris zapisov). Izračun povprečne porabe goriva naj ne zahteva točenja polnega rezervoarja goriva. Na voljo naj bo tudi izvoz vseh zapisov na zunanji pomnilnik telefona (npr. na SD kartico) v `.xml` ali `.txt` formatu, možen pa naj bo tudi pregled osnovnih statističnih podatkov dosedanjih zapisov (podrobnejše statistične analize bo mogoče narediti s pomočjo izvoženih zapisov). Dosedanje zapise o porabi goriva, ostalih stroških ter statistične podatke naj bo možno pregledovati glede na izbrano obdobje. Prav tako naj na podoben princip deluje tudi izvoz zapisov. Vse te funkcionalnosti želimo uporabljati za enega ali več avtomobilov (možno naj bo tako dodajanje kot odstranjevanje avtomobilov), pri čemer za vsakega od avtomobilov vodimo tudi podrobne podatke o vozilu (obvezen naj bo vnos vsaj osnovnih podatkov). Za vsak avtomobil

naj bo omogočeno opozarjanje na bližajočo se registracijo ali servis avtomobila. Na voljo naj bodo tudi navodila oziroma pomoč pri uporabi aplikacije, tako da bodo uporabniki znali uporabljati vse funkcionalnosti aplikacije. Na svojo željo pa lahko uporabniki preko svetovnega spleta pridobijo podatke o trenutnih cenah vseh pogonskih goriv (to ne bo omogočeno avtomatsko, ampak samo na zahtevo uporabnika). Uporabniški vmesnik naj bo prijazen do uporabnika ter preprost za uporabo, omogoča naj opozarjanje na napačno uporabo funkcionalnosti (preko opozorilnih oken), za vsako uspešno zaključeno uporabo funkcionalnosti (npr. uspešno vnesen zapis o porabi goriva, uspešno izbrisan avtomobil itd.) pa naj bo uporabnik ustrezno obveščen.

3.3.2 Specifikacije zahtev aplikacije

Glede na opisane zahteve oziroma želene funkcionalnosti aplikacije smo naredili specifikacije zahtev aplikacije:

- aplikacijo bo možno uporabljati na operacijskem sistemu Android, in sicer od verzije 2.3 dalje, pri čemer bodo v aplikaciji upoštevane vse omejitve mobilnih naprav. Optimizirana bo za uporabo na pametnih mobilnih telefonih, možna pa bo uporaba tudi na tabličnih računalnikih;
- za hrambo podatkov oziroma zapisov se bo uporabljala SQLite lokalna baza podatkov;
- aplikacijo bo uporabnik lahko prenesel in namestil brezplačno. Med samo namestitvijo bo moral uporabnik aplikaciji dovoliti dostop do zunanjega pomnilnika, interneta ter uporabo vibriranja. Aplikacija bo sicer delovala popolnoma lokalno, torej povezava z internetom ne bo nujno potrebna. Ko bo uporabnik aplikacijo uspešno namestil, jo bo lahko začel uporabljati, ne da bi se moral registrirati. Še preden bo pa lahko začel uporabljati same funkcionalnosti aplikacije, pa bo moral dodati nov avtomobil (drugače ne bo možen prehod na glavno stran, kjer se sama uporaba funkcionalnosti aplikacije začne). Pri dodajanju avtomobila bo obvezen vnos nekaterih osnovnih podatkov o vozilu (naziv vozila, lastnik, datum registracije, datum zadnjega servisa, število prevoženih kilometrov pri zadnjem servisu, servisni interval ter trenutno število prevoženih kilometrov), medtem ko vnos ostalih podatkov o vozilu (številka prometnega dovoljenja, registrska številka, letnik, barva, moč v kw in ccm in številka šasije) ne bo obvezen;
- preko tipke `menu` na mobilni napravi bo uporabniku na voljo dostop do pomoči za uporabo neke funkcionalnosti. Vsebina pomoči bo prilagojena delu aplikacije, kjer bo uporabnik zahteval pomoč (npr. pri vpisu nove porabe goriva se bo vsebina pomoči nanašala samo na vpis nove porabe goriva itd.). Preko iste tipke bo na voljo tudi dostop do nastavitev aplikacije (v vseh delih aplikacije bodo nastavitve enake), kjer bo uporabnik lahko nastavil število dni oziroma število kilometrov, preden ga bo

aplikacija začela opozarjati na servis ali registracijo avtomobila, izbral katere vrste goriv uporablja pri trenutno izbranem avtomobilu (avtoplin, diesel, 95-okt. in 100-okt.), prav tako pa bo tu možno pridobiti podatke o cenah vseh pogonskih goriv preko svetovnega spleta. Preko menija pa bo omogočena tudi vrnitev na glavno stran aplikacije in izhod iz aplikacije iz kateregakoli dela aplikacije;

- uporabnik bo lahko izbiral (v primeru, da bo imel dodanih več avtomobilov) avtomobil, za katerega trenutno želi uporabljati funkcionalnosti aplikacije. Prav tako bo lahko dodal nov avtomobil, pri čemer bo postopek dodajanja identičen dodajanju prvega avtomobila (preden lahko prične uporabljati funkcionalnosti aplikacije), ki je bil že opisan;
- uporabnik bo vse podatke o trenutnem izbranem vozilu lahko poljubno pregledoval (pri čemer bo imel na voljo tudi podatke o tem, koliko zanaša trenutna povprečna poraba goriva in koliko časa oziroma kilometrov je do naslednjega servisa ali registracije) in spreminjal, vozilo pa bo lahko tudi izbrisal (vendar samo v primeru, da ima trenutno dodanih več vozil, saj uporaba funkcionalnosti aplikacije brez dodanega vozila ni možna);
- za trenutno izbran avtomobil bo možen vpis nove porabe goriva ter pregled in izbris obstoječih zapisov o porabi. Pri vpisu porabe bosta na volja dva različna uporabniška vmesnika. Pri prvem bo potreben vpis podatkov o datumu točenja goriva, kje in katero vrsto goriva smo točili (možna bo samo izbira goriv, ki jih je uporabnik izbral v nastavitvah, prav tako pa bo možna samo izbira bencinskih črpalk, ki so vpisane v bazi podatkov: OMV, Agip, Petrol, MOL, Shell, tujina in ostalo), koliko litrov goriva smo natočili, znesek točenja ter stanje kilometrov na števcu, ko smo točili gorivo (stanje kilometrov bo moralo biti večje ali enako končnemu stanju kilometrov zadnjega zapisa o porabi goriva). Pri drugem uporabniškem vmesniku (ki v bistvu služi za dokončanje zapisa) pa bo potreben le vpis končnega stanja kilometrov (oz. začetno stanje kilometrov pri naslednjem točenju), pri čemer mora biti končno stanje kilometrov večje začetnemu stanju kilometrov tega zapisa o porabi goriva. Potrebna bo tudi izbira, ali je bil rezervoar prazen ali ne. Vpis nove porabe goriva v primeru, da nek zapis še ni bil dokončan, ne bo mogoč. Naj tu omenimo še, da se bo podatek o stanju prevoženih kilometrov trenutno izbranega avtomobila avtomatsko posodabljal glede na vpisano začetno ali končno stanje kilometrov pri vpisu porabe goriva. V primeru, da je bil rezervoar prazen, se bo lahko izračunala povprečna poraba goriva tega zapisa (pri tem velja omeniti, da je najbolje, da imamo pri prvem vpisu porabe goriva prazen rezervoar), v nasprotnem pa bo zapis upoštevan pri naslednjem možnem izračunu povprečne porabe goriva. Izračun povprečne porabe goriva deluje po naslednjem principu: ko imamo rezervoar prazen, natočimo določeno število litrov goriva. Nato prevozimo določeno število kilometrov in ko je rezervoar prazen (najbolje se je

orientirati po lučki, ki zasveti takrat, ko je potrebno natočiti gorivo, ali pa po potovalnem računalniku) lahko izračunamo povprečno porabo goriva po naslednji formuli:

$$\text{povprečna poraba goriva} = \frac{\text{število porabljenih litrov goriva} * 100}{\text{število prevoženih kilometrov}}$$

Po nekaj vpisih porabe goriva bomo dobili dokaj natančno povprečno porabo goriva našega avtomobila. Kot smo že omenili pa bo možen tudi pregled obstoječih zapisov o porabi goriva, katere bomo poleg pregledovanja lahko tudi spreminjali (tu veljajo enake omejitve vpisa podatkov kot pri vpisu porabe goriva) ali brisali. Možen bo pregled zapisov o porabi goriva za izbrano leto ali pa za vsa leta skupaj. Na isti princip bo deloval tudi izvoz zapisov o porabi goriva na zunanji pomnilnik naprave v formatu `.txt` ali `.xml`;

- za trenutno izbran avtomobil bo možen vpis ostalih stroškov, povezanih z avtomobilom. Uporabnik bo moral vpisati naziv stroška, kategorijo stroška (na voljo bodo kategorije, shranjene v lokalni bazi podatkov: servis, registracija, zavarovanje, obvezna oprema, ostala oprema, kazni in ostalo), znesek stroška, datum ter podroben opis stroška. Če bo izbrana kategorija servis, se bosta v primeru, da je šlo za redni servis, podatka o datumu ter številu prevoženih kilometrov pri zadnjem servisu vozila avtomatsko posodobila. Možen bo pregled obstoječih zapisov o ostalih stroških, katere bomo poleg pregledovanja lahko tudi spreminjali ali brisali. Pregled zapisov o stroških vozila bo možen za izbrano leto ali pa za vsa leta skupaj. Na isti princip bo deloval tudi izvoz zapisov o ostalih stroških na zunanji pomnilnik naprave v formatu `.txt` ali `.xml`;
- za trenutno izbran avtomobil bo možen tudi pregled nekaterih osnovnih statističnih podatkov o porabi goriva (število zapisov, skupni znesek, skupno natočenih litrov goriva, skupno prevoženih kilometrov, minimalna ter maksimalna povprečna poraba goriva, skupna povprečna poraba goriva ter število točenj goriva glede na posamezne bencinske črpalke in vrsto goriva) ter ostalih stroških (skupni znesek glede na posamezno kategorijo in skupni znesek vseh kategorij skupaj). Tudi tukaj bo možen pregled zapisov glede na izbrano leto ali pa za vsa leta skupaj;
- izgled uporabniškega vmesnika bo v vseh delih aplikacije enak (uporabljene bodo črna, siva in bela barva), prav tako pa bo v vseh delih aplikacije zelo podobna tudi sama uporaba vmesnikov;
- v primeru, ko bo uporabnik napačno uporabljal katerokoli od funkcionalnosti oziroma ko vpisani podatki ne bodo ustrezni, bo uporabnik na to opozorjen preko okna. Prav

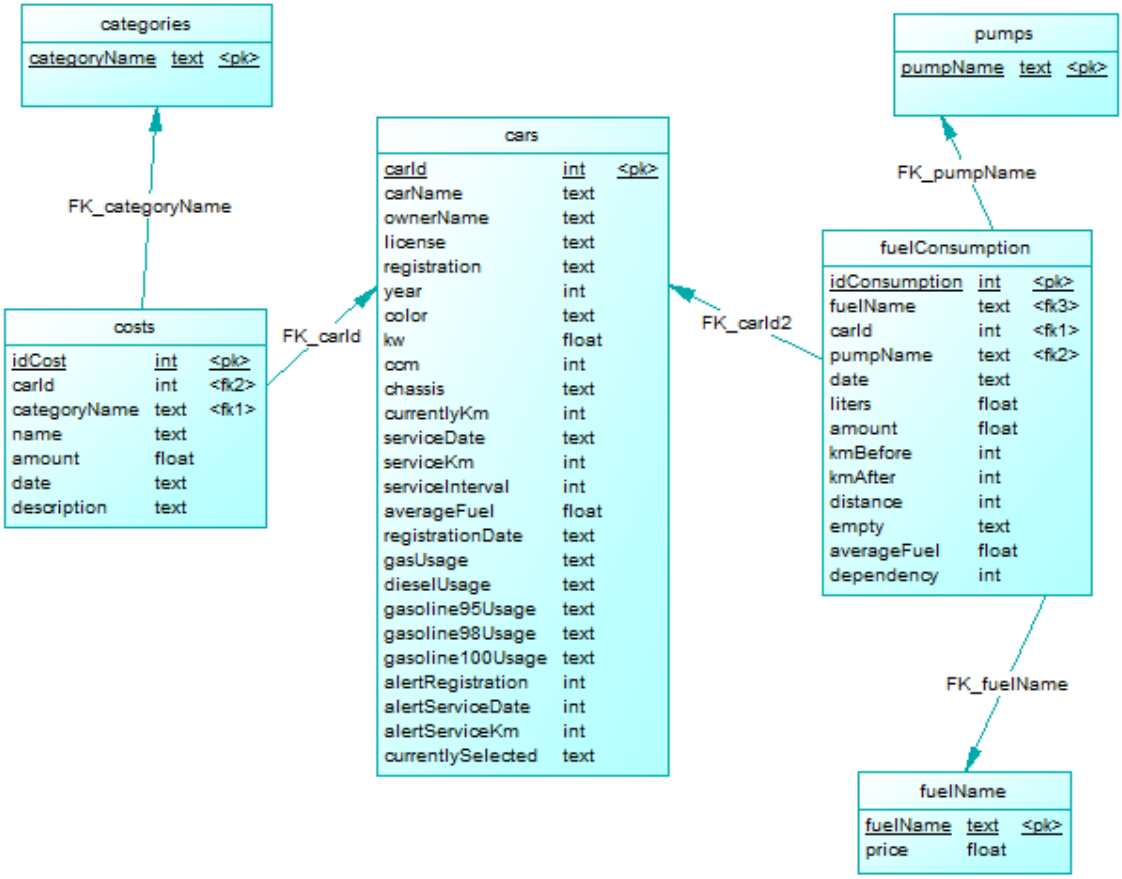
tako bo uporabnik preko okna obveščen o vsaki uspešno zaključeni uporabi katerekoli od funkcionalnosti aplikacije.

Določili smo specifikacije zahtev in sedaj se lahko lotimo načrtovanja aplikacije.

3.4 Načrtovanje aplikacije

3.4.1 Načrtovanje podatkovne baze

Najprej smo na podlagi specifikacije zahtev aplikacije naredili podatkovni načrt, ki jo bo aplikacija uporabljala. Podatkovna baza se bo imenovala `androiddatabase`, glede na zahteve pa bo obsegala 6 podatkovnih tabel (`CATEGORIES`, `CARS`, `PUMPS`, `FUELS`, `FUELCONSMUPTION`, `COSTS`). S pomočjo orodja PowerDesigner smo naredili logični podatkovni model (slika 23), kjer so vidne vse tabele ter polja posameznih tabel.



Slika 23: Logični podatkovni model.

Tabela CARS

V tej tabeli bodo shranjeni vsi avtomobili, za katere bo uporabnik želel uporabljati funkcionalnosti aplikacije (spomnimo se, da uporabnik lahko doda enega ali več avtomobilov). Prav tako bodo v tej tabeli shranjene nastavitve aplikacije, ki so povezane s samim avtomobilom. Tabela CARS vsebuje sledeča polja:

- **carId** – identifikacijska številka avtomobila (INTEGER, primarni ključ, obvezen);
- **carName** – naziv avtomobila (TEXT, obvezen);
- **ownerName** – ime lastnika avtomobila (TEXT, obvezen);
- **license** – številka prometnega dovoljenja avtomobila (TEXT);
- **registration** – registrska številka avtomobila (TEXT);
- **year** – letnik izdelave avtomobila (INTEGER);
- **color** – barva avtomobila (TEXT);
- **kw** – moč avtomobila v kilovatih (REAL);
- **ccm** – prostornina motorja avtomobila (INTEGER);
- **chassis** – številka šasije avtomobila (TEXT);
- **currentlyKm** – trenutno število prevoženih kilometrov (INTEGER, obvezen);
- **serviceDate** – datum zadnjega servisa avtomobila (TEXT, obvezen);
- **serviceKm** – število prevoženih kilometrov pri zadnjem servisu (INTEGER, obvezen);
- **serviceInterval** – servisni interval avtomobila glede na število prevoženih kilometrov (INTEGER, obvezen);
- **averageFuel** – avtomatsko izračunana povprečna poraba goriva avtomobila, izračunana je na podlagi zapisov o porabi goriva (REAL);
- **registrationDate** – datum zadnje registracije avtomobila (TEXT, obvezen);
- **gasUsage** – ali avtomobil uporablja avtoplin (TEXT, obvezen);
- **dieselUsage** – ali avtomobil uporablja dizelsko gorivo (TEXT, obvezen);
- **gasoline95Usage** – ali avtomobil uporablja 95 oktanski bencin (TEXT, obvezen);
- **gasoline98Usage** – ali avtomobil uporablja 98 oktanski bencin (TEXT, obvezen);
- **gasoline100Usage** – ali avtomobil uporablja 100 oktanski bencin (TEXT, obvezen);
- **alertRegistration** – število dni, preden nas aplikacija za izbran avtomobil prične opozarjati na registracijo (INTEGER, obvezen);
- **alertServiceDate** – število dni, preden nas aplikacija za izbran avtomobil prične opozarjati na servis (INTEGER, obvezen);

- **alertServiceKm** – število kilometrov, preden nas aplikacija za izbran avtomobil prične opozarjati na servis (INTEGER, obvezen);
- **currentlySelected** – ali je avtomobil v aplikaciji trenutno izbran avtomobil (TEXT, obvezen).

Tabela CATEGORIES

V tej tabeli bodo shranjeni nazivi kategorij, ki jih uporabljamo pri vnosu novega stroška (izbira kategorije stroška). Tabela CATEGORIES vsebuje sledeča polja:

- **categoryName** – naziv kategorije (TEXT, primarni ključ, obvezen).

Tabela PUMPS

V tej tabeli bodo shranjeni nazivi vseh bencinskih črpalk, ki jih uporabljamo pri vpisu nove porabe goriva (izbira bencinske črpalke). Tabela PUMPS vsebuje sledeča polja:

- **pumpName** – ime bencinske črpalke (TEXT, primarni ključ, obvezen).

Tabela FUELS

V tej tabeli bodo shranjena vsa goriva, ki jih lahko najdemo na bencinskih črpalkah po Sloveniji. Podatki iz te tabele se uporabljajo pri vpisu nove porabe goriva (izbira vrste goriva). Tabela FUELS vsebuje sledeča polja:

- **fuelName** – naziv goriva (TEXT, primarni ključ, obvezen);
- **price** – cena goriva, ki jo lahko pridobimo iz svetovnega spleta (REAL, obvezen);

Tabela COSTS

V tej tabeli bodo shranjeni vsi zapisi o ostalih stroških (vseh dodanih avtomobilov). Tabela COSTS vsebuje sledeča polja:

- **idCost** – identifikacijska številka zapisa (INTEGER, primarni ključ, obvezen);
- **carId** - identifikacijska številka avtomobila na katerega se zapis nanaša (INTEGER, tuj ključ, obvezen);
- **categoryName** – naziv kategorije stroška (TEXT, tuj ključ, obvezen);
- **name** – naziv stroška (TEXT, obvezen);
- **amount** – znesek stroška (REAL, obvezen);
- **dateOfCost** – datum stroška (TEXT, obvezen);

- **description** – opis stroška (TEXT, obvezen).

Tabela FUELCONSUMPTION

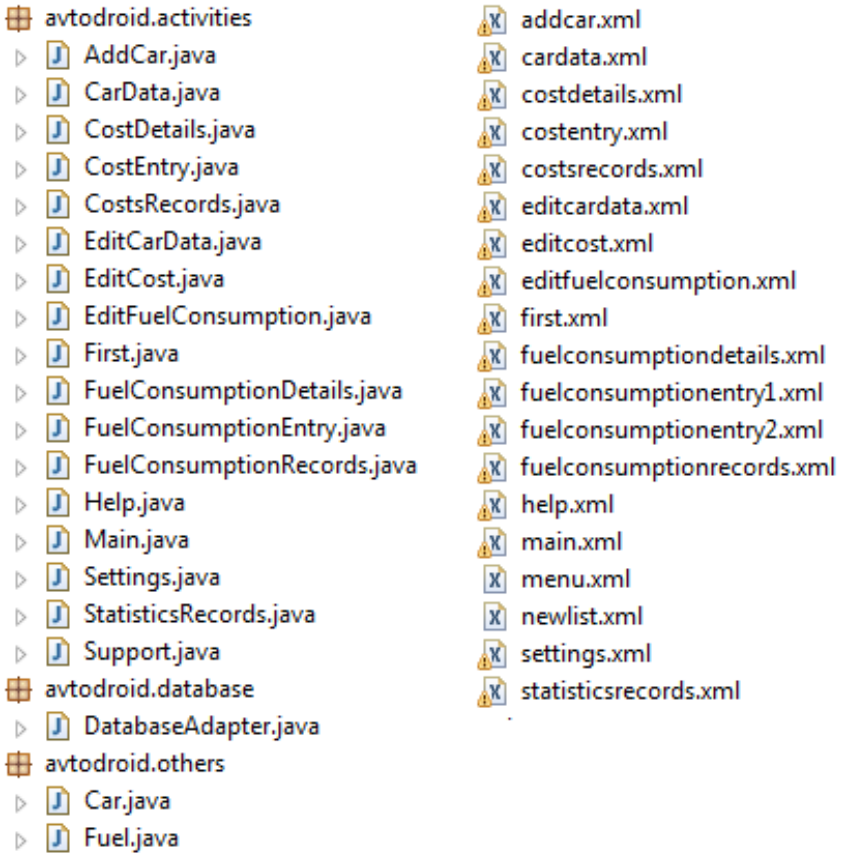
V tej tabeli bodo shranjeni vsi zapisi o porabi goriva (vseh dodanih avtomobilov). Tabela FUELCONSUMPTION vsebuje sledeča polja:

- **idConsumption** - identifikacijska številka zapisa (INTEGER, primarni ključ, obvezen);
- **fuelName** – naziv goriva v zapisu (TEXT, tuj ključ, obvezen);
- **carId** - identifikacijska številka avtomobila na katerega se zapis nanaša (INTEGER, tuj ključ, obvezen);
- **pumpName** – naziv bencinske črpalke v zapisu (TEXT, tuj ključ, obvezen);
- **dateOfConsumption** – datum točenja goriva (TEXT, obvezen);
- **liters** – število natočenih litrov (REAL, obvezen);
- **amount** – znesek točenja goriva (REAL, obvezen);
- **kmBefore** – stanje kilometrov pred točenjem goriva (INTEGER, obvezen);
- **kmAfter** – stanje kilometrov ob naslednjem točenju goriva (INTEGER);
- **distance** – število prevoženih kilometrov v tem zapisu (INTEGER);
- **empty** – ali smo porabili vso natočeno gorivo (TEXT);
- **averageFuel** – kolikšna je bila povprečna poraba goriva v tem zapisu (REAL);
- **dependency** – ali je povprečna poraba goriva izračunana samo iz tega zapisa ali pa je del izračune povprečne porabe kakšnega drugega zapisa. Vrednost v tem polju predstavlja identifikacijsko številko zapisa porabe goriva, od katerega je odvisna izračunana povprečna poraba goriva (INTEGER).

3.4.2 Načrtovanje posameznih delov aplikacije ter predstavitev aplikacije

Ko smo končali z načrtovanjem podatkovne baze za našo aplikacijo, smo se lotili načrtovanja posameznih delov aplikacije. Glede na specifikacije zahtev smo prišli do ugotovitve, da bomo rabili kar nekaj različnih aktivnosti (ter pripadajočih .xml datotek, ki predstavljajo uporabniške vmesnike), pa tudi nekaj ostalih (pomožnih) javanskih razredov (npr. za delo s podatkovno bazo itd.). Seznam vseh potrebnih paketov in razredov ter pripadajočih uporabniških vmesnikov aktivnosti je prikazan na sliki 24. Sledi opis načrtovanega delovanja aktivnosti (s pripadajočimi uporabniškimi vmesniki) ter ostalih (pomožnih) javanski razredov. Opis načrtovanega delovanja aplikacije bomo v tem poglavju združili s predstavitvijo (že delujoče končne verzije aplikacije, saj predstavitvi funkcionalnosti oziroma tipičnih primerov

uporabe aplikacije ne bomo posvetili posebnega poglavja). Pri vsaki izmed aktivnosti bomo omenili tudi na katerih podatkovnih tabel se izvajajo SQL poizvedbe.



Slika 24: Seznam razredov ter pripadajočih uporabniških vmesnikov aktivnosti.

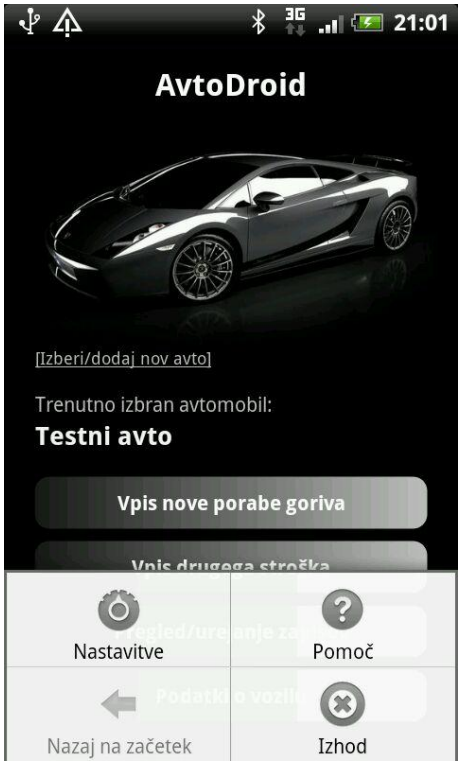
Osnovni meni aplikacije

Še pred samimi opisom razredov naše aplikacije bomo predstavili načrtovane funkcionalnosti osnovnega menija aplikacije. Do osnovnega menija aplikacije (slika 25) lahko pridemo s pritiskom gumba `menu`, ki je del vsake Android naprave. Preko menija lahko iz katerega koli dela aplikacije (torej preko vseh aktivnosti) pridemo do nastavitev aplikacije, in prikaza pomoči, katere vsebina je prilagojena delu aplikacije, kjer pomoč zahtevamo. Prav tako pa je preko menija možna vrnitev na glavno stran aplikacije ter izhod iz aplikacije. Osnovni meni bo sicer bolj podrobno predstavljen v poglavju 3.5.2.

Paket `avtodroid.database`

Paket `avtodroid.database` vsebuje samo en pomožen razred, imenovan `DatabaseAdapter`. V tem razredu so zbrane vse metode, ki nam služijo za delo s podatkovno bazo (vstavljanje, spreminjanje in brisanje podatkov v podatkovnih tabelah ter

samo odpiranje ter zapiranje podatkovne baze). Razred razširja (angl. *extends*) razred `SQLiteOpenHelper.java`, ki služi kot pomoč pri delu s podatkovno bazo.



Slika 25: Osnovni meni aplikacije.

Paket `avtodroid.others`

Tudi v tem paketu se nahajajo samo pomožni razredi (torej so nam samo v pomoč pri delovanju aplikacije). Paket vsebuje naslednje razrede:

- **Car.java –** ta razred nam služi za shranjevanje vseh dodanih avtomobilov uporabnika aplikacije v podatkovni zbirki `ArrayList<Car>`. Tako namesto večkratnega branja uporabnikovih avtomobilov iz podatkovne tabele `CARS`, to storimo samo enkrat;
- **Fuel.java** – ta razred vsebuje metodo za pridobivanje cen vseh pogonskih goriv preko svetovnega spleta [15].

Paket `avtodroid.activities`

Kot lahko razberemo že iz imena paketa so tu shranjene vse aktivnosti naše aplikacije. Ker aplikacija AvtoDroid obsega kar nekaj funkcionalnosti, je temu primerno potrebnih veliko število različnih aktivnosti (ter pripadajočih uporabniških vmesnikov). Primer načrtovanja enega uporabniškega vmesnika aktivnosti bo predstavljen v poglavju 3.4.3. V tem poglavju pa bomo predstavili načrt delovanja posameznih aktivnosti (skupaj z zaslonskimi slikami

pripadajočih uporabniških vmesnikov). Vsi v nadaljevanju opisani razredi razširjajo razred `Activity` (razen razreda `Support`, ki razširja razred `Application`). Prav tako lahko na tem mestu omenimo tudi, da so vsa okna, ki se odpirajo skozi celotni življenjski cikel aplikacije, zgrajena s pomočjo razreda `AlertDialog`. Paket vsebuje naslednje razrede oziroma aktivnosti (ter pripadajoče uporabniške vmesnike):

- **`Support.java`** – je edini razred aplikacije, ki razširja razred `Application` (referenca do razreda se hrani skozi celoten življenjski cikel aplikacij in ne more biti nikoli uničena). Razred nam služi predvsem kot pomožen razred, kjer lahko shranjujemo trenutne lastnosti oziroma stanje aplikacije (npr. trenutno izbran avtomobil, izbran indeks zapisa porabe, ali je potrebno opozarjanje na servis in registracijo itd.). Razred prav tako vsebuje pomožne metode za izračun nekaterih podatkov (npr. čez koliko dni bo potreben servis in registracija itd.). Ko hočemo v katerikoli aktivnosti aplikacije dostopati do podatkov in metod, ki so shranjene v tem razredu, ustvarimo novo spremenljivko razreda ter ji vrnemo kontekst aplikacije (s pomočjo metode `getApplicationContext()`). V razredu se ne izvaja nobena SQL poizvedba na podatkovnih tabelah;
- **`First.java (first.xml)`** – ta aktivnost je prikazana samo pri prvi uporabi aplikacije (takoj po namestitvi). Zahtevan je vnos podatkov o uporabnikovem avtomobilu, pri čemer je za uspešno dodajanje avtomobila v podatkovno tabelo `CARS` potrebna obvezna izpolnitev nekaterih polj. V primeru, da katera od obveznih polj niso izpolnjena oziroma niso izpolnjena pravilno, bo aplikacija uporabnika preko okna na to tudi opozorila. Ko uporabnik pravilno izpolni vsa obvezna polja ter izbere gumb `Potrditev` v uporabniškem vmesniku, je preko okna obveščen o uspešnem vnosu novega zapisa v tabelo `CARS` ter preusmerjen na glavno stran aplikacije (`Main`). Kot že rečeno, se bo aktivnost pojavila samo v primeru, ko v tabeli `CARS` ne bo dodanega še nobenega avtomobila, nato pa aktivnost pri naslednjih uporabah aplikacije ne bo nikoli več prikazana. V razredu se izvajajo SQL poizvedbe na podatkovni tabeli `CARS`. Uporabniški vmesnik te aktivnosti lahko vidimo na sliki 26;
- **`Main.java (main.xml)`** – to je glavna aktivnost aplikacije, saj predstavljala glavno (navigacijsko) stran, kjer je možen dostop do vseh funkcionalnosti aplikacije. Uporabnik preko okna lahko izbere avtomobil (okno se odpre z izbiro povezave `Izberi/dodaj nov avto` v uporabniškem vmesniku), za katerega želi uporabljati funkcionalnosti aplikacije, preko istega okna pa lahko zažene tudi novo aktivnost, ki omogoča dodajanje novega avtomobila (`AddCar`). Uporabniški vmesnik sestavljajo tudi štirje gumbi, ki omogočajo:

Slika 26: Uporabniški vmesnik aktivnosti `First`.

- izbira gumba `Vpis nove porabe goriva` v uporabniškem vmesniku uporabnika preusmeri na aktivnost, ki omogoča vpis nove porabe goriva (`FuelConsumptionEntry`),
- izbira gumba `Vpis drugega stroška` v uporabniškem vmesniku uporabnika preusmeri na aktivnost, ki omogoča vpis novega stroška (`CostEntry`),
- izbira gumba `Pregled/urejanje zapisov` v uporabniškem vmesniku najprej odpre okno, kjer je potrebno izbrati, katere zapise želimo pregledovati (zapise o porabi, ostalih stroških ali statistične zapise). Nato se odpre še eno okno, kjer je potrebno izbrati, za katero obdobje želimo imeti prikazane zapise. Po izbiri želenega obdobja je uporabnik preusmerjen na aktivnost, kjer so prikazani izbrani zapisi (`FuelConsumptionRecords` ali `CostsRecords` ali `StatisticsRecords`),
- izbira gumba `Podatki o vozilu` v uporabniškem vmesniku uporabnika preusmeri na aktivnost, ki omogoča pregled vseh podatkov o trenutno izbranem vozilu (`CarData`).

V primeru, da bo kmalu potreben servis ali registracija trenutno izbranega avtomobila, se odpre okno, ki nas poleg izpisa opozorila o bližajočem se servisu ali registraciji na to opozori tudi z vibriranjem. Izbira gumba `nazaj` na Android napravi pomeni izhod

iz aplikacije. V razredu se izvajajo SQL poizvedbe na podatkovnih tabelah `CARS`, `COSTS` in `FUELCONSUMPTION`. Uporabniški vmesnik te aktivnosti lahko vidimo na sliki 27.

Slika 27: Uporabniški vmesnik aktivnosti `Main`.

- **`AddCar.java (addcar.xml)`** – ta aktivnost nam omogoča dodajanje novega avtomobila v podatkovno tabelo `CARS`. Dodajanje novega avtomobila je identično prvem dodajanju avtomobila (pri aktivnosti `First`). Ko uporabnik pravilno izpolni vsa obvezna polja ter izbere gumb `Potrdi` v uporabniškem vmesniku, je preko okna obveščen o uspešnem vnosu novega zapisa v tabelo `CARS` ter preusmerjen na glavno stran aplikacije (`Main`). V primeru izbire gumba `Prekliči` v uporabniškem vmesniku oziroma ob kliku na gumb `nazaj` na Android napravi, je uporabnik, potem ko v oknu potrdi svojo odločitev, prav tako preusmerjen na glavno stran aplikacije (`Main`). V razredu se izvajajo SQL poizvedbe na podatkovni tabeli `CARS`. Uporabniški vmesnik te aktivnosti lahko vidimo na sliki 28.
- **`FuelConsumptionEntry.java (fuelconsumptionentry1.xml, fuelconsumptionentry2.xml)`** – ta aktivnost ima dva različna uporabniška vmesnika. Prvi uporabniški vmesnik je prikazan ob novem vpisu porabe goriva, medtem ko je drugi uporabniški vmesnik prikazan takrat, ko je potrebno nek (še nezaključen) zapis zaključiti (vpis nove porabe

Slika 28: Uporabniški vmesnik `AddCar`.

goriva takrat ni možen). V primeru, da nekatera polja niso izpolnjena oziroma niso izpolnjena pravilno, bo aplikacija uporabnika preko okna na to tudi opozorila (pri obeh uporabniških vmesnikih). Ko uporabnik pravilno izpolni vsa polja pri prvem uporabniškem vmesniku ter izbere gumb `Potrdi`, je preko okna obveščen o uspešnem vnosu novega zapisa v tabelo `FUELCONSUMPTION` ter preusmerjen na glavno stran aplikacije (`Main`). Ko uporabnik pravilno izpolni vsa polja pri drugem uporabniškem vmesniku ter izbere gumb `Potrdi`, je preko okna obveščen o uspešen zaključku zapisa (oziroma posodobitvi tega zapisa v podatkovni tabeli `FUELCONSUMPTION`) ter preusmerjen na aktivnost, ki omogoča vpis nove porabe goriva (potrebna je osvežitev aktivnosti `FuelConsumptionEntry`). V obeh primerih sledi avtomatska posodobitev polja `currentlyKm` v tabeli `CARS` (vrednost je enaka zadnjemu vpisanemu stanju kilometrov). Pri obeh uporabniških vmesnikih pa je v primeru izbire gumba `Prekliči` v uporabniškem vmesniku oziroma ob kliku na gumb `nazaj` na Android mobilni napravi uporabnik, potem ko v oknu potrdi svojo odločitev, preusmerjen na glavno stran aplikacije (`Main`). V razredu se izvajajo SQL poizvedbe na podatkovnih tabelah `CARS`, `PUMPS`, `FUELS` in `FUELCONSUMPTION`. Uporabniška vmesnika te aktivnosti lahko vidimo na sliki 29;

Slika 29: Uporabniška vmesnika aktivnosti `FuelConsumptionEntry`.

- **`FuelConsumptionRecords.java (fuelconsumptionrecords.xml)`** – ta aktivnost nam omogoča prikaz zapisov o porabi goriva za neko izbrano obdobje. Izbira enega od zapisov uporabnika preusmeri na aktivnost, kjer so prikazani podrobni podatki o tem zapisu (`FuelConsumptionDetails`). Možen je tudi izvoz zapisov o porabi goriva v datoteko na zunanji pomnilnik. Ob izbiri gumba `Izvozi zapise` v uporabniškem vmesniku se najprej odpre okno, kjer je potrebno izbrati, katere zapise želimo izvoziti (vse zapise ali samo prikazane) ter format datoteke (`.txt` ali `.xml`). Nato se odpre še eno okno, kjer je potreben vpis imena datoteke. Temu sledi izvoz datoteke na zunanji pomnilnik (uporabnik je o uspešnem izvozu obveščen preko okna). Klik gumba `nazaj` na Android napravi nas preusmeri na glavno stran aplikacije (`Main`). V razredu se izvajajo SQL poizvedbe na podatkovnih tabelah `CARS` in `FUELCONSUMPTION`. Uporabniški vmesnik te aktivnosti lahko vidimo na sliki 30;
- **`FuelConsumptionDetails.java (fuelconsumptiondetails.xml)`** – ta aktivnost nam omogoča prikaz podrobnih podatkov o izbranem zapisu porabe goriva. Izbira gumba `Izbriši` v uporabniškem vmesniku omogoča uporabniku izbris izbranega zapisa iz tabele `FUELCONSUMPTION`, potem ko v oknu potrdi svojo odločitev o izbrisu. V primeru uspešnega izbrisa oziroma ob kliku na gumb `nazaj` na Android napravi je uporabnik preusmerjen na aktivnost, ki omogoča prikaz zapisov o porabi goriva za neko izbrano obdobje (`FuelConsumptionRecords`). Izbira

Slika 30: Uporabniški vmesnik aktivnosti `FuelConsumptionRecords`.

gumba `Uredi` v uporabniškem vmesniku pa uporabnika preusmeri na aktivnost, ki omogoča urejanje izbranega zapisa porabe goriva (`EditFuelConsumption`). V razredu se izvajajo SQL poizvedbe na podatkovnih tabelah `CARS` in `FUELCONSUMPTION`. Uporabniški vmesnik te aktivnosti lahko vidimo na sliki 31;

Slika 31: Uporabniški vmesnik aktivnosti `FuelConsumptionDetails`.

- **`EditFuelConsumption.java (editfuelconsumption.xml)`** – ta aktivnost nam omogoča urejanje podatkov o izbranem zapisu porabe goriva. V primeru, da nekatera polja niso izpolnjena oziroma niso izpolnjena pravilno, bo aplikacija uporabnika preko okna na to tudi opozorila. Ko uporabnik pravilno izpolni oziroma popravi vsa polja ter izbere gumb `Potrdi` v uporabniškem vmesniku, je preko okna obveščen o uspešni posodobitvi zapisa v tabeli `FUELCONSUMPTION` ter preusmerjen na aktivnost, ki omogoča prikaz podrobnih podatkov o izbranem zapisu porabe goriva (`FuelConsumptionDetails`). V primeru izbire gumba `Prekliči` v uporabniškem vmesniku oziroma ob kliku na gumb `nazaj` na Android napravi je uporabnik, potem ko v oknu potrdi svojo odločitev, prav tako preusmerjen na prej omenjeno aktivnost (`FuelConsumptionDetails`). V razredu se izvajajo SQL poizvedbe na podatkovnih tabelah `CARS`, `PUMPS`, `FUELS` in `FUELCONSUMPTION`. Uporabniški vmesnik te aktivnosti lahko vidimo na sliki 32;

Slika 32: Uporabniški vmesnik aktivnosti `EditFuelConsumption`.

- **`CostEntry.java (costentry.xml)`** – ta aktivnost nam omogoča vpis novega stroška, povezanega s trenutno izbranim avtomobilom, v podatkovno tabelo `COSTS`. V primeru, da katera od polj niso izpolnjena oziroma niso izpolnjena pravilno, bo aplikacija uporabnika okna na to tudi opozorila. Ko uporabnik pravilno izpolni vsa

polja ter izbere gumb `Potrdi` v uporabniškem vmesniku, je preko okna obveščen o uspešnem vnosu novega zapisa v tabelo `COSTS` ter preusmerjen na glavno stran aplikacije (`Main`). Če je bila izbrana kategorija servis, se odpre okno, kjer moramo v primeru, da je bil opravljen redni servis, vpisati število kilometrov, pri katerih je bil servis opravljen (temu sledi avtomatska posodobitev polj `serviceDate` in `serviceKm` v tabeli `CARS`). V primeru izbire gumba `Prekliči` v uporabniškem vmesniku oziroma ob kliku na gumb `nazaj` na Android napravi je uporabnik, potem ko v oknu potrdi svojo odločitev, prav tako preusmerjen na glavno stran aplikacije (`Main`). V razredu se izvajajo SQL poizvedbe na podatkovnih tabelah `CARS`, `CATEGORIES` in `COSTS`. Uporabniški vmesnik te aktivnosti lahko vidimo na sliki 33;

Slika 33: Uporabniški vmesnik aktivnosti `CostEntry`.

- **`CostRecords.java (costrecords.xml)`** - ta aktivnost nam omogoča prikaz zapisov o ostalih stroških za neko izbrano obdobje. Izbira enega od zapisov uporabnika preusmeri na aktivnost, kjer so prikazani podrobni podatki o tem zapisu (`CostDetails`). Možen je tudi izvoz zapisov o ostalih stroških v datoteko na zunanji pomnilnik. Ob izbiri gumba `Izvozi zapise` v uporabniškem vmesniku se najprej odpre okno, kjer je potrebno izbrati, katere zapise želimo izvoziti (vse zapise ali samo prikazane) ter format datoteke (`.txt` ali `.xml`). Nato se odpre še eno okno, kjer je potreben vpis imena datoteke. Temu sledi izvoz datoteke na zunanji pomnilnik (uporabnik je o uspešnem izvozu obveščen preko okna). Klik gumba `nazaj` na

Android napravi nas preusmeri na glavno stran aplikacije (`Main`). V razredu se izvajajo SQL poizvedbe na podatkovnih tabelah `CARS` in `COSTS`. Uporabniški vmesnik te aktivnosti lahko vidimo na sliki 34;

Slika 34: Uporabniški vmesnik aktivnosti `CostRecords`.

- **`CostDetails.java (costdetails.xml)`** - ta aktivnost nam omogoča prikaz podrobnih podatkov o izbranem zapisu o ostalih stroških. Izbira gumba `Izbriši` v uporabniškem vmesniku omogoča uporabniku izbris izbranega zapisa iz tabele `COSTS`, potem ko v oknu potrdi svojo odločitev o izbrisu. V primeru uspešnega izbrisa oziroma ob kliku na gumb `nazaj` na Android napravi je uporabnik preusmerjen na pregled zapisov o ostalih stroških za izbrano obdobje (`CostsRecords`). Izbira gumba `Uredi` v uporabniškem vmesniku pa uporabnika preusmeri na aktivnost, ki omogoča urejanje izbranega zapisa (`EditCost`). V razredu se izvajajo SQL poizvedbe na podatkovni tabeli `COSTS`. Uporabniški vmesnik te aktivnosti lahko vidimo na sliki 35;
- **`EditCost.java (editcost.xml)`** – ta aktivnost nam omogoča urejanje podatkov o izbranem zapisu o ostalih stroških. V primeru, da katera od polj niso izpolnjena oziroma niso izpolnjena pravilno, bo aplikacija uporabnika preko okna na to tudi opozorila. Ko uporabnik pravilno izpolni oziroma popravi vsa polja ter izbere gumb `Potrdi` v uporabniškem vmesniku, je preko okna obveščen o uspešni posodobitvi zapisa v tabeli `COSTS` ter preusmerjen na aktivnost, ki omogoča podroben pregled podatkov o izbranem zapisu o ostalih stroških (`CostDetails`). V primeru

Slika 35: Uporabniški vmesnik aktivnosti `CostDetails`.

izbire gumba `Prekliči` v uporabniškem vmesniku oziroma ob kliku na gumb `nazaj` na Android napravi je uporabnik, potem ko v oknu potrdi svojo odločitev, prav tako preusmerjen na prej omenjeno aktivnost (`CostDetails`). V razredu se izvajajo SQL poizvedbe na podatkovnih tabelah `CATEGORIES` in `COSTS`. Uporabniški vmesnik te aktivnosti lahko vidimo na sliki 36;

Slika 36: Uporabniški vmesnik aktivnosti `EditCost`.

- **`StatisticsRecords.java (statisticsrecords.xml)`** – ta aktivnost nam za izbrano obdobje omogoča prikaz osnovnih statističnih podatkov zapisov o porabi goriva (število zapisov, skupni znesek, skupno natočenih litrov goriva, skupno prevoženih kilometrov, minimalna ter maksimalna povprečna poraba goriva, skupna povprečna poraba goriva ter število točenj goriva glede na posamezne bencinske črpalke in vrsto goriva) ter ostalih stroških (skupni znesek glede na posamezno kategorijo in skupni znesek vseh kategorij skupaj). Klik gumba `nazaj` na Android napravi nas preusmeri na glavno stran aplikacije (`Main`). V razredu se izvajajo SQL poizvedbe na podatkovnih tabelah `COSTS` in `FUELCONSUMPTION`. Uporabniški vmesnik te aktivnosti lahko vidimo na sliki 37;

Slika 37: Uporabniški vmesnik aktivnosti `StatisticsRecords`.

- **`CarData.java (cardata.xml)`** – ta aktivnost nam omogoča prikaz vseh podatkov o trenutno izbranem avtomobilu. Izbira gumba `Izbriši` v uporabniškem vmesniku omogoča uporabniku izbris trenutno izbranega avtomobila iz tabele `CARS`, potem ko v oknu potrdi svojo odločitev o izbrisu. Izbris avtomobila je možen samo v primeru, če v tabeli `CARS` obstaja več kot en zapis (v nasprotnem primeru bo aplikacija uporabnika preko okna opozorila, da izbris ni mogoč). V primeru uspešnega izbrisa oziroma ob kliku na gumb `nazaj` na Android napravi bo uporabnik preusmerjen na glavno stran aplikacije (`Main`). Izbira gumba `Uredi` v uporabniškem vmesniku pa bo uporabnika preusmerila na aktivnost, ki omogoča

urejanje podatkov trenutno izbranega avtomobila (`EditCarData`). V razredu se izvajajo SQL poizvedbe na podatkovnih tabelah `CARS`, `COSTS` in `FUELCONSUMPTION`. Uporabniški vmesnik te aktivnosti lahko vidimo na sliki 38;

Slika 38: Uporabniški vmesnik aktivnosti `CarData`.

- **`EditCarData.java` (`editcardata.xml`)** - ta aktivnost nam omogoča urejanje podatkov o trenutno izbranem avtomobilu. V primeru, da katera od obveznih polj niso izpolnjena oziroma niso izpolnjena pravilno, bo aplikacija uporabnika preko okna na to tudi opozorila. Ko uporabnik pravilno izpolni oziroma popravi vsa obvezna polja ter izbere gumb `Potrdi` v uporabniškem vmesniku, je preko okna obveščen o uspešni posodobitvi zapisa v tabeli `CARS` ter preusmerjen na aktivnost, ki omogoča prikaz vseh podatkov o trenutno izbranem avtomobilu (`CarData`). V primeru izbire gumba `Prekliči` v uporabniškem vmesniku oziroma ob kliku na gumb `nazaj` na Android napravi je uporabnik potem, ko v oknu potrdi svojo odločitev, prav tako preusmerjen na prej omenjeno aktivnost (`CarData`). V razredu se izvajajo SQL poizvedbe na podatkovni tabeli `CARS`. Uporabniški vmesnik te aktivnosti lahko vidimo na sliki 39;

Slika 39: Uporabniški vmesnik aktivnosti `EditCarData`.

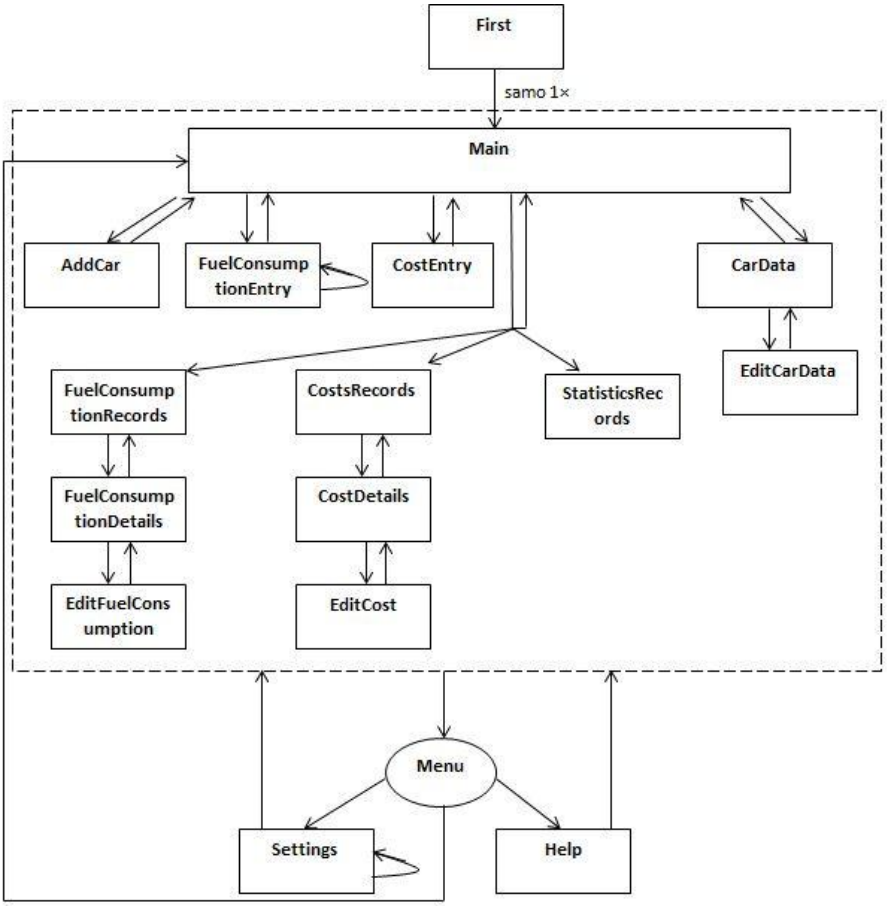
- **`Help.java (help.xml)`** – ta aktivnost nam omogoča prikaz pomoči za del aplikacije, kjer smo pomoč zahtevali. Na voljo so podrobna navodila kako uporabljati funkcionalnosti, ki so na voljo v tem delu aplikacije. Ob kliku na gumb `nazaj` na Android napravi je uporabnik preusmerjen na aktivnost dela aplikacije, kjer je bila pomoč zahtevana. V razredu se ne izvaja nobena SQL poizvedba na podatkovnih tabelah. Primer uporabniškega vmesnika te aktivnosti lahko vidimo na sliki 40;

Slika 40: Primer uporabniškega vmesnika aktivnosti `Help`.

- **`Settings.java (settings.xml)`** – ta aktivnost nam omogoča dostop do nastavitev v aplikaciji. Uporabnik v nastavitvah lahko nastavi število dni in število kilometrov, preden ga aplikacija prične opozarjati na servis in registracijo, prav tako pa uporabnik lahko izbere, katere vrste goriv uporablja pri trenutno izbranem avtomobilu (avtoplin, diesel, 95-okt. in 100-okt.). Ob izbiri gumba `Posodobi cene goriv` v uporabniškem vmesniku je preko svetovnega spleta možno pridobiti podatke o trenutnih cenah vseh pogonskih goriv (ob tem se cene goriv v tabeli `FUELS` avtomatsko posodobijo na nove vrednosti). Za takojšen prikaz posodobljenih cen goriv v tej aktivnosti je potrebna osvežitev aktivnosti `Settings`. Ob kliku na gumb `nazaj` na Android napravi je uporabnik preusmerjen na aktivnost dela aplikacije, kjer so bile nastavitve odprte. V razredu se izvajajo SQL poizvedbe na podatkovnih tabelah `CARS` in `FUELS`. Uporabniški vmesnik te aktivnosti lahko vidimo na sliki 41;

Slika 41: Primer uporabniškega vmesnika aktivnosti `Settings`.

Slika 42 nam prikazuje vse možne prehode med posameznimi aktivnostmi.



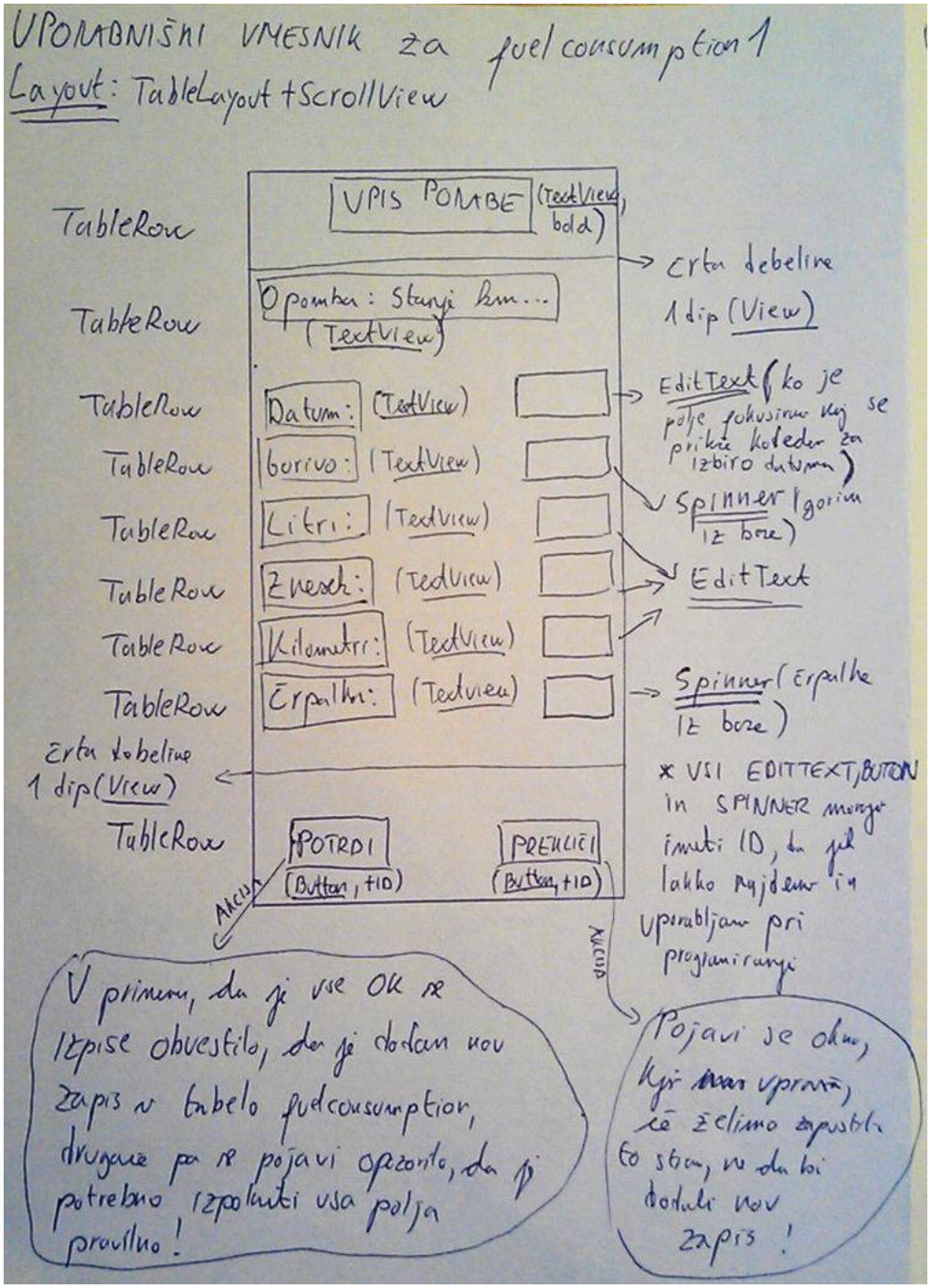
Slika 42: Možni prehodi med aktivnostmi.

3.4.3 Primer načrtovanja uporabniškega vmesnika

Za vsak razred, ki predstavlja aktivnost, je potrebno izdelati tudi uporabniški vmesnik za komunikacijo z uporabnikom. Za načrtovanje uporabniških vmesnikov nismo uporabili nobenega programskega orodja, ampak smo vse uporabniške vmesnike načrtovali ročno. Iz opisa samega načrtovanja delovanja aktivnosti lahko vidimo, da aplikacija AvtoDroid vsebuje kar sedemnajst različnih uporabniških vmesnikov. Ker bi bilo predstavljanje načrtovanja vseh nesmiselno, bomo predstavili načrtovanje le enega. Na sliki 43 lahko vidimo načrtovanje uporabniškega vmesnika za aktivnost, ki omogoča vpis nove porabe goriva (`fuelconsumptionentry1.xml`).

Na podlagi te skice uporabniškega vmesnika smo se lotili izdelave uporabniškega vmesnika v razvojnem okolju Eclipse, in sicer z orodjem imenovanim Android Layout Editor. Izgradnjo tega uporabniškega vmesnika bomo predstavili v poglavju 3.5.1. Seveda je pri sami gradnji uporabniškega vmesnika prišlo do nekaterih popravkov oziroma optimizacij, saj skica ni

dovolj natančna, prav tako pa se pri sami implementaciji aplikacije običajno pojavijo še kakšne nove ideje.



Slika 43: Primer načrtovanja uporabniškega vmesnika.

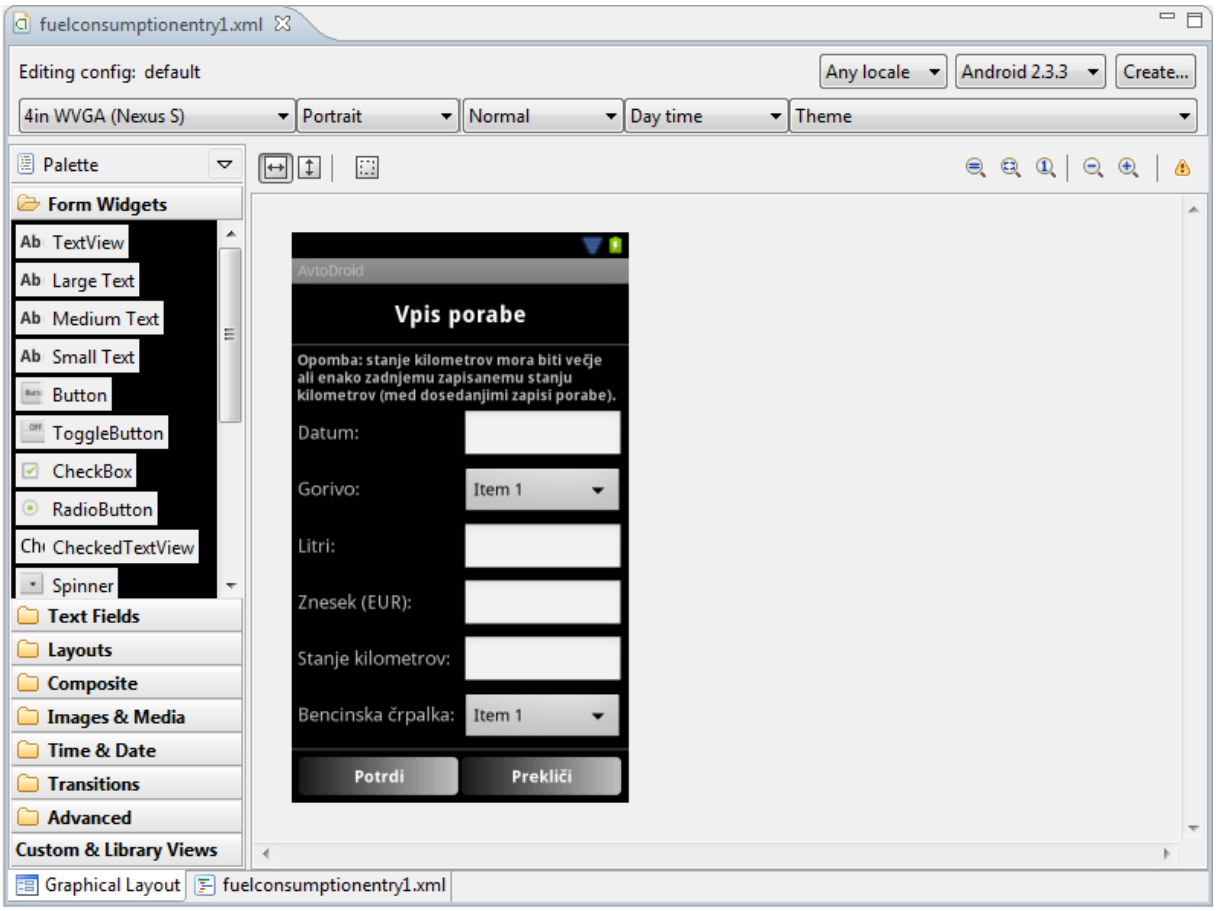
3.5 Implementacija aplikacije

Načrtovanju aplikacije je sledila implementacija. Najprej smo s pomočjo orodja SQLite Database Browser implementirali načrtovani model podatkovne baze. Kreirano podatkovno bazo smo poimenovali `avtodroiddatabase`, sama imena podatkovnih tabel v tej bazi pa so enaka načrtovanim, prav tako tudi polja posameznih tabel. Sledila je implementacija posameznih delov aplikacije. V nadaljevanju bomo predstavili samo najpomembnejše (oziroma najpogosteje uporabljene) dele kode aplikacije, saj bi bilo predstavljanje celotne kode aplikacije nesmiselno (skupaj ima aplikacija brez upoštevanja `.xml` datotek, ki predstavljajo uporabniške vmesnike, več kot 8000 vrstic kode). Predstavili bomo tudi `AndroidManifest.xml` datoteko naše aplikacije, datoteko `menu.xml`, ki predstavlja osnovni meni, primere obeh vrst izvoznih datotek ter primer izgradnje uporabniškega vmesnika aktivnosti, ki omogoča vpis nove porabe goriva.

3.5.1 Primer izgradnje uporabniškega vmesnika

Na podlagi načrtovanih uporabniških vmesnikov se najprej lotimo izgradnje uporabniškega vmesnika neke aktivnosti, nato pa še implementacije razreda te aktivnosti. Kot že rečno, vsaka aktivnost potrebuje (sicer ni nujno) uporabniški vmesnik, ki ga najlažje zgradimo s pomočjo orodja Android Layout Editor. Primer izgradnje vmesnika v tem orodju lahko vidimo na sliki 44 (v meniju na levi strani so zbrani vsi gradniki, ki jih lahko uporabimo pri gradnji, medtem ko je v osrednjem delu prikazan trenutni izgled grafičnega uporabniškega vmesnika, ki ga gradimo). Najprej izberemo samo postavitev grafičnega uporabniškega vmesnika (mi smo v večini primerov uporabili postavitev `TableLayout` v kombinaciji s `ScrollView`, tako da so v primeru velika števila gradnikov prikazani tudi drsniki, kateri nam omogočeno pomikanje (angl. *scroll*) v smeri gor-dol (angl. *up-down*) in obratno po uporabniškem vmesniku). Nato sledi postopno dodajanje gradnikov v uporabniški vmesnik. To poteka na način, da izberemo želen gradnik, ga postavimo na mesto, kjer ga potrebujemo, ter ga nato ustrezno oblikujemo ter mu po potrebni določimo tudi enolično ime (da ga pri programiranju aktivnosti lahko poiščemo ter spreminjamo).

Najpogostejši gradniki, ki smo jih uporabili pri gradnji uporabniški vmesnikov aplikacije AvtoDroid so `TextView`, `EditText`, `Button`, `Spinner`, `ListView`, `CheckBox` itd. Gradnikov ne bomo podrobno predstavili, saj so zelo podobni gradnikom v drugih programskih oziroma označevalnih jezikih.



Slika 44: Primer izgradnje uporabniškega vmesnika s pomočjo orodja Android Layout Editor.

Pri gradnji uporabniškega vmesnika na sliki 44 smo uporabili postavitev `TableLayout` znotraj `ScrollView` ter tudi (za postavitev gumbov) `LinearLayout` znotraj `TableLayout`. Uporabili smo gradnike `TextView`, `EditText`, `View`, `Spinner` in `Button`. Celotna datoteka `fuelconsumptionentry1.xml` vsebuje 186 vrstic, zato bomo predstavili le kratek izsek iz te datoteke:

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
            android:id="@+id/scroll"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content">

<TableLayout
            xmlns:android="http://schemas.android.com/apk/res/android"
            android:layout_height="fill_parent"
            android:layout_width="fill_parent">


...
```

```
...
<TableRow>
            <TextView
             android:layout_marginLeft="5dp"
             android:text="Datum: "
             android:textSize="18sp"
             android:layout_marginTop="5dp"
             android:layout_weight="1" />

            <EditText
             android:id="@+id/porabaDatum"
             android:digits="0123456789."
             android:inputType="date"
             android:focusable="false"
             android:layout_marginBottom="5dp"
             android:layout_marginRight="5dp"
             android:layout_marginTop="5dp"
             android:layout_weight="1"
             android:layout_width="150dp"/>
</TableRow>
...
</TableLayout>
</ScrollView>
```

Predstavljen izsek vsebuje (poleg same postavitve vmesnika) dva gradnika, in sicer `TextView` in `EditText`. Prvi gradnik, `TextView`, smo uporabili za izpis besedila »Datum:«, medtem ko bomo drug gradnik, `EditText`, uporabili za branje datuma, ki ga bo uporabnik vnesel v to polje. Ker bo branje potekalo v sami programski kodi (kako branje poteka, bomo predstavili v nadaljevanju), moramo temu gradniku določiti tudi enolično ime (atribut `android:id`), da ga bomo lahko poiskali, ko bomo želeli prebrati vrednost tega polja. Obema gradnikoma smo določili oblikovne lastnosti, pri drugem gradniku pa smo dodali še omejitve vnosa (uporabnik lahko v to polje vnaša samo znak . in števke).

Kot smo lahko opazili je gradnja uporabniških vmesnikov Android aplikacij zelo enostavna, saj jih ni potrebno graditi v sami programski kodi, prav tako pa lahko takoj vidimo grafičen izgled vmesnika (ni potrebno nobenega prevajanja).

3.5.2 Osnovni meni aplikacije

Izgradnja izgleda osnovnega menija aplikacije, do katerega lahko pridemo s pomočjo gumba `menu` na Android napravi, poteka v orodju imenovanem Android Menu Editor. Sama gradnja poteka tako, da dodajamo elemente, katere želimo imeti v meniju, vsakemu elementu pa določimo naslov, ikono ter enolično ime, na katerega se bomo sklicevali ob implementaciji programske logike menija. Določimo lahko tudi nekatere druge lastnostni, ki pa jih mi nismo uporabljali. Vsebina datoteke `menu.xml` naše aplikacije je takšna:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/nastavitve"
          android:title="Nastavitve"
          android:icon="@drawable/ic_menu_preferences" />
    <item android:id="@+id/pomoc"
          android:title="Pomoč"
          android:icon="@drawable/ic_menu_help" />
    <item android:id="@+id/nazaj"
          android:title="Nazaj na začetek"
          android:icon="@drawable/ic_menu_back" />
    <item android:id="@+id/izhod"
          android:title="Izhod"
          android:icon="@drawable/ic_menu_close_clear_cancel" />
</menu>
```

Kot lahko razberemo ima naš osnovni meni (katerega izgled smo predstavili na sliki 25 v poglavju 3.4.2) štiri elemente, in sicer `Nastavitve` (ime elementa je `nastavitve`), `Pomoč` (ime elementa je `pomoc`), `Nazaj na začetek` (ime elementa je `nazaj`) in `Izhod` (ime elementa je `izhod`). Ikone, ki predstavljajo te elemente, imamo shranjene v mapi `drawable`. Ta datoteka pa še ne predstavlja delovanja osnovnega menija, ampak samo izgled menija. Programsko logiko moramo implementirati pri vsaki aktivnosti (oziroma razredu) posebej (kjer želimo, da je meni na voljo). Metodi, ki ju je potrebno implementirati, sta `onCreateOptionsMenu()` in `onOptionsItemSelected()`:

```
@Override
    public boolean onCreateOptionsMenu(Menu menu) {
      MenuInflater inf = getMenuInflater();
      inf.inflate(R.layout.menu, menu);
      return true;
    }
      @Override
    public boolean onOptionsItemSelected(MenuItem item) {
      switch(item.getItemId()){
      case R.id.nazaj:
            Intent back = new Intent(getApplicationContext(),Main.class);
            startActivity(back);
            return true;
      case R.id.izhod:
            finish();
            return true;
      case R.id.pomoc:
            Intent help = new Intent(getApplicationContext(),Help.class);
            startActivity(help);
            return true;
      case R.id.nastavitve:
            Intent nastavi = new
            Intent(getApplicationContext(),Settings.class);
            startActivity(nastavi);
            return true;
      default:
            return false;
      }
    }
```

Prva metoda (`onCreateOptionsMenu(Menu menu)`) nam omogoča prikaz osnovnega menija. Če ta metoda v razredu ni implementirana, potem je gumb `menu` na Android napravi neaktiven. Pri drugi (`onOptionsItemSelected(MenuItem item)`) je potrebno implementirati programsko logiko, da bo meni deloval tako, kot smo si zamislili. Prehode med aktivnostmi nam omogoča objekt razreda `Intent`, sam zagon nove aktivnost pa metoda `startActivity(Intent intent)` (samo prehajanje med aktivnostmi bo predstavljeno v nadaljevanju). Naš osnovni meni deluje na naslednji način:

- ob izbiri elementa `nazaj` smo preusmerjeni na glavno stran aplikacije (aktivnost `Main`),
- ob izbiri elementa `izhod` se aplikacija zapre,
- ob izbiri elementa `pomoc` smo preusmerjeni na stran, ki nam nudi pomoč pri uporabi nekega dela aplikacije (aktivnost `Help`),
- ob izbiri elementa `nastavitve` pa smo preusmerjeni na stran, kjer lahko pregledujemo oziroma spreminjamo nastavitve aplikacije (aktivnost `Settings`).

3.5.3 Datoteka AndroidManifest.xml

Ko smo zgradili vse grafične uporabniške vmesnike aplikacije ter osnovni meni, smo se pred kodiranjem aplikacijskih delov lotili še pisanja datoteke `AndroidManifest.xml`. Glede na načrt delovanja posameznih delov aplikacije (ter specifikacije zahtev) smo dobili naslednjo vsebino datoteke (predstavljen bo le kratek izsek iz te datoteke) :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="avtodroid.activities"
          android:versionCode="1"
          android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="10" />
    <uses-sdk android:minSdkVersion="10"/>

    <uses-permission
     android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission
     android:name="android.permission.INTERNET"/>
    <uses-permission
     android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission
     android:name="android.permission.VIBRATE" />

    <application
     android:name=".Support"
     android:icon="@drawable/carz"
     android:label="@string/app_name">
...
```

```
...
        <activity
         android:name=".First"
         android:label="@string/app_name"
         android:windowSoftInputMode="stateHidden"
         android:screenOrientation="portrait"
         android:theme="@android:style/Theme.NoTitleBar">
         <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
         </intent-filter>
        </activity>

        <activity
         android:name=".Main"
         android:windowSoftInputMode="stateHidden"
         android:screenOrientation="portrait"
         android:theme="@android:style/Theme.NoTitleBar"
        />
...
</application>
</manifest>
```

Manifest datoteka vsebuje vse razrede, ki so razširitev razreda `Activity` ali `Application` (če npr. katere od aktivnosti ne bi našteli v tej datoteki, potem bi se v primeru, da bi želeli to aktivnost zagnati, aplikacija zaradi napake ustavila) ter potrebna dovoljenja za uporabo. Atributu `package` značke (angl. *tag*) `<manifest>` je potrebno nastaviti ime glavnega paketa, v katerem je prisotna glavna aktivnost aplikacije, v našem primeru je to paket `android.activities`. Atribut `android:minSdkVersion` značke `<uses-sdk>` vsebuje vrednost, ki predstavlja minimalno verzijo operacijskega sistema, na kateri naša aplikacija deluje (API 10 ustreza verziji Android 2.3). Pri znački `<uses-permission>` atribut `android:name` vsebuje vrednost, ki predstavlja, katera dovoljena aplikacija zahteva pri svojem delovanju. V našem primeru aplikacija zahteva dostop do interneta (`android.permission.INTERNET`), možnost pisanja na zunanji pomnilnik (`android.permission.WRITE_EXTERNAL_STORAGE`), dostop do pregleda trenutnega stanja omrežja (`android.permission.ACCESS_NETWORK_STATE`) ter možnost uporabe vibriranja (`android.permission.VIBRATE`). Ker imamo v naši aplikaciji razred, ki razširja razred `Application`, moramo to navesti tudi v manifest datoteki (značka `<application>`), kjer v atributih značke vpišemo ime aplikacije (`Support`), izberemo kje se nahaja ikona naše aplikacije itd. Na koncu moramo dodati še vsako od aktivnosti aplikacije, ki jih je našem primeru 16. V izseku datoteke sta dodani dve aktivnosti, in sicer `First` ter `Main` (vsaka aktivnost posebej mora biti navedena v svoji znački `<activity>`, ime aktivnosti pa je vsebovana v atributu `android:name`). V ostalih atributih značke lahko nastavimo nekaj dodatnih lastnosti aktivnosti (pri vseh aktivnostih smo

recimo izbrali, da ne želimo imeti prikazane naslovne vrstice aplikacije, ki vsebuje ime aplikacije, da ni možno spreminjati orientacije aplikacije itd.).

3.5.4 Interakcija s podatkovno bazo

Za interakcijo s podatkovno bazo imamo v naši aplikacijo poseben razred `DatabaseAdapter`, ki je razširitev razreda `SQLiteOpenHelper`. V tem razredu imamo zbrane vse metode, ki jih potrebujemo pri interakciji naše aplikacije s podatkovno bazo `androiddatabase`. Podatkovna baza, ki smo jo ustvarili, se v razvojnem okolju Eclipse nahaja v mapi `assets` (opisana je bila v poglavju 2.1.6). To podatkovno bazo je ob prvem zagonu aplikacije na mobilni napravi najprej potrebno skopirati na notranji pomnilnik naprave, kjer je nameščena tudi aplikacija. Ko je podatkovna baza na voljo na notranjem pomnilniku naprave, lahko v aplikaciji začnemo uporabljati vse podatkovne tabele, ki so del podatkovne baze. Sledi predstavitev vseh metod, ki so del razreda `DatabaseAdapter`.

Metoda za kopiranje podatkovne baze na notranji pomnilnik naprave

```
private static final String BASE_NAME = "avtodroiddatabase";
private static String BASE_PATH = "/data/data/"+
context.getApplicationContext().getPackageName()+"/databases/";

private void copyDatabase() throws IOException{

            //odprem podatkovno bazo, ki se nahaja v mapi assets
            InputStream input = context.getAssets().open(BASE_NAME);

            //pot do baze na mobilni napravi
            String dat = BASE_PATH+BASE_NAME;

            //odprem novo datoteko, ki predstavja PB
            OutputStream out = new FileOutputStream(dat);

            byte [] buffer = new byte [1024];
            int len;

            //prepis podatkovne baze iz mape assets
            while((len=input.read(buffer))>0){
                  out.write(buffer, 0, len);
            }

            out.flush();
            out.close();
            input.close();

}
```

Spremenljivka `BASE_NAME` predstavlja ime naše podatkovne baze, medtem ko spremenljivka `BASE_PATH` vsebuje pot do mape, kjer bo datoteka na mobilni napravi shranjena (vse SQLite podatkovne baze neke aplikacije so na napravi shranjene v mapi

`/data/data/ime_glavnega_paketa_aplikacije/databases`). Vse datoteke, ki so shranjene v mapi `assests`, dobimo s pomočjo ukaza `context.getAssets().open(ime_datoteke)`, v našem primeru je `ime_datoteke` enako `avtodroiddatabase` (spremenljivka `BASE_NAME`). Nato sledi kopiranje podatkovne baze, ki deluje po istem principu kot kopiranje navadne tekstovne datoteke.

Odpiranje in zapiranje podatkovne baze

```
private static final String BASE_NAME = "avtodroiddatabase";
private static String BASE_PATH = "/data/data/"+
context.getApplicationContext().getPackageName()+"/databases/";
private SQLiteDatabase database;
private final Context context;


//konstruktor
private DatabaseAdapter (Context context){
            super(context, BASE_NAME, null, 1);
            this.context=context;
}
//odpiranje
public void openDatabase() throws SQLException{
            String dat = BASE_PATH+BASE_NAME;
            database = SQLiteDatabase.openDatabase(dat, null,
            SQLiteDatabase.OPEN_READWRITE);
}
//zapiranje
public synchronized void closeDatabase(){
            if(database!=null){
                  database.close();
            }
            super.close();
}
//kreiranje
public void createDatabase() throws IOException{
            boolean zeObstaja = checkDatabase();
            if(!zeObstaja){
                  this.getReadableDatabase();
                  try{
                        copyDatabase();
                  }
                  catch (IOException e){
                        throw new Error("Napaka pri kopiranju baze!");
                  }
            }
}
//preverimo, če imamo že podatkovno bazo na notranjem pomnilniku naprave
private boolean checkDatabase(){
            File dbFile = new File(BASE_PATH+BASE_NAME);
            return dbFile.exists();

}
```

Metode `createDatabase()`, `openDatabase()` in `closeDatabase()` se uporabljajo pri vseh razredih aplikacije, kjer poteka interakcija s katero od podatkovnih tabel. Sledi predstavitev odpiranja in zapiranja podatkovne baze na konkretnem primeru (v vseh razredih je postopek popolnoma identičen):

```
private DatabaseAdapter databaseConnection;
databaseConnection = DatabaseAdapter.getDatabaseAdapter(this);

try{
    databaseConnection.createDatabase();
}
catch (IOException e){
    throw new Error ("Povezava z bazo ni uspela!");
}

databaseConnection.openDatabase();
//interakcija s podatkovnimi tabelami (branje, pisanje, spreminjanje,
//brisanje)
databaseConnection.closeDatabase();
```

Branje, pisanje, spreminjanje in brisanje zapisov v podatkovni tabeli

```
private SQLiteDatabase database;

//vnos novega zapisa
public long insertRecordsInDB(String tableName, String nullColumnHack,
                ContentValues initialValues) {
            return database.insert(tableName, nullColumnHack,
                   initialValues);
}

//spreminjanje zapisa
public long updateRecordInDB(String tabela, ContentValues values,
                String where, String [] arg){
            return database.update(tabela, values, where, arg);
}

//brisanje zapisa
public long deleteRecordInDB(String tabela, String where, String [] arg){
            return database.delete(tabela, where, arg);
}

//branje zapisa (poizvedba)
public Cursor query(String poizvedba){
            return database.rawQuery(poizvedba,null);
}
```

To so vse metode, ki jih rabimo pri delu s podatkovnimi tabelami. Podatke za recimo nov vnos stroška v podatkovno tabelo `COSTS` vnese uporabnik preko gradnikov uporabniškega vmesnika, katerih vrednosti nato preberemo in jih zapišemo v tabelo. Praktične primere bomo predstavili na podatkovni tabeli `COSTS` (metode se pri vseh podatkovnih tabelah uporablja na

enak način, različna so le imena tabel ter imena polj tabel). Pri vseh primerih bo imel zapis, ki ga bomo obravnavali, vrednost polja `idCost` enako 1, v primerih poizvedbe ter spreminjanja zapisa pa nas bo zanimalo samo polje `name` zapisa (vrednost polja `idCost` je prav tako 1).

```
private DatabaseAdapter databaseConnection;
//povezava s podatkovno bazo je vzpostavljena, baza je odpra

//BRANJE (poizvedba)
Cursor c = databaseConnection.query("SELECT * FROM costs
                                     WHERE idCost=1;");
int nameIndex = c.getColumnIndex("name");
String name = "";
if(c!=null && c.moveToFirst()){
    name = c.getString(nazivIndex)
}
else{
    //zapisa ni v tabeli costs
}
c.close();
//BRISANJE
String [] arg = {"1"};
long l;
try{
    l = databaseConnection.deleteRecordInDB("costs", "idCost=?", arg);
}catch(Exception e){
    throw new Error("Prislo je do napake");
}
//SPREMINJANJE
ContentValues vrednosti = new ContentValues();
vrednosti.put("name", "Novo Ime");
String [] arg = {"1"};
long l;
try{
    l = databaseConnection.updateRecordInDB("costs", vrednosti,
                                            "idCost=?", arg);

}catch(Exception e){
    throw new Error("Prislo je do napake.");
}
//VNOS
ContentValues vrednosti = new ContentValues();
vrednosti.put("name", "Redni servis");
vrednosti.put("carId", 1);
vrednosti.put("categoryName", "Servis");
vrednosti.put("amount", 159);
vrednosti.put("dateOfCost", "1.1.2012");
vrednosti.put("description", "Ni opisa.");
long l;
try{
    l = databaseConnection.insertRecordsInDB("costs", null, vrednosti);
}catch(Exception e){
    throw new Error("Prislo je do napake.");
}

//zapiranje podatkovne baze
```

3.5.5 Definiranje uporabniškega vmesnika v aktivnosti

V principu naj bi vse aktivnosti omogočale komunikacijo z uporabnikom, zato vsak razred, ki je razširitev razreda `Activity`, poskrbi za ustvarjanje okna, kateremu lahko določimo uporabniški vmesnik. Vsi uporabniški vmesniki, ki smo jih zgradili v razvojnem okolju Eclipse, se nahajajo v mapi `res/layout` (opisana je bila v poglavju 2.1.6). V tej mapi se nahajajo tudi vsi uporabniški vmesniki za aktivnosti v naši aplikaciji, za aktivnost `FuelConsumptionEntry` pa sta na voljo celo dva različna uporabniška vmesnika (uporabniška vmesnika se prikazujeta glede na trenutno stanje zadnjega zapisa v tabeli `FUELCONSUMPTION` – če je ta zapis dokončan se prikaže prvi uporabniški vmesnik, v nasprotnem primeru pa drugi). Za primer določimo uporabniški vmesnik `main.xml` oknu aktivnosti `Main`:

```
public void onCreate(Bundle savedInstanceState){
      …

      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      //dogajanje pri posamezni aktivnosti…

      …
}
```

Vsaki aktivnosti torej lahko uporabniški vmesnik določimo s pomočjo metode `setContentView (View view)`, kjer kot argument nastopa referenca na datoteko, ki predstavlja uporabniški vmesnik aktivnosti. Razred `R` je avtomatsko generiran razred, ki za vse uporabniške vmesnike (oziroma tudi za vse gradnike uporabniškega vmesnika) v aplikaciji definira statične reference.

3.5.6 Prehod med aktivnostmi

Možnost prehoda med aktivnostmi nam zagotavlja dinamično uporabo aplikacije, saj ima načeloma vsaka aktivnost (oziroma razred) svoj uporabniški vmesnik, kjer je uporabniku omogočena komunikacija z aplikacijo. S tem, ko je zagotovljen prehod med aktivnostmi, je zagotovljen tudi prehod med uporabniškimi vmesniki. Zagon aktivnosti v času izvajanja druge aktivnosti je zelo preprost, način kako prehod med aktivnostma zagotoviti pa je vedno enak ne glede na to, kaj trenutna oziroma naslednja zagnana aktivnost omogoča. Nova aktivnost je aktivirana s pomočjo asinhronskega sporočila, imenovanega namera. Namero ustvarimo s kreiranjem novega objekta razreda `Intent`, ki mu kot parameter posredujemo trenutni kontekst aplikacije (s pomočjo metode `getApplicationContext()`) ter aktivnost (oziroma razred aktivnosti), ki jo želimo zagnati. Aktivnost je nato zagnana s

pomočjo metode `startActivity (Intent intent)`, kjer kot parameter posredujemo pravkar ustvarjeno namero. Ker se prehod med aktivnostmi običajno izvede takrat, ko uporabnik izbere enega od gradnikov uporabniškega vmesnika (največkrat gumb), se ta postopek prehoda med aktivnostmi običajno implementira znotraj poslušalca dogodka (angl. *event listener*). Kako poteka sama implementacija poslušalcev dogodkov v naši aplikaciji si bomo pogledali v nadaljevanju. V aplikaciji AvtoDroid imamo zelo veliko prehodov med aktivnostmi (vsi možni prehodi so bili predstavljeni na sliki 42 v poglavju 3.4.2), zato ne bomo predstavljali vseh, ampak bomo predstavili samo primer prehoda med aktivnostma `Main` in `FuelConsumptionEntry` (torej uporabnik iz glavnega dela aplikacije z izbiro gumba `Vpis nove porabe goriva` želi vpisati novo porabo goriva):

```
//aktivnost Main, metoda onCreate()
//poslušalec dogodka gumba »Vpis nove porabe goriva«
//metoda onClick()
...

Intent startActivity = new Intent(getApplicationContext(),
                           FuelConsumptionEntry.class);
startActivity(startActivity);


...
```

3.5.7 Implementacija oken (dialogov)

Aplikacija AvtoDroid ima v osnovi dva tipa oken (oziroma dialogov): prva služijo opozarjanju in obveščanju uporabnika, druga pa od uporabnika zahtevajo izbor oziroma vnos podatkov (primere oken naše aplikacije lahko vidimo na sliki 45).

Slika 45: Primeri oken v aplikaciji.

Vsa okna so objekti razreda `AlertDialog`, le okno, kjer mora uporabnik izbrati datum, je objekt razreda `DatePickerDialog`. Tako kot pri prehodu med aktivnostmi se tudi dialog običajno pojavi takrat, ko uporabnik izbere enega od gradnikov uporabniškega vmesnika (obstajajo tudi izjeme, npr. ob zagonu aplikacije, ko aplikacija uporabnika avtomatsko opozori na bližajoči se servis ali registracijo), zato je tudi dialog potrebno običajno implementirati znotraj poslušalcev dogodkov. Najprej bomo predstavili dialog za opozarjanje uporabnika (na vrnitev na osnovno stran, ne da bi dodal nov zapis o porabi goriva), nato dialog, kjer je potreben vnos podatkov (število dni, preden aplikacija prične opozarjati na servis vozila), na koncu pa še dialog, kjer mora uporabnik izbrati datum (servisa). Vsi predstavljeni dialogi so vidni tudi na sliki 45. Kot bomo videli, so vsi dialogi implementirani na zelo podoben način, prav tako pa so na zelo podoben način (z nekaj manjšimi spremembami) implementirani dialogi, ki jih ne bomo podrobno predstavili.

Opozarjanje

```
//aktivnost FuelConsumptionEntry, metoda onCreate()
//poslušalec dogodka gumba »Prekliči«
//metoda onClick()
...
//ustvarimo nov dialog
AlertDialog.Builder ab= new AlertDialog.Builder
                        (FuelConsumptionEntry.this);
//besedilo v naslovni vrstici dialoga
ab.setTitle("Opozorilo");
//besedilo dialoga
ab.setMessage("Ali se želite vrniti na osnovno stran, ne da bi dodali nov
zapis o porabi goriva?");
//implementiramo poslušalca dogodka v primeru, da je izbran
//pozitivi gumb (ki ga poimenujemo »Da«)
ab.setPositiveButton("Da", new DialogInterface.OnClickListener() {
  public void onClick(DialogInterface dialog, int id) {
      //želimo vrnitev na glavno stran
      Intent naprej = new Intent(getApplicationContext(),Main.class);
      //ne želimo, da je dialog še prikazan, zato ga zavržemo
      dialog.dismiss();
      startActivity(naprej);
      //sami končamo trenutno aktivnost, saj je ne rabimo več
      finish();
  }
});
//implementiramo poslušalca dogodka v primeru, da je izbran
//negativni gumb (ki ga poimenujemo »Ne«)
ab.setNegativeButton("Ne", new DialogInterface.OnClickListener() {
  public void onClick(DialogInterface dialog, int id) {
       //ne želimo, da je dialog še prikazan, zato ga zavržemo
       dialog.dismiss();
  }
});
//prikažemo dialog
ab.show();

...
```

Vnos podatkov

```
//aktivnost Settings, metoda onCreate()
//spremenljivka p predstavlja instanco razreda Support
//vzpostaljeno imamo tudi povezavo s podatkovno bazo
//metoda changeServiceSettings()
...

//ustvarimo nov dialog
AlertDialog.Builder alert = new AlertDialog.Builder(this);
//besedilo v naslovni vrstici dialoga
alert.setTitle("Opozarjanje na servis");
//besedilo dialoga
alert.setMessage("Vnesite novo število dni:");
//v dialogu želimo imeti gradnik tipa EditText,
//tako da bo uporabnik lahko vnesel zahtevane podatke
final EditText input = new EditText(this);
input.setKeyListener(DigitsKeyListener.getInstance());
//dialogu dodamo ustvarjen gradnik
alert.setView(input);
//implementiramo poslušalca dogodka v primeru, da je izbran pozitivni
//gumb (ki ga poimenujemo »Potrdi«)
alert.setPositiveButton("Potrdi", new DialogInterface.OnClickListener() {
     public void onClick(DialogInterface dialog, int whichButton) {
        //branje vpisanih podatkov in zapis v PB
        databaseConnection.openDatabase();
        ContentValues vrednosti = new ContentValues();
        if(input.getText().toString().equals(""))
           input.setText(0+"");
        vrednosti.put("alertServiceKm", input.getText().toString());
        String [] arg = {p.getSelectedCar()+""};
        try{
            databaseConnection.updateRecordInDB("cars", vrednosti,
                                                "carId=?", arg);
        }catch(Exception e){
             throw new Error("Prislo je do napake.");
        }
        databaseConnection.closeDatabase();
        //dialoga ne potrebujemo več
        dialog.dismiss();
        //osvežimo aktivnost, tako da so prikazani posodobljene vrednosti
        startActivity(getIntent());
        finish();
      }
});
//še za negativni gumb (poimenujemo ga »Prekliči«)
alert.setNegativeButton("Prekliči",new DialogInterface.OnClickListener()
{
     public void onClick(DialogInterface dialog, int which) {
        dialog.dismiss();
     }
});
alert.show();
...
```

Kot smo lahko opazili, objektov razreda `AlertDialog` torej ne ustvarjamo neposredno, ampak preko razreda `AlertDialog.Builder`. V obeh primerih smo potem, ko smo

dialog kreirali, dialogu nastavili naslov in besedilo, kar sicer ni nujno potrebno, je pa smiselno. V drugem primeru smo dialogu dodali tudi vnosno polje, kamor bo uporabnik lahko vpisal podatke, po katerih ga dialog sprašuje. Nato je v obeh primerih sledila implementacija poslušalcev pozitivnega ter negativnega gumba dialoga, pri čemer smo sami poimenovali obe vrsti gumbov. V samih poslušalcih dogodkov smo nato še določili, kako naj se dialog odzove na klik obeh vrst gumbov. Dialog je v primeru, da se ob izbiri kateregakoli od gumbov zažene nova aktivnost, avtomatsko zavržen. V primeru, da želimo sami zavreči dialog (npr. da se ne zažene nobena nova aktivnost, dialoga pa ne želimo imeti več prikazanega), pa uporabimo metodo `dismiss()`. Vsi dialogi tipa `AlertDialog` so implementirani po istem principu, razlika je le v sami programski logiki poslušalcev dogodkov dialoga.

Dialog za izbiro datuma

```
//aktivnost FuelConsumptionEntry, izven vseh metod
private int mYearS;
private int mMonthS;
private int mDayS;
static int DATE_DIALOG_ID = 0;
//metoda onCreate()
//poslušalec dogodka vnosnega polja EditText (imenovan date), ki od
//uporabnika zahteva vnos datuma točenja goriva
//metoda onClick()
...
showDialog(DATE_DIALOG_ID);
...
//metoda onCreateDialog() se nahaja izven metode onCreate()
//ta metoda se kliče ob klicu metode showDialog
protected Dialog onCreateDialog(int id) {
         switch (id) {
         case 0:
            return new DatePickerDialog(this,
                      mDateSetListenerS,
                      mYearS, mMonthS, mDayS);
         }
         return null;
}
...

//dialog mDateSetListenerS se tudi nahaja zunaj metode onCreate()
//omogoča prikaz dialoga za izbiro datuma
DatePickerDialog.OnDateSetListener mDateSetListenerS =
        new DatePickerDialog.OnDateSetListener() {
            @Override
            public void onDateSet(DatePicker view, int year,
                                  int monthOfYear, int dayOfMonth){
                mYearS = year;
                mMonthS = monthOfYear;
                mDayS = dayOfMonth;
                //posodobi vrednost polja date
            }

        };
...
```

V naši aplikaciji dialog za izbiro datuma povsod uporabljamo v kombinaciji z vnosnim poljem tipa `EditText`, kjer je od uporabnika zahtevan vnos nekega datuma (npr. servisa, registracije, točenja goriva itd.). Ko uporabnik želi v to polje vnesti datum, se mu prikaže dialog za izbiro datuma. Potem ko izbere želen datum ter potrdi svojo izbiro (z izbiro gumba `Set`) se tudi v vnosnem polju pojavi izbran datum.

3.5.8 Interakcija z gradniki uporabniškega vmesnika

Zelo pomemben del vsake aplikacije je tudi interakcija z gradniki uporabniškega vmesnika. Poleg tega, da je prav preko poslušalcev dogodkov gradnikov največkrat zagotovljen prehod med aktivnostmi, pa nam omogoča tudi to, da uporabnik recimo lahko dodaja nove zapise v podatkovne tabele, katerih vrednosti običajno preko tipkovnice vnese v vnosna polja. Čeprav imamo v naši aplikaciji največ gradnikov tipa `TextView` (prikaz besedila), je interakcije s tem tipom gradnikov zelo malo (največkrat služijo za izpis nekega statičnega besedila). V naši aplikaciji je prisotne največ interakcije z gradniki tipa `EditText` (vnosno polje), `Button` (gumb) in `Spinner` (podoben običajnim *drop-down* menijem). Ker interakcija z vsemi gradniki poteka na zelo podoben način, bomo predstavili primere interakcije samo s temi gradniki. Primer bo prikazan na vpisu nove porabe goriva. Če želimo omogočiti interakcijo z gradniki, moramo že pri sami izgradnji uporabniškega vmesnika tem gradnikom določiti enolično ime (atribut `android:id`), kar smo že omenili v poglavju 3.5.1.

```
//aktivnost FuelConsumptionEntry, izven vseh metod
//omogočena je povezava s PB
...
//predstavili bomo enega od vsakega tipa gradnikov
private EditText date;
private Spinner pump;
private Button confirm; //prikazano ime gumba v UV je »Potrdi«
//tu naštejemo še vse ostale…
...

//metoda onCreate()
...

//prej definiranim globalnim spremenljivkam določimo
//katere gradnike predstavljajo

date = (EditText)findViewById(R.id.porabaDatum);
pump = (Spinner)findViewById(R.id.porabaCrpalka);
confirm = (Button)findViewById(R.id.potrdi);

...
```

```
...
//gradniku Spinner (pump) moramo najprej dodati vsebino, ki naj se
//prikazuje (v tem primeru bencinske črpalke preberemo iz PB)
ArrayList<String> showPumps = new ArrayList<String>();
databaseConnection.openDatabase();
Cursor c = databaseConnection.query("SELECT * FROM pumps");
int pumpNameIndex = c.getColumnIndex("pumpName");
if(c!=null && c.moveToFirst()){
  do{
     //v zbirko najprej dodamo vse benicnske črpalke
     showPumps.add(c.getString(pumpNameIndex));
  }while(c.moveToNext());
}
c.close();
databaseConnection.closeDatabase();
//ustvarimo nov adapter, kateremu nastavimo želen tip Spinnerja ter mu
//dodamo vsebino iz zbirke showPumps
ArrayAdapter<String> ab = new ArrayAdapter<String>(
                        this,
                        android.R.layout.simple_spinner_item,
                        showPumps);
ab.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
//gradniku nastavimo ustvarjen adapter
pump.setAdapter(ab);
...
//gradniku Button (confirm) nastavimo poslušalca dogodkov. Ko uporabnik
//izbere gumb »Potrdi«, se preveri vsebina vseh polj, ki jih je uporabnik
//izpolnil. V primeru, da so vsa izpolnjena pravilno, se doda nov zapis
//sestavljen iz vsebine teh polj v podatkovno bazo fuelConsumption
confirm.setOnClickListener(new OnClickListener() {
      @Override
       public void onClick(View v) {
            //pridobimo vsebino vseh polj, ki nas zanimajo
            String date = date.getText().toString();
            String pump = pump.getSelectedItem().toString();
            ...
            //preverimo ali so podatki ustrezni in če so
            //jih zapišemo v podatkovno tabelo
       }
});


...
```

Zgornji primer prikazuje najbolj tipično uporabo gradnikov uporabniškega vmesnika Android aplikacije. Spremenljivke, ki bodo predstavljale gradnike uporabniškega vmesnika, najprej definiramo na nivoju razreda, tako da jih potem lahko uporabimo kjerkoli v razredu. V metodi `onCreate()` vsako od teh spremenljivk inicializiramo s pomočjo metode `findViewById(View view)`, kjer kot argument nastopa referenca na gradnik (vse reference gradnikov, ki imajo določeno enolično ime, so shranjene v avtomatsko generiranem razredu `R`). Interakcija z gradniki se nato lahko prične. Pri gradnikih tipa `Spinner` in `EditText` največkrat želimo dostopati do (teksta) njihove trenutne vsebine, medtem ko gradnikom tipa `Button` največkrat nastavimo oziroma definiramo poslušalce dogodkov, kjer je nato implementirana programska logika želenega odziva na izbiro gumba.

3.5.9 Pridobivanje podatkov preko interneta ter izvoz podatkov

Aplikacija AvtoDroid deluje povsem lokalno. Zato so tudi vsi podatki, ki jih aplikacija potrebuje za delovanje, shranjeni v zapisih podatkovnih tabel v lokalni bazi podatkov ali pa je vsebina za prikaz shranjena že v samih gradnikih uporabniškega vmesnika. Vseeno pa lahko preko svetovnega spleta pridobimo podatke o trenutnih cenah vseh pogonskih goriv. Prav tako pa aplikacija omogoča tudi izvoz vseh zapisov (o porabi goriva ter ostalih stroških) uporabnika aplikacije na zunanji pomnilnik naprave, in sicer v formatu `.txt` ali `.xml`.

Razred `Fuel` nam omogoča pridobitev cen vseh pogonskih gorivo preko svetovnega spleta [15]. V razredu se uporabljajo standardni postopki za ustvarjanje `.xml` dokumenta, ki jih uporabljamo tudi pri drugih javanskih programih, zato tega razreda ne bomo podrobno predstavljali. Metoda, ki v tem razredu omogoča pridobitev cen pogonskih goriv, se imenuje `getFuelPrices()`. Sledi predstavitev dela programske kode, ki se izvede, ko uporabnik z izbiro gumba `Posodobi cene goriv` v aktivnosti `Settings` želi pridobiti podatke o trenutnih cenah goriv.

```
//aktivnost Settings, izven vseh metod
private Button updatePrices;
...
//metoad onCreate()
...
updatePrices = (Button)findViewById(R.id.posodobiGorivo);
...
updatePrices.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
         //pridobimo podatke o razpoložljivih omrežnih povezavah
         ConnectivityManager cm = (ConnectivityManager)
                                    getSystemService(CONNECTIVITY_SERVICE);
         NetworkInfo netInfo = cm.getActiveNetworkInfo();
         if (netInfo==null){
            //aplikacija mora uporabnika obvestiti, da ni na voljo nobene
            //aktivne povezave z internetom, zato cen goriv ni mogoče
            //pridobiti
         }
         else{
            //prikaz obvestila o posodabljanju cen goriv…
            //branje podatkov iz interneta je potrebno izvesti v novi
            //niti, tako da je obvestilo o posodabljanju goriv prisotno
            //dokler poteka branje podatkov iz interneta
            new Thread(new Runnable(){
               public void run() {
                  Fuel.getFuelPrices();
                  //zapis novih vrednosti cen goriv v tabelo fuels
                  //zavržemo dialog, ki prikazuje obvestilo o
                  //posodabljanju cen goriv…
                  //osvežimo trenutno aktivnost, da so prikazane že
                  //posodobljene cene goriv
            }).start();
         }
      }
});
```

V poslušalcu dogodkov gumba `Posodobi cene goriv` torej implementiramo celotno programsko logiko za pridobitev cen pogonskih goriv preko metode `getFuelPrices()`. Najprej je potrebno pridobiti podatke o trenutno razpoložljivih omrežnih povezavah (zato smo morali to dovoljenje dodati v datoteko `AndroidManifest.xml`). V primeru, da je na voljo aktivna povezava z internetom, lahko s pomočjo metode `getFuelPrices()` pridobimo cene goriv (tu je potrebno imeti dovoljenje za uporabo internetne povezave, ki smo ga že prej dodali v datoteko `AndroidManifest.xml`). Branje podatkov izvedemo v ločeni niti. Tako vemo, kdaj lahko zavržemo dialog, ki uporabnika obvešča o posodabljanju cen goriv, ter kdaj so podatki o cenah goriv uspešno posodobljeni v tabeli `FUELS`. Ko je branje končano moramo trenutno izvajano aktivnost `Settings` še osvežiti, da imamo prikazane že posodobljene cene goriv.

Aplikacija omogoča tudi izvoz zapisov o porabi goriva ter ostalih stroških na zunanji pomnilnik naprave. Ker izvoz obeh vrst zapisov poteka na enak način, bomo predstavili potek izvoza podatkov ter primer obeh formatov izvoženih datotek samo za zapise o ostalih stroških, povezanih s trenutno izbranim avtomobilom. Za izvoz datotek na zunanji pomnilnik moramo v datoteki `AndroidManifest.xml` dodati dovoljenje za pisanje na zunanji pomnilnik. Najprej sledi predstavitev dela programske kode, ki se izvede, ko uporabnik z izbiro gumba `Izvozi zapise` v aktivnosti `CostsRecords` želi izvozi zapise o ostalih stroških na zunanji pomnilnik naprave, nato pa še primeri izvoženih datotek.

```
//aktivnost CostsRecords, izven vseh metod
...
private Button export;
private String fileFormat="";
private String fileName="";
private int option = 0;
private XmlSerializer serializer;
...
//metoda onCreate()
...
export = (Button)findViewById(R.id.izvozStoroski);
...
export.setOnClickListener(newOnClickListener(){
      @Override
      public void onClick(final View v) {
          //dobimo stanje za zunanji pomnilnik (ali je na voljo za
          //pisanje)
          String stanje = Environment.getExternalStorageState();
          if (Environment.MEDIA_MOUNTED.equals(stanje)) {
              File root = Environment.getExternalStorageDirectory();
              //želimo imeti direktorij
              // /sdcard/zapisiOstaliStroski/imeDatoteke...
              final File direktorij = new
                    File(root.getAbsolutePath()+"/zapisiOstaliStroski");
              //ustvarimo ta direktorij (če še ni ustvarjen)
              direktorij.mkdirs();
...
```

```
...
            //prikaže se dialog, kjer uporabnik izbere, katere zapise
            //želi izvoziti (vse ali samo prikazane)
            //ter format izvozne datoteke, nato pa se prikaže dialog
            //kjer uporabnik vnese ime datoteke, ki se shrani v
            //spremenljivko fileName
            //ustvarimo novo datoteko z izbranim imenom in formatom...
            File file = new File(direktorij, fileName+fileFormat);
            try {
                file.createNewFile();
            }catch (IOException e1) {
                //prišlo je do napake
            }
            FileOutputStream fos=null;
            //izbran je bil format xml (option 0 ali 1)
            if(option==0 || option==1){
                 fos = new FileOutputStream(file);
                 serializer.setOutput(fos, "UTF-8");
                 serializer.startDocument(null, Boolean.valueOf(true));
                 serializer.setFeature
                 (http://xmlpull.org/v1/doc/features.html#indent-output,
                 true);
                 if(option==0)
                     getXml(indexi);
                 else
                     getXml(null);
                 serializer.endDocument();
                 serializer.flush();
                 fos.close();
                 //prikaži obvestilo o uspehu oz. neuspehu izvoza
            }
            //izbran je bil format txt (option 2 ali 3)
            else if(option==2){
                 vsebina = getTxt(indexi);
            }
            else if(option==3){
                 vsebina = getTxt(null);
            }
            if(option==2 || option==3){
                 byte [] data = vsebina.getBytes();
                 try {
                     fos = new FileOutputStream(file);
                     fos.write(data);
                     fos.flush();
                     fos.close();
                     //prikaži obvestilo o uspehu oz. neuspehu izvoza
                 }
                 catch (IOException e) {
                     //prišlo je do napake
                 }
        }
        else{
             //prikaže se obvestilo, da zunanji pomnilnik ni na voljo
             //za pisanje; izvoz ni možen
        }
    }
});
...
```

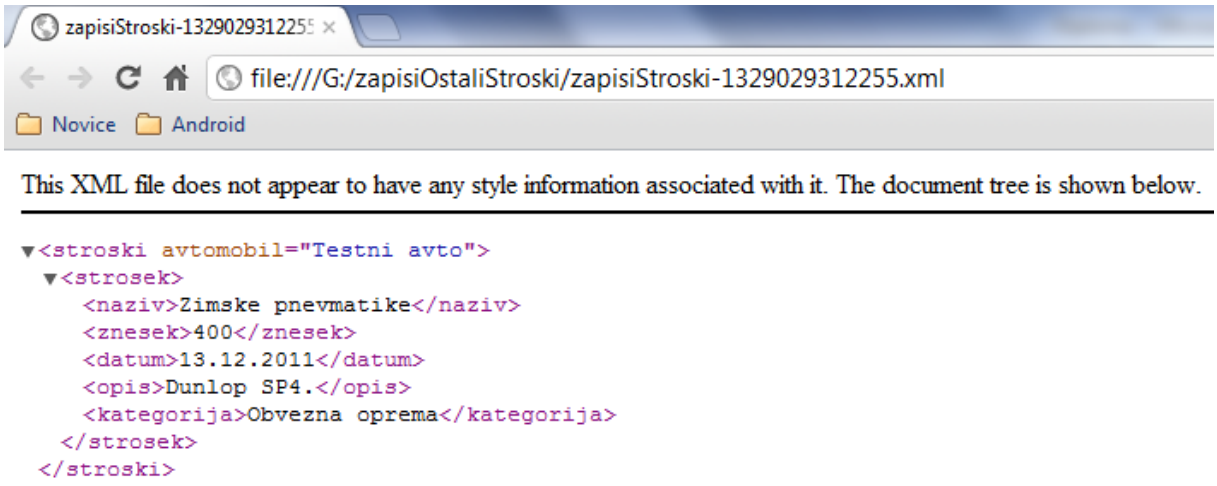
```
...
//konec metode onCreate()
private String getTxt(ArrayList<Integer> indexi){
    String opis ="";
    //iz podatkovne tabele cars naprej dobimo ime trenutnega avtomobila
    //iz podatkovne tabele costs nato pridobimo vse zapise, ki imajo
    //idCost enak enemu od vrednosti zbirke indexi
    //vsak zapis posebej sestavimo o obliko:
    //imeAvtomobila;naziv;znesek;datum;opis;kategorija
    //ter ga pripnemo nizu opis, ki ga metoda na koncu vrne
}

private void getXml(ArrayList<Integer> indexi) throws Exception{
    Cursor c;
    //iz podatkovne tabele cars naprej dobimo ime trenutnega avtomobila
    //začnemo oblikovati xml datoteko, najprej določimo korenski element
    serializer.startTag(null, "stroski");
    serializer.attribute(null, "avtomobil", imeAvtomobila);
    //iz podatkovne tabele costs nato pridobimo vse zapise, ki imajo
    //idCost enak enemu od vrednosti zbirke indexi, shranjeni so v
    //spremenljivki c
    //vsak zapis (vse elemente) posebej nato dodamo v xml datoteko
    if(c!=null && c.moveToFirst()){
       do{
         serializer.startTag(null, "strosek");
         serializer.startTag(null, "naziv");
         serializer.text(c.getString(nazivIndex));
         serializer.endTag(null, "naziv");
         serializer.startTag(null, "znesek");
         serializer.text(c.getString(cenaIndex));
         serializer.endTag(null, "znesek");
         ...
        serializer.endTag(null,"strosek");
       }while(c.moveToNext());
    }
    serializer.endTag(null, "stroski");
    c.close();
}
```

Na sliki 46 lahko vidimo primer izvoza enega zapisa o ostalih stroških v datoteko formata `.txt`, na sliki 47 pa v datoteko formata `.xml`:

zapisiStroski-1329029315489.txt

```
1  avtomobil;naziv;znesek;datum;opis;kategorija
2  Testni avto;Zimske pnevmatike;400;13.12.2011;Dunlop SP4.;Obvezna oprema
```

Slika 46: Primer izvoza enega zapisa v datoteko `.txt`.



Slika 47: Primer izvoza enega zapisa v datoteko `.xml`.

3.6 Testiranje aplikacije

Testiranje posameznih delov aplikacije je potekalo že med samim razvojem aplikacije tako na emulatorju (resolucija 320×480, operacijski sistem Android 2.3.3) kot tudi na pametnem mobilnem telefonu HTC Desire HD (resolucija 480×800, operacijski sistem Android 2.3.5). Pri testiranju posameznih modulov se je pojavilo kar nekaj napak (večina jih je nastala zaradi površnosti pri programiranju), ki pa smo jih sproti odpravili. Končna verzija aplikacije je bila z naše strani testirana na istih dveh napravah, s strani drugih dveh uporabnikov pa je bila testirana na pametnih mobilni telefonih Samsung Galaxy Ace (resolucija 320×480, operacijski sistem Android 2.3.6) in Samsung Galaxy S (resolucija 480×800, operacijski sistem Android 2.3.6). Pred testiranjem smo si najprej pripravili testni scenarij (upoštevali smo vse možne scenarije pri vsaki izmed funkcionalnosti aplikacije), nato pa smo se lotili testiranja najprej na emulatorju, potem pa še na pametnem mobilnem telefonu HTC Desire HD. Pri testiranju nismo našli nobenih napak, kar pomeni, da je bilo že samo testiranje posameznih modulov aplikacije dovolj dobro. Tudi samo delovanje aplikacije je bilo tekoče, tako na emulatorju kot tudi na fizični Android napravi. Tudi po približno dveh mesecih redne uporabe aplikacije zaenkrat nismo našli nobenih napak, prav tako pa napak ni bilo najdenih s strani drugih dveh uporabnikov. Enega od souporabnikov aplikacije je zmotilo dejstvo, da pri vpisu nove porabe goriva aplikacija ne preverja, ali je vpisan datum večji od datuma zadnjega zapisa v podatkovni tabeli `FUELCONSUMPTION`, enako velja za vpis novega stroška. Oba souporabnika pri izvozu zapisov pogrešata še druge možnosti izvozov (npr. izvoz zapisov o porabi goriva glede na bencinske črpalke itd.), prav tako pa pogrešata možnost uvoza zapisov v podatkovno bazo (npr. iz spletne aplikacije Poraba.com, kjer sta uporabnika že vrsto let in imata tam shranjenih že veliko zapisov).

4 Analiza razvoja aplikacije AvtoDroid

Z razvojem mobilne aplikacije AvtoDroid smo se ukvarjali približno en mesec (za vse faze razvoja aplikacije smo potrebovali nekaj več kot 100 ur dela, pri čemer je bilo več kot polovico tega časa porabljenega za implementacijo ter sprotno testiranje aplikacijskih modulov). Aplikacija obsega (tu so upoštevane samo javanske datoteke) nekaj več kot 8000 vrstic kode, velikost `.apk` datoteke (ki predstavlja namestitveno datoteko aplikacije) pa je okoli 150KB. Na mobilni napravi nameščena aplikacija zaseda (brez podatkov v lokalni bazi podatkov) nekaj manj kot 400KB prostora, sami podatki pa na naši mobilni napravi zaenkrat zasedajo 20KB prostora (po približno dveh mesecih uporabe). Pri razvoju aplikacije nismo imeli večjih težav, saj je na voljo ogromno različnih forumov, kjer smo našli odgovore na vsa vprašanja. Prav tako pa smo si pomagali z veliko različne literature, kjer je razvoj aplikacij zelo podrobno opisan. Že med samim razvojem aplikacije smo dobili nekaj idej za izboljšave oziroma za implementacijo dodatnih funkcionalnosti, vendar zaradi omejenega časa razvoja teh nismo realizirali. Te dodatne funkcionalnosti in izboljšave bodo verjetno realizirane enkrat v prihodnosti, v mislih pa imamo zaenkrat naslednje:

- poleg obstoječe možnosti vnosa nove porabe goriva bo uporabnik lahko izbral tudi drug način vnosa nove porabe goriva, ki bo deloval na podobnem principu kot pri spletni aplikaciji Poraba.com (uporabnik bo moral namesto izbire, ali je bil rezervoar prazen, izbrati, ali je bil rezervoar poln, saj se bo povprečna poraba računala enako kot pri aplikaciji Poraba.com). Uporabnik pa bo moral vseskozi uporabljati enak način vnosa nove porabe goriva;
- pri vnosu nove porabe goriva oziroma stroška bo aplikacija preverjala, ali je vpisan datum večji od datuma zadnjega zapisa v tabeli `FUELCONSUMPTION` oziroma `COSTS`;
- v lokalno podatkovno bazo, ki jo uporablja aplikacija, bo možen uvoz obstoječih zapisov iz aplikacije Poraba.com (oziroma morebitnih drugih zapisov, ki pa bodo morali biti v enakem formatu, kot so zapisi v izvoznih datotekah iz Poraba.com);
- na voljo bo več možnih načinov filtriranja zapisov porabe goriva in ostalih stroškov, ki jih bomo želeli imeti prikazati. Trenutno je opcija prikaz vseh zapisov ali glede na leto zapisa, nato pa bo možen tudi pregled zapisov glede na izbrano obdobje (izbor začetnega in končnega datum), glede na znesek (izbor minimalnega in maksimalnega zneska) itd. To bo omogočeno tudi pri izvozu zapisov v datoteko na zunanji pomnilnik;
- poleg formatov `.xml` in `.txt`, ki sta trenutno edina možna formata izvoznih datotek, bo možen izvoz tudi v formatu `.pdf`;

- možna bo izdelava varnostne kopije podatkovne baze, tako da v primeru izgube ali okvare mobilne naprave ne bo prišlo do izgube vseh podatkov. Varnostno kopijo bomo lahko izvozili na zunanji pomnilnik mobilne naprave, nato pa bo moral uporabnik sam poskrbeti, da jo bo iz zunanjega pomnilnika prenesel npr. na osebni računalnik.

Idej je sicer še veliko, vendar se tu postavlja vprašanje, ali jih je sploh mogoče oziroma smiselno uresničiti. Motivacije nam zaenkrat ne manjka, tako da upamo, da bomo katero od idej tudi uresničili.

5 Sklepne ugotovitve

V diplomskem delu smo v teoretičnem delu predstavili razvojna orodja in tehnologije, ki so povezane z razvojem mobilne aplikacije na platformi Android. V praktičnemu delu diplomskega dela smo nato podrobno predstavili vse faze razvoja mobilne aplikacije, katerih končni rezultat je delujoča aplikacija AvtoDroid. Pri razvoju nismo imeli večjih težav, saj je na temo razvoja Android aplikacij na voljo veliko literature in spletnih forumov, kjer smo našli vse odgovore na naša vprašanja. Implementirane so bile vse funkcionalnosti in lastnosti aplikacije, ki so bile zahtevane in načrtovane. AvtoDroid je mobilna aplikacija, ki jo lahko uporabljamo na katerikoli mobilni napravi, ki ima nameščen operacijski sistem Android. Uporabljamo jo lahko tako rekoč kjerkoli in kadarkoli, saj za njeno uporabo ni nikakršnih omejitev (npr. ni potrebno imeti aktivne internetne povezave). Uporabniški vmesnik je intuitiven in enostaven za uporabo, prav tako je enostavna uporaba vseh funkcionalnosti, za katere so nam na voljo tudi navodila oziroma pomoč pri uporabi. Prednost pred ostalimi podobnimi mobilnimi aplikacijami je v tem, da je popolnoma brezplačna ter (vsaj za uporabnike v Sloveniji) vsebuje uporabniški vmesnik v slovenskem jeziku. Tudi nabor nekaterih podatkov je prilagojen za slovenske uporabnike (npr. seznam bencinskih črpalk). Trenutno imamo v načrtu razvoj nove verzije aplikacije, ki bo vključevala nekatere posodobitve in nove funkcionalnosti, katere bodo aplikacijo naredile še bolj zanimivo in uporabno. To bi sicer lahko naredili že v času razvoja prve verzije aplikacije, vendar zaradi omejenega časa razvoja tega ni bilo moč realizirati.

Aplikacijo uporabljamo že približno dva meseca in v tem času nismo našli še nobenih napak pri delovanju. Z delovanjem aplikacije smo zadovoljni, uporabljamo pa jo tako za vodenje stroškov porabe goriva kot tudi ostalih stroškov. Zadovoljni smo tudi s samo hitrostjo delovanja aplikacije. Aplikacijo trenutno uporabljata še dva druga uporabnika, ki sta s funkcionalnostmi aplikacije prav tako zadovoljna. Pogrešata edino nekatere funkcionalnosti, ki pa bodo na voljo že v naslednji verziji. Širše uporabe aplikacije zaenkrat ne načrtujemo, zato jo tudi ne bomo distribuirali (npr. na Android Marketu).

Preko podrobne predstavitve razvoja mobilne aplikacije na platformi Android v praktičnem delu diplomske naloge smo se v prvi vrsti podrobno seznanili z novimi orodji in tehnikami razvoja Android aplikacij, prav tako pa smo s tem nadgradili svoje znanje programiranja v programskem jeziku Java, saj je bil predstavljen primer razvoja Android aplikacije dokaj obsežna aplikacija AvtoDroid. Zahvaljujoč se široki zbirki različne literature in spletnih forumov na temo programiranja Android aplikacij ter tudi predhodnim izkušnjam programiranja v programskem jeziku Java, smo orodja in tehniko razvoja Android aplikacij spoznali in usvojili dokaj hitro, čeprav se zavedamo, da so v prikazanem primeru uporabljene bolj ali manj samo osnove razvoja Android aplikacij. Tudi pri pisanju teoretičnega dela

diplomske naloge smo pridobili kar nekaj novih informacij v zvezi z mobilnimi tehnologijami, ki nam do sedaj niso bile poznane. Cilj diplomske naloge, to je predstavitev razvoja mobilne aplikacije na platformi Android, je bil torej dosežen. Ker smo s tem razširili tudi naše obstoječe programersko znanje, je to zagotovo velika prednost tako pri razvoju novih mobilnih aplikacij za lastne potrebe kot tudi za potrebe drugih, saj je povpraševanje po razvoju tovrstnih aplikacij trenutno zelo veliko.

Literatura in viri

[1] M. Gargenta, *Learning Android*, Sebastopol: O'Reilly Media, 2011, poglavja 1, 2 in 3.

[2] S. Y. Hashimi, S. Komatineni, D. MacLean, *Pro Android 2*, New York: Apress, 2010, poglavji 1 in 2.

[3] U. Hribar, "Razvoj mobilnih tehnologij", *Mobilne refleksije*, Ljubljana: Fakulteta za družbene vede, 2007, poglavje 5.

[4] R. Meier, *Professional Android 2 Application Development*, Indianapolis: Wiley Publishing, 2010, poglavja 1, 2 in 3.

[5] (2012) Rupnik R., "Modeli uporabe mobilnih aplikacij v državni upravi". Dostopno na: http://bajecm.fri.uni-lj.si/CRP2001/Clanki/Indo2003.pdf.

[6] (2012) D. Tomažin, M. Pogačnik, "Razvoj in zasnova prepletene aplikacije na mobilnem sistemu Android". Dostopno na: http://www.ltfe.org/wp-content/uploads/2011/11/pogacnikrazvoj_inp.pdf.

[7] (2012) About.com, "What Makes a Smartphone Smart?". Dostopna na: http://cellphones.about.com/od/smartphonebasics/a/what_is_smart.htm.

[8] (2012) Addictivetips, "An Introduction To Modern Mobile Operating Systems". Dostopno na: http://www.addictivetips.com/mobile/an-introduction-to-modern-mobile-operating-systems.

[9] (2012) Android developers, "Activities". Dostopno na: http://developer.android.com/guide/topics/fundamentals/activities.html.

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

[11] (2012) Android developers, "What is Android? ". Dostopno na: http://developer.android.com/guide/basics/what-is-android.html.

[12] (2012) Mobitel, "Pokritost in hitrost". Dostopno na: http://www.mobitel.si/storitve/info/pokritost.aspx.

[13] (2012) Oblak Boštjan, "Portal Poraba.com" . Dostopno na: http://www.poraba.com/.

[14] (2012) Oblak Boštjan, "Mobilna Poraba.com". Dostopno na:
http://m.poraba.com.

[15] (2012) Petrol, "Trenutne cene goriv". Dostopno na:
http://www.petrol.si/api/gas_prices.xml.

[16] (2012) Smartphoneguide, "A short history of the Smartphone". Dostopno na:
http://smartphone-guide.com/a-short-history-of-the-smartphone.html.

[17] (2012) Squido, "Android OS - Operacijski sistem prihodnosti". Dostopno na:
http://www.squidoo.com/android-operacijski-sistem.

[18] (2012) Vogella, "Android Development Tutorial". Dostopno na:
http://www.vogella.de/articles/Android/article.html.

[19] (2012) Vogella, "Android SQLite Database and ContentProvider". Dostopno na:
http://www.vogella.de/articles/AndroidSQLite/article.html.

[20] (2012) Wikipedia, "Android (operating system)". Dostopno na:
http://en.wikipedia.org/wiki/Android_(operating_system).

[21] (2012) Wikipedia, "Bada". Dostopno na:
http://en.wikipedia.org/wiki/Bada.

[22] (2012) Wikipedia, "BlackBerry OS". Dostopno na:
http://en.wikipedia.org/wiki/BlackBerry_OS.

[23] (2012) Wikipedia, "Bluetooth". Dostopno na:
http://en.wikipedia.org/wiki/Bluetooth.

[24] (2012) Wikipedia, "Eclipse (software)". Dostopno na:
http://en.wikipedia.org/wiki/Eclipse_(software).

[25] (2012) Wikipedia, "IBM Simon". Dostopno na:
http://en.wikipedia.org/wiki/IBM_Simon.

[26] (2012) Wikipedia, "iOS". Dostopno na:
http://en.wikipedia.org/wiki/IOS.

[27] (2012) Wikipedia, "Mobile operating system". Dostopno na:
http://en.wikipedia.org/wiki/Mobile_operating_system.

[28] (2012) Wikipedia, "Power Designer". Dostopno na:
http://en.wikipedia.org/wiki/PowerDesigner.

[29] (2012) Wikipedia, "Smartphone". Dostopno na:
http://en.wikipedia.org/wiki/Smartphone.

[30] (2012) Wikipedia, "Symbian". Dostopno na:
http://en.wikipedia.org/wiki/Symbian.

[31] (2012) Wikipedia, "Symbian OS". Dostopno na:
http://sl.wikipedia.org/wiki/Symbian_OS.

[32] (2012) Wikipedia, "Windows Mobile". Dostopno na:
http://en.wikipedia.org/wiki/Windows_Mobile.

[33] (2012) Wikipedia, "Windows Phone". Dostopno na:
http://en.wikipedia.org/wiki/Windows_Phone.

[34] (2012) Wikipedia, "Wireless LAN". Dostopno na:
http://en.wikipedia.org/wiki/Wireless_LAN.