Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Klemen Modic

Optimizacija kontrole stanja v marini s pomočjo pametnih telefonov

diplomsko delo na univerzitetnem študiju

Mentor: prof. dr. Saša Divjak

Ljubljana, 2012

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

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: 01880/2012
Datum: 05.11.2012

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

Kandidat: **KLEMEN MODIC**

Naslov: **OPTIMIZACIJA KONTROLE STANJA V MARINI S POMOČJO PAMETNIH TELEFONOV**

OPTIMIZATION OF MARINA STATUS MANAGEMENT USING SMART PHONES

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Opišite razvoj mobilne aplikacije MarinaMaster za upravljanje marine. Najprej opišite obstoječo situacijo upravljanja marin v podjetju IRM d. o. o, za katerega naj bi bila aplikacija razvita. Predstavite zahteve podjetja in pričakovane rezultate. Predstavite programska jezika Basic4Android in PHP ter razvojni orodji SQLite Manager
in Basic4Android. Opišite operacijski sistem Android, na katerem naj bi tekla aplikacija. Razložite podatkovni model in opišite sam proces razvoja aplikacije prek grafičnega uporabniškega vmesnika in logike. Podajte opis sinhronizacije z glavno podatkovno bazo preko omrežja.

Mentor:

prof. dr. Saša Divjak

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Dekan:

prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Klemen Modic, z vpisno številko **63080193**, sem avtor diplomskega dela z naslovom:

Optimizacija kontrole stanja v marini s pomočjo pametnih telefonov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Saše Divjaka
- 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 29. novembra 2012 Podpis avtorja:

Zahvala

Rad bi se zahvalil mentorju prof. dr. Saši Divjaku in mag. Tonetu Britovšku za pomoč in usmerjanje pri diplomskem delu.
Prav tako bi se rad zahvalil celotnemu kolektivu podjetja IRM d. o. o. za izkazano pomoč in prijaznost.
Posebej bi se rad zahvalil celotni družini, ki so mi omogočili študij in me vsa leta spodbujali ter podpirali.

Kazalo

Seznam uporabljenih kratic in okrajšav

Povzetek

Abstract

1 Uvod **1**

1.1 Obstoječa situacija 2

1.2 Opis zahtev in pričakovani rezultati 3

2 Razvojno okolje **5**

2.1 Programski jeziki 5

2.2 Razvojna orodja 7

2.3 Operacijski sistem Android 17

3 Razvoj aplikacije **23**

3.1 Podatkovni model 23

3.2 Grafični uporabniški vmesnik in logika 39

3.3 Sinhronizacija prek omrežja 69

4 Zaključek **77**

Seznam uporabljenih kratic in okrajšav

IDE - Integrated Development Environment
RAD - Rapid Application Development
SDK - Software Development Kit
APK - Application Package File
SQL - Structured Query Language
PHP - PHP Hypertext Preprocessor
HTML - Hyper Text Markup Language
CSV - Comma-Separated Values
XML - Extensible Markup Language
SD - Secure Digital
PNG - Portable Network Graphics
GPS - Global Positioning System
SHA - Secure Hash Algorithm
IP - Internet Protocol

Povzetek

V diplomski nalogi je opisan razvoj mobilne aplikacije MarinaMaster za upravljanje marine. V uvodu naloge je opisana obstoječa situacija upravljanja marine v podjetju IRM, d. o. o, za katerega je bila razvita aplikacija. V nadaljevanju so predstavljene zahteve podjetja in pričakovani rezultati. Pred samim opisom postopka razvoja aplikacije sta predstavljena programska jezika Basic4Android in PHP, zatem pa še razvojni orodji SQLite Manager in Basic4Android. Temu sledi še opis operacijskega sistema Android, na katerem teče aplikacija. Predstavitvi razvojnega okolja sledi razlaga podatkovnega modela in opis samega procesa razvoja aplikacije prek grafičnega uporabniškega vmesnika in pripadajoče logike. Sledi še opis sinhronizacije z glavno podatkovno bazo prek omrežja. Na koncu diplomske naloge so navedene nastale težave pri delu, opisani končni rezultati in predstavljene možne izboljšave.

Ključne besede: Android, aplikacija, marina, Basic4Android, PHP.

Abstract

The thesis describes the development of the MarinaMaster mobile application for marina management. The introduction involves a description of the current situation regarding marina management in the company IRM d. o. o. for which the application was developed. Further on, the thesis outlines the requirements of the company and envisaged results as well as presents two programming languages, Basic4Android and PHP, and two development tools, SQLite Manager and Basic4Android. The thesis then describes the Android operating system on which the application is running. The description of the development environment is followed by an explanation of the data model and the process of application development through a graphical user interface and the corresponding logic. Synchronisation with the main database over the network is also presented. The conclusion of the thesis outlines the difficulties encountered throughout the process, presents final results and proposes possible improvements.

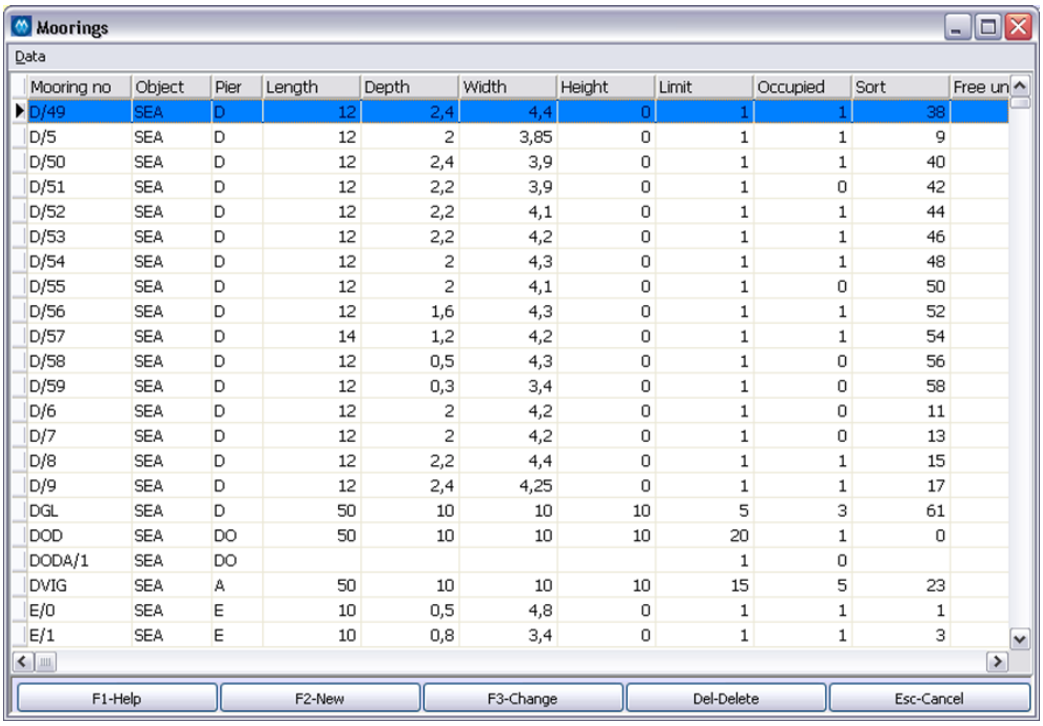
Key words: Android, application, marina, Basic4Android, PHP.

Poglavje 1

Uvod

Aplikacij, povezanih s tematiko marin ne zasledimo ravno pogosto. Posebej to velja za Slovenijo z majhnim odstotkom lastništva morskih površin. Kljub temu pa take aplikacije vseeno obstajajo. Prav tako obstajajo tudi podjetja, ki se ukvarjajo z njihovim razvojem. V Sloveniji se je na področju upravljanja marin specializiralo podjetje IRM d. o. o. z že razvito namizno aplikacijo *MarinaMaster*, ki predstavlja programsko rešitev za popolno upravljanje in kontrolo nad marino [12].

Podjetje IRM d. o. o. je bilo ustanovljeno leta 1992 in se poleg marinskega poslovanja ukvarja še z razvojem programske opreme za investicijsko bančništvo [10]. Pri podjetju sem bil štipendist, kjer sem opravljal vsakoletno obvezno prakso. V zadnjem letu prakse smo se skupaj odločili, da razvijemo aplikacijo za operacijski sistem Android, ki bi hkrati predstavljala zaokroženo celoto za diplomsko nalogo. Aplikacija naj bi bila okrnjena mobilna verzija že obstoječe aplikacije *MarinaMaster* s poudarkom na bolj preglednem grafičnem vmesniku in enostavnim upravljanjem.



Mooring no	Object	Pier	Length	Depth	Width	Height	Limit	Occupied	Sort	Free un
D/49	SEA	D	12	2,4	4,4	0	1	1	38	
D/5	SEA	D	12	2	3,85	0	1	1	9	
D/50	SEA	D	12	2,4	3,9	0	1	1	40	
D/51	SEA	D	12	2,2	3,9	0	1	0	42	
D/52	SEA	D	12	2,2	4,1	0	1	1	44	
D/53	SEA	D	12	2,2	4,2	0	1	1	46	
D/54	SEA	D	12	2	4,3	0	1	1	48	
D/55	SEA	D	12	2	4,1	0	1	0	50	
D/56	SEA	D	12	1,6	4,3	0	1	1	52	
D/57	SEA	D	14	1,2	4,2	0	1	1	54	
D/58	SEA	D	12	0,5	4,3	0	1	0	56	
D/59	SEA	D	12	0,3	3,4	0	1	0	58	
D/6	SEA	D	12	2	4,2	0	1	0	11	
D/7	SEA	D	12	2	4,2	0	1	0	13	
D/8	SEA	D	12	2,2	4,4	0	1	1	15	
D/9	SEA	D	12	2,4	4,25	0	1	1	17	
DGL	SEA	D	50	10	10	10	5	3	61	
DOD	SEA	DO	50	10	10	10	20	1	0	
DODA/1	SEA	DO					1	0		
DVIG	SEA	A	50	10	10	10	15	5	23	
E/0	SEA	E	10	0,5	4,8	0	1	1	1	
E/1	SEA	E	10	0,8	3,4	0	1	1	3	

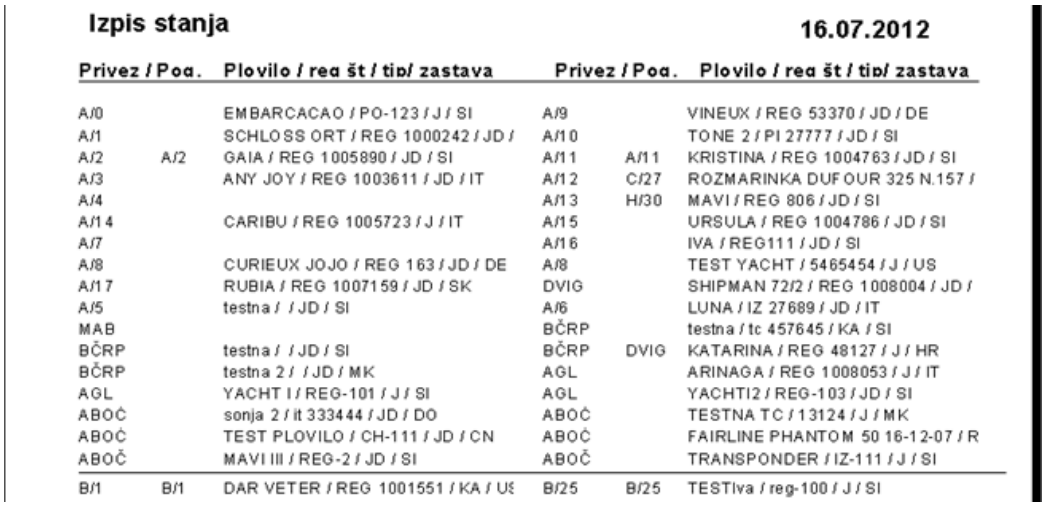
Slika 1.1: Tabela privezov z njihovimi osnovnimi podatki

1.1 Obstoječa situacija

V omenjeni aplikaciji *MarinaMaster* obstaja tabela privezov plovil, imenovana *NNPRIVEZ*. Iz nje lahko razberemo osnovne podatke o privezih, kot so: oznaka, višina, širina, globina, itd. Primer take tabele je prikazan na Sliki 1.1.

Poleg privezov obstaja tudi tabela plovil, imenovana *PLOVILA*, ki vsebuje osnovne podatke o plovilih. Za povezavo plovil in privezov je definirana še tretja tabela *PRIVEZI*. Eno plovilo se lahko namreč razteza čez več privezov, na enem privezu pa je lahko več kot eno plovilo.

Na recepciji določene marine se vsak dan izpiše seznam vseh privezov s pripadajočimi plovili. Primer takega izpisa je prikazan na Sliki 1.2.



Izpis stanja 16.07.2012

Privez / Poa.		Plovilo / reg št / tip/ zastava	Privez / Poa.		Plovilo / reg št / tip/ zastava
A/0		EMBARCACAO / PO-123 / J / SI	A/9		VINEUX / REG 53370 / JD / DE
A/1		SCHLOSS ORT / REG 1000242 / JD /	A/10		TONE 2 / PI 27777 / JD / SI
A/2	A/2	GAIA / REG 1005890 / JD / SI	A/11	A/11	KRISTINA / REG 1004763 / JD / SI
A/3		ANY JOY / REG 1003611 / JD / IT	A/12	C/27	ROZMARINKA DUFOUR 325 N.157 /
A/4			A/13	H/30	MAVI / REG 806 / JD / SI
A/14		CARIBU / REG 1005723 / J / IT	A/15		URSULA / REG 1004786 / JD / SI
A/7			A/16		IVA / REG111 / JD / SI
A/8		CURIEUX JOJO / REG 163 / JD / DE	A/8		TEST YACHT / 5465454 / J / US
A/17		RUBIA / REG 1007159 / JD / SK	DVIG		SHIPMAN 72/2 / REG 1008004 / JD /
A/5		testna / / JD / SI	A/6		LUNA / IZ 27689 / JD / IT
MAB			BČRP		testna / tc 457645 / KA / SI
BČRP		testna / / JD / SI	BČRP	DVIG	KATARINA / REG 48127 / J / HR
BČRP		testna 2 / / JD / MK	AGL		ARINAGA / REG 1008053 / J / IT
AGL		YACHT I / REG-101 / J / SI	AGL		YACHTI2 / REG-103 / JD / SI
ABOČ		sonja 2 / it 333444 / JD / DO	ABOČ		TESTNA TC / 13124 / J / MK
ABOČ		TEST PLOVILO / CH-111 / JD / CN	ABOČ		FAIRLINE PHANTOM 50 16-12-07 / R
ABOČ		MAVI III / REG-2 / JD / SI	ABOČ		TRANSPONDER / IZ-111 / J / SI
B/1	B/1	DAR VETER / REG 1001551 / KA / US	B/25	B/25	TESTIva / reg-100 / J / SI

Slika 1.2: Izpis stanja vseh privezov

Pregled marine poteka tako, da delavec marine vzame izpis stanja, se sprehodi po pomolih in primerja dejansko situacijo z izpisom ter vpiše morebitne pripombe. Delavec torej preverja, če se dejansko stanje sklada s stanjem v aplikaciji. V izpis stanja vpiše morebitne poškodbe na privezih, pripombe in svoja opažanja. Tega po končanem pregledu odnese nazaj na recepcijo, kjer se v računalnik vnesejo potrebni popravki in beležke.

Podatki o pregledu plovil se beležijo v tabelo *PREGLEDI*, pri tem pa se uporablja šifrant vrst opomb *NSTANJE*. Slikovni material za plovila se dodaja v tabelo *DATOTEKE_PLOVIL*, beležke plovil pa se zapisujejo v tabelo *PLOVILABELEZKE*. Podatki o pregledu privezov se zapisujejo v tabelo *PRIVEZIBELEZKE*.

1.2 Opis zahtev in pričakovani rezultati

Glavni cilj aplikacije je omogočiti delavcu v marini pregled in urejanje situacije na privezih prek mobilnega telefona. V nadaljevanju sledijo opisi osnovnih in dodatnih zahtev za končni produkt.

Osnovne zahteve:

1. Izris trenutnega stanja na privezih
2. Pregled privezov in pripadajočih plovil. Pregled informacij ali je privez prost ali zaseden.
3. Možnost potrjevanja pravilnosti situacije na posameznem privezu. Vnašajo se samo odstopanja od realne situacije
4. Možnost vpogleda v osnovne podatke privezov in plovil ter možnost iskanja plovil
5. Možnost vnosa beležk za priveze in plovila
6. Možnost vnosa komentarjev na pregled plovil in ustvarjanja slikovnih priponk
7. Možnost vnosa stanj števcev za vodo in elektriko ter ustvarjanja slikovnih priponk

Dodatne zahteve:

1. Možnost prijave v sistem
2. Podpora večjezičnosti (angleščina, slovenščina, hrvaščina)
3. Omogočiti delo v nepovezanem načinu (angl. offline) in obojestransko sinhronizacijo na željo uporabnika
4. Vse spremembe se morajo zapisati v obstoječo centralno Oracle-ovo podatkovno bazo

Poglavje 2

Razvojno okolje

V tem poglavju bodo opisani uporabljeni programski jeziki, razvojna orodja in operacijski sistem Android.

2.1 Programski jeziki

Programski jezik Basic4Android sem uporabil za razvoj celotne aplikacije, z jezikom PHP pa sem spisal strežniške skripte za potrebe sinhronizacije. V nadaljevanju sledi opis obeh uporabljenih programskih jezikov.

2.1.1 Basic4Android

Basic4Android je visokonivojski računalniški programski jezik, podoben jeziku Visual Basic, le da je skoncentriran na razvoj aplikacij Android in omogoča dodatno podporo za objekte [3].

Sintakso jezika lahko ob poznavanju kateregakoli drugega programskega jezika hitro osvojimo, saj je relativno enostavna za učenje in uporabo. Bralec si jo lahko ogleda v [14]. V nadaljevanju bom opisal programski tok oz. tok izvajanja aktivnosti.

```
1  'Activity module
2  Sub Process_Globals
3    'These global variables will be declared once when the application starts.
4    'These variables can be accessed from all modules.
5
6  End Sub
7
8  Sub Globals
9    'These global variables will be redeclared each time the activity is created
10   'These variables can only be accessed from this module.
11
12 End Sub
13
14 Sub Activity_Create(FirstTime As Boolean)
15
16 End Sub
17
18 Sub Activity_Resume
19
20 End Sub
21
22 Sub Activity_Pause (UserClosed As Boolean)
23
24 End Sub
25
```

Slika 2.1: Programski tok po rutinah

Programski tok

Ko zaženemo aplikacijo in s tem začetno aktivnost, gre programski tok po rutinah, prikazanih na Sliki 2.1 in sicer od zgoraj navzdol. To je hkrati najmanjši del kode, ki jo lahko vsebuje neka aktivnost.

1. *Process_Globals*: Rutina je namenjena deklaraciji procesnih globalnih spremenljivk. Te so veljavne skozi celoten proces in so dostopne iz poljubne lokacije v programu.

2. *Globals*: Rutina je namenjena deklaraciji globalnih spremenljivk aktivnosti. Te so veljavne le toliko časa, dokler obstaja aktivnost. Dostopne so le znotraj aktivnosti.

3. *Activity_Create(FirstTime As Boolean)*: Rutina je namenjena iniciali-

zaciji spremenljivk aktivnosti. Če želimo neko spremenljivko inicializirati samo enkrat po zagonu aplikacije, lahko uporabimo parameter *FirstTime*.

4. *Activity_Resume*: Ta rutina je klicana kadarkoli zaženemo aktivnost (po *Activity_Create*), jo nadaljujemo ali re-aktiviramo.

5. *Activity_Pause(UserClosed As Boolean)*: Ta rutina je klicana ob prekinitvi aktivnosti. To se lahko zgodi ob pritisku na gumb *Nazaj (angl. Back)*, spremembi orientacije, ali ob zagonu druge aktivnosti.

2.1.2 PHP

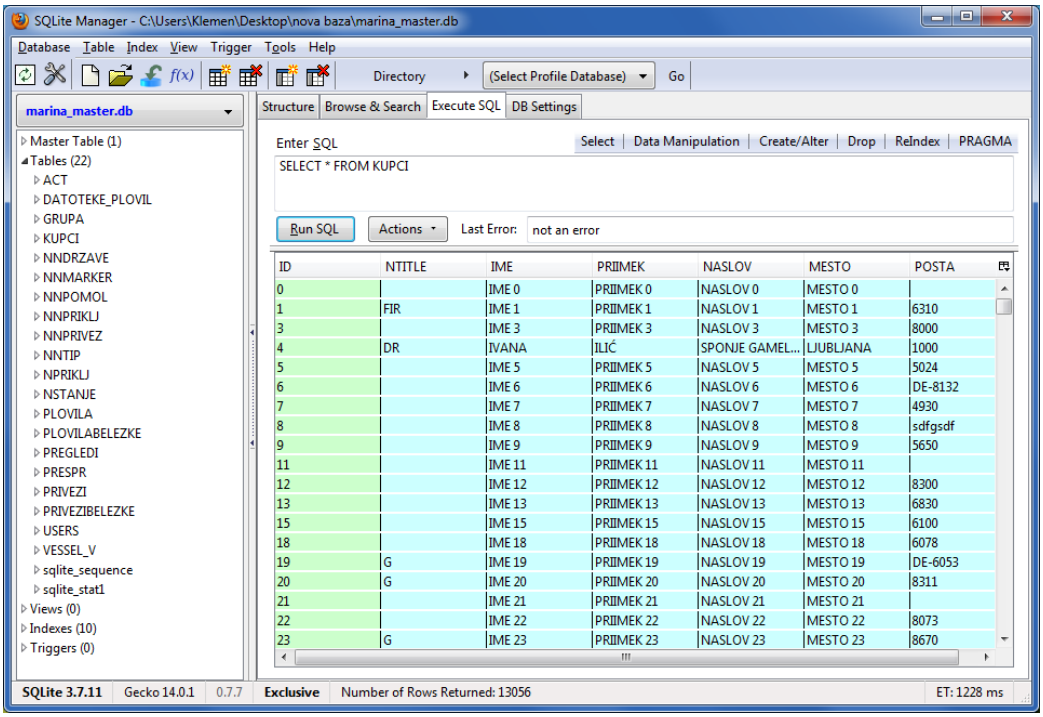
PHP je široko razširjen strežniški skriptni jezik, ki je pogosto zapisan v povezavi s HTML-jem. S kodo PHP v skriptu lahko poizvedujemo po zbirkah podatkov, izdelujemo slike, beremo ali zapisujemo datoteke, komuniciramo z oddaljenimi strežniki, itd. Jezik je odprtega izvora in nam omogoča hiter razvoj aplikacij ter visoko učinkovitost delovanja. Načrtovan je tako, da se lahko izvaja v mnogih operacijskih sistemih in je združljiv z mnogimi strežniki in zbirkami podatkov. Bralec si lahko ogleda celotno dokumentacijo PHP v [4].

2.2 Razvojna orodja

2.2.1 Vtičnik SQLite Manager za brskalnik Mozilla Firefox

Za delo z lokalno podatkovno bazo SQLite sem uporabljal orodje SQLite Manager verzije 0.7.7. Orodje je bilo razvito kot dodatek (angl. add-on) za spletni brskalnik Mozilla Firefox, ki si ga lahko brezplačno prenesemo z naslova [2]. Grafični izgled tega orodja lahko vidimo na Sliki 2.2.

Za to orodje sem se odločil, ker ponuja največ funkcionalnosti za delo s



Slika 2.2: Grafični izgled orodja SQLite Manager

podatkovno bazo SQLite od vseh ostalih orodij, ki sem jih imel možnost preizkusiti. Orodje nam omogoča kreiranje nove podatkovne baze ali povezavo z že obstoječo bazo. Prav tako omogoča operacije nad tabelami, indeksi, pogledi in prožilci (angl. triggers). Nad kreiranimi objekti nam orodje nudi pregled in izvajanje SQL stavkov. Orodje ima vgrajeno tudi funkcionalnost za izvoz in uvoz podatkovne baze, kar mi je prišlo še posebej prav pri prenašanju podatkov iz Oracle-ove centralne podatkovne baze v lokalno podatkovno bazo. Uvoz je možen v formatih CSV, SQL in XML.

2.2.2 Basic4Android

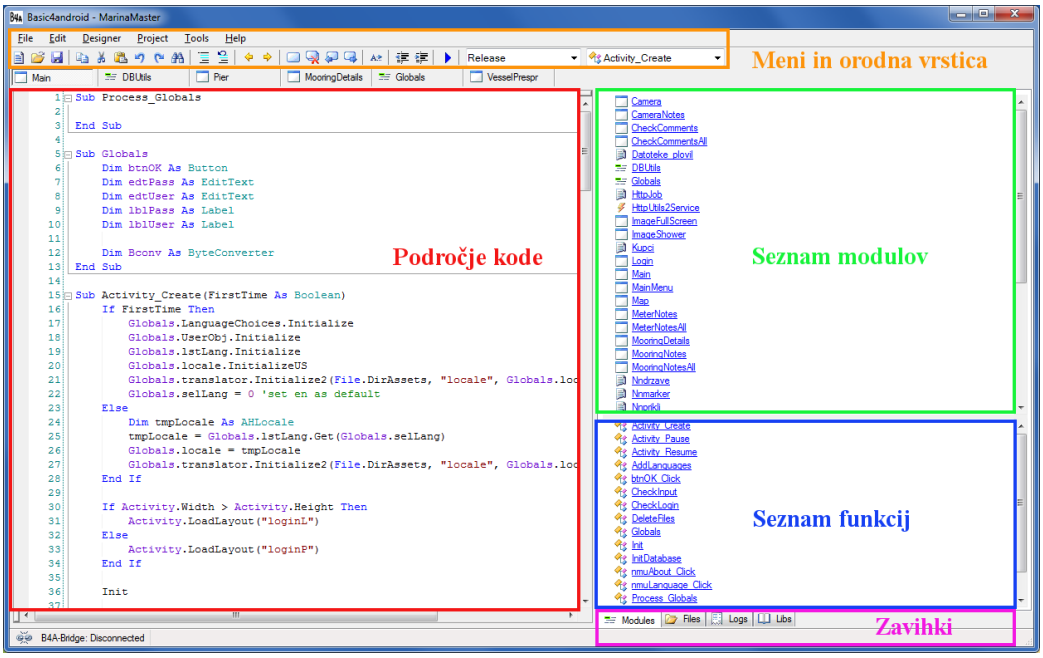
Aplikacijo sem razvijal v razvojnem okolju Basic4Android, ki ga razvija podjetje Anywhere Software. Domača spletna stran [6] ga opisuje kot najlažje in najbolj zmogljivo orodje RAD za platformo Android. Je precej novo razvojno orodje na trgu, ki se stalno razvija. V času pisanja diplomske naloge sem uporabljal verzijo 2.22. Za razliko od ostalih okolij IDE, je to 100% usmerjeno v razvoj aplikacij Android. Bralec si lahko celotno dokumentacijo o tem razvojem okolju ogleda v [3].

Polna funkcionalnost tega orodja je mogoča le z nabavo licence, kjer standardna stane 49 USD in vključuje dva meseca brezplačnih nadgradenj. Možni sta še dve licenci, poimenovani Enterprise in Site. Ti dve ponujata več let brezplačnih nadgradenj in licence za več razvijalcev ter so posledično dražje. Osebno sem kupil akadamsko licenco, ki stane enako kot standardna.

Izgled orodja in opis funkcionalnosti

Slika 2.3 prikazuje izgled razvojnega orodja Basic4Android ob prvem zagonu. Orodje je strukturirano na podoben način kot ostala razvojna orodja in vsebuje vse potrebne gradnike za hiter razvoj aplikacij.

V zgornjem meniju lahko najdemo šest zavihkov, med katerimi prva dva (*File* in *Edit*) ponujata običajne funkcionalnosti, kot smo jih navajeni že iz



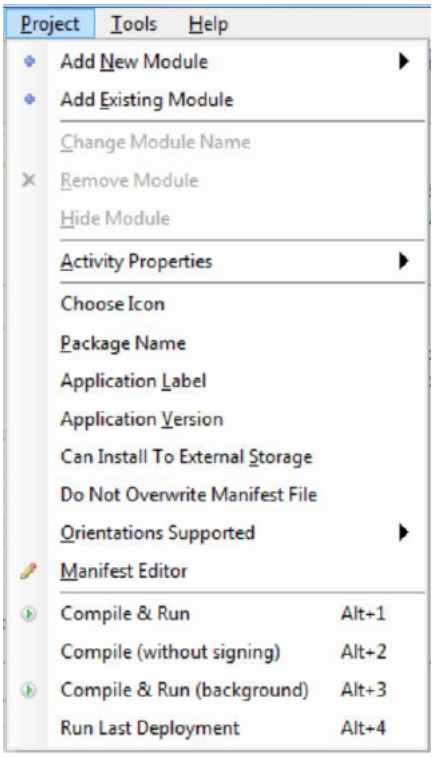
Slika 2.3: Razvojno orodje Basic4Android

ostalih razvojnih okolij. Te funkcionalnosti obsegajo odpiranje dokumenta, shranjevanje, izvoz, tiskanje, tekstovno urejanje, iskanje, komentiranje, itd.

Bolj zanimiv je zavihek *Project*, ki ob kliku nanj razkrije možnosti, prikazane na Sliki 2.4. V nadaljevanju bom opisal njihovo funkcionalnost.

Možnost *Add New Module* nam omogoča dodajanje novih modulov oz. gradnikov s katerimi sestavljamo aplikacijo. Na izbiro imamo 4 različne module in sicer:

- *Modul za aktivnost (angl. activity module)*, ki nam omogoča dodajanje novih zaslonskih mask.
- *Modul za razred (angl. class module)*, ki nam omogoča ustvarjanje novih razredov iz katerih lahko potem naredimo primerke.
- *Modul za kodo (angl. code module)*, ki služi kot vsebovalnik navadne



Slika 2.4: Nabor možnosti ob kliku na zavihek *Project*

kode in ponavadi vsebuje uporabniško definirane funkcije.

- *Modul za storitve (angl. service module)*, ki nam omogoča dodajanje funkcionalnosti, ki se izvajajo kot storitve v ozadju aplikacije.

Z izbiro možnosti *Add Existing Module* lahko dodamo obstoječi modul, z naslednjimi tremi možnostmi pa lahko spremenimo ime modula (*Change Module Name*), ga odstranimo (*Remove Module*) ali pa samo skrijemo (*Hide Module*). Moduli, ki jih urejamo, so namreč vidni kot zavihki pod orodno vrstico in jih lahko poljubno skrijemo.

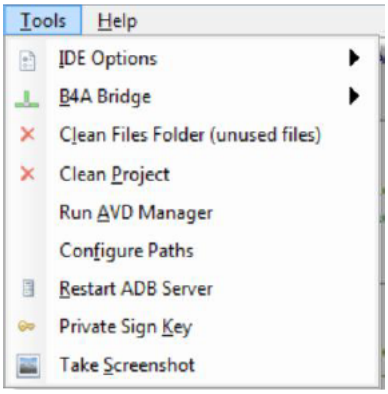
Možnost *Activity properties* nam za trenutno izbrano aktivnost omogoča določiti ali bo ta prikazana čez cel zaslon in ali naj vsebuje naslovno vrstico.

Prek možnosti *Choose Icon* lahko izberemo prikazno ikono aplikacije, prek *Package Name* pa določimo ime paketa za aplikacijo. Z možnostjo *Application Label* določimo ime aplikacije, ki bo prikazano poleg izbrane ikone, z možnostjo *Application Version* pa določimo njeno verzijo.

Prek možnosti *Can Install to External Storage* lahko določimo ali naj se aplikacija zapiše na zunanji prostor, ki ga običajno predstavlja kartica SD. Če ta možnost ni izbrana, se aplikacija naloži v notranji pomnilnik telefona.

Možnost *Do Not Overwrite Manifest File* ni priporočena za uporabo in v zadnji verziji ostaja samo zaradi ohranjanja kompatibilnosti za nazaj. Sedaj je priporočena uporaba urejevalnika manifesta, ki ga lahko pokličemo s klikom na možnost *Manifest Editor*. Tudi te možnosti se redko poslužujemo, saj je ena izmed glavnih prednosti Basic4Android-a ravno v tem, da pisanje dokumentov XML ni več potrebno. Le v redkih primerih se zgodi, da je to še vedno potrebno.

Z možnostjo *Orientations Supported* lahko za trenutno aktivnost določamo



Slika 2.5: Nabor možnosti ob kliku na zavihek *Tools*

njeno postavitev, ki je lahko ležeča, pokončna ali obe. To je lahko koristno pri aktivnostih, za katere nočemo, da so prikazane v določeni postavitvi. Primer take aktivnosti je aktivnost za kamero, kjer je ležeča postavitev edina smiselna.

Z izbiro možnosti *Compile&Run* orodje Basic4Android prevede program in ga naloži na telefon.

Med menijskimi zavihki bom opisal le še zavihek *Tools*, ki ob kliku nanj razkrije možnosti, prikazane na Sliki 2.5.

Prek možnosti *IDE Options* dobimo seznam naslednjih možnosti:

- *Tab Size*, s katero lahko nastavimo velikost presledka.
- *Change Font*, s katero lahko spremenimo tipografske značilnosti pisave v urejevalniku.
- *Word Wrap*, s katero lahko vključimo lomljenje vrstic v urejevalniku.
- *Auto Save*, s katero lahko vključimo avtomatsko shranjevanje programa, ko ga enkrat prevedemo in poženemo.

- *Show Tooltips During Typing*, s katero lahko vključimo prikaz namigov pri pisanju kode.
- *Configure Process Timeout*, s katero lahko nastavimo število sekund, preden nas program opozori, da proces nalaganja na telefon traja predolgo.

Možnost *B4A Bridge* nam ob kliku ponudi dodatne možnosti za delo s vgrajenim komunikacijskim mostom B4A-Bridge. Ta nam omogoča povezovanje s telefonom prek brezžične ali modrozobe povezave (angl. Bluetooth). V obeh primerih moramo na telefon namestiti odjemalca B4A-Bridge, ki ga lahko brezplačno prenesemo z naslova [8]. Uporaba je sila preprosta in je opisana v nadaljevanju.

V primeru povezave z brezžično povezavo (možnost *Connect - Wireless*) se moramo povezati z računalnikom, na odjemalcu klikniti na *Start Wireless* in nato v pojavnem oknu Basic4Android-a vnesti naslov IP, ki nam ga je generiral odjemalec. Na koncu kliknemo gumb *Connect*.

V primeru povezave prek povezave Bluetooth pa v odjemalcu kliknemo na možnost *Connect - Bluetooth* in nato v pojavnem oknu Basic4Android-a kliknemo na možnost *Find Devices*. Ko orodje najde napravo, jo izberemo in kliknemo na gumb *Connect*.

Možnost *Clean Files Folder* izbriše datoteke, ki smo jih vključili v imenik *Files* (prek spodnjih zavihkov) in jih naš projekt ne uporablja.

Možnost *Clean Project* zbriše vse datoteke, ki so bile ustvarjene med prevajanjem programa.

Možnost *Run AVD Manager* odpre ločen program *Android Virtual Device Manager*, s katerim lahko dodajamo in urejamo virtualne naprave Android. To pride zelo koristno v primeru, ko testiramo različne postavitve grafičnih

elementov na različne velikosti zaslonov.

Prek možnosti *Configure Paths* lahko nastavimo pot do javanskega prevajalnika *javac.exe*, *android.jar*-a in pa dodatnih knjižnjic, ki jih uporabljamo pri razvoju aplikacije in niso del že vgrajenih knjižnic.

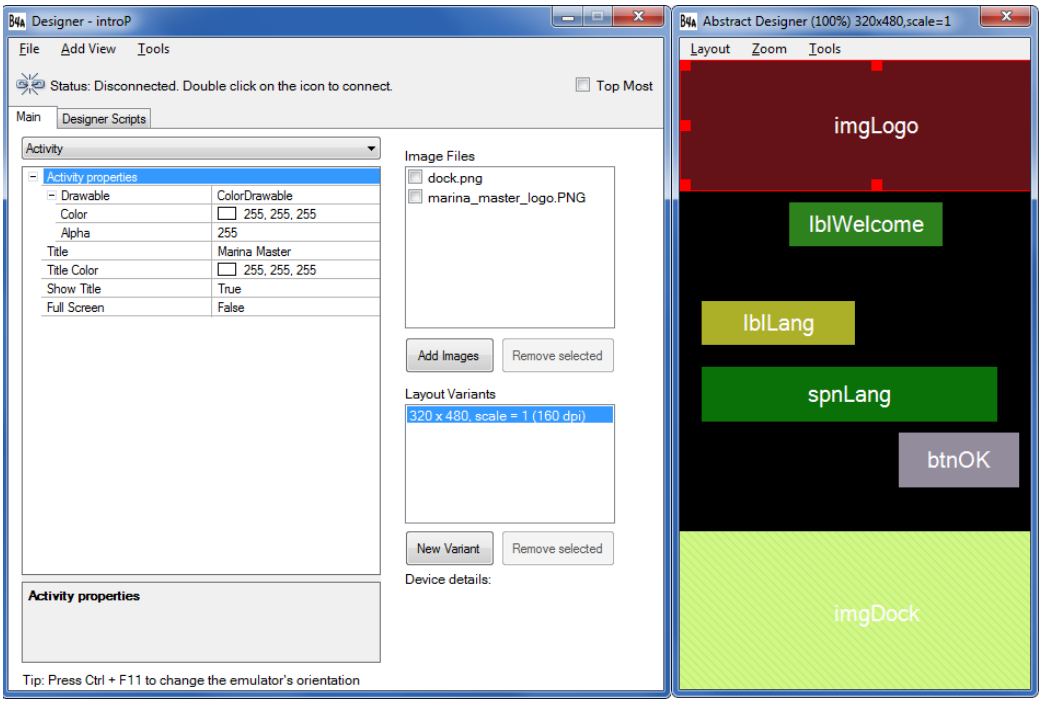
Možnost *Restart ADB Server* ponovno zažene strežnik, ki gosti virtualne naprave.

Možnost *Private Sign Key* ustvari privatni ključ z vpisanim imenom in priimkom, organizacijo ter državo.

Prek možnosti *Take Screenshot* lahko zajamemo sliko iz naprave ali emulatorja.

Poleg zavihkov v menijski vrstici, imamo štiri različne zavihke tudi v spodnjem desnem kotu razvojnega orodja in sicer:

- Zavihek *Moduli (angl. Modules)*, ki prikazuje seznam modulov in funkcij.
- Zavihek *Datoteke (angl. Files)*, ki prikazuje vse datoteke dodane v projekt. Te se nahajajo v posebnem imeniku *Files*. Omogočeno je tudi dodajanje in brisanje teh datotek.
- Zavihek *Dnevniški zapisi (angl. Logs)*, ki nam omogoča povezavo z napravo in izpis vseh komentarjev *Log*, ki jih generira program med izvajanjem. Prav tako lahko tudi vidimo potek izvajanja programa. Če izberemo možnost *Filter*, potem vidimo samo tista sporočila, ki se nanašajo na naš program. V nasprotnem primeru pa vidimo vsa sporočila, ki tečejo v sistemu.
- Zavihek *Knjižnjice (angl. Libs*), ki prikazuje knjižnjice, ki jih lahko uporabimo v našem projektu (že vgrajene in dodatne).



Slika 2.6: Grafični izgled orodja Designer in Abstract Designer

Designer in Abstract Designer

V nadaljevanju bom na kratko opisal še orodje za postavljanje grafičnih komponent v načinu *Povleci&Spusti (angl. Drag&Drop)* na izbrano podlago. Na Sliki 2.6 je prikazan grafični izgled glavnega okna *Designer* in pomožnega okna *Abstract Designer*.

Abstract Designer odpremo naknadno prek menijske vrstice, ko že zaženemo program Designer. Samo postavitev grafičnih komponent omogoča *Abstract Designer*, medtem ko *Designer* omogoča samo izbiro določenih komponent in nastavljanje njihovih lastnosti, kot npr.: ime komponente, ime dogodka, pozicija, itd. *Designer* ima še posebno okno *Designer Scripts*, v katerega lahko pišemo skripte za postavitev komponent na zaslon. Skripte ponavadi uporabimo za dinamično postavitev komponent, neodvisno od velikosti zaslona.

Koda 2.1 prikazuje del skripte, kjer v prvi vrstici nastavimo širino elementa *imgDock* na celotno širino ekrana, medtem ko v drugi vrstici nastavimo višino elementa na 30% višine zaslona. Tako dosežemo dinamičnost, saj bo element avtomatično razširjen glede na velikost zaslona, na katerem teče aplikacija.

```
1 imgDock.Width = 100%x
2 imgDock.Height = 30%y
3 imgDock.Bottom = 100%y
```

Koda 2.1: Izsek kode iz skripte *Designer*

V *Abstract Designer*-ju lahko poleg postavitve samih komponent določamo tudi velikost virtualnega zaslona, za katerega razvijamo aplikacijo. Izbiramo lahko med standardnimi velikostmi zaslonov ali pa izberemo možnost, da se velikost virtualnega zaslona prilagodi velikosti zaslona trenutno priključenega telefona.

2.3 Operacijski sistem Android

V nadaljevanju bo na kratko predstavljen operacijski sistem Android, za katerega je bila razvita aplikacija.

Splošno

Android je operacijski sistem, osnovan na Linux-u, narejen predvsem za mobilne naprave na dotik. Razvilo ga je podjetje Google v sodelovanju z konzorcijem Open Handset Alliance, ki se ukvarja z razvojem odprtih standardov za mobilne naprave. Google je izdal izvorno kodo Android-a kot odprto pod licenco Apache. Android ima zelo veliko skupnost razvijalcev, ki največkrat razvijajo v programskem jeziku Java. Razvoj aplikacij je hiter in enostaven, saj Android zagotavlja velik nabor uporabnih knjižnjic in orodij [15].

Android obravnava vse aplikacije enako, kar pomeni, da ne razlikuje med

Slika 2.7: Logotip operacijskega sistema Android

jedrnimi aplikacijami na telefonu in razvitimi aplikacijami. To pa nadaljnje pomeni, da lahko vse aplikacije dostopajo do enakih zmožnosti telefona.

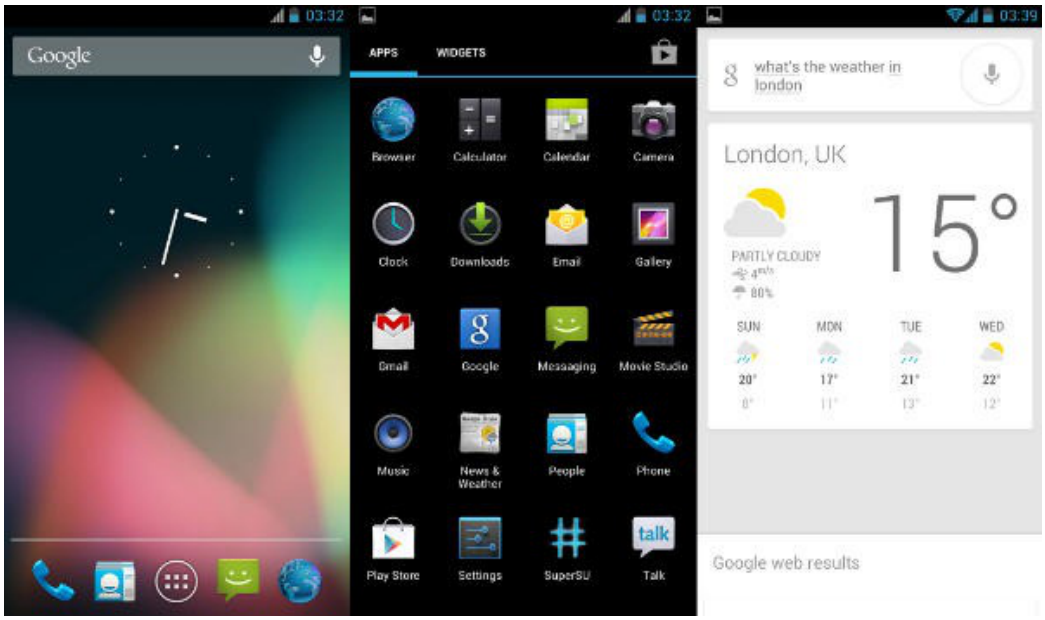
Prvi telefon T-Mobile G1 z operacijskim sistemom Android je bil prodan oktobra, leta 2008. Do leta 2010 je Android postal vodilna platforma na trgu pametnih naprav. Uporabljamo ga lahko tudi na ostalih elektronskih napravah.

Operacijski sistem Android predstavlja ikona zelenega robota, sam logotip pa je napisan v posebnem fontu, imenovanem NORAD. Logotip vsebuje napis Android, ki ga lahko vidimo na Sliki 2.7.

Vmesnik

Uporabniški vmesnik Android-a je osnovan na direktni interakciji z zaslonom prek dotikov. Poleg tega lahko pri večini telefonov uporabimo še vgrajeni meter pospeškov (angl. accelerometer) ali žiroskop (angl. gyroscope).

Same zaslonske maske so oblikovane podobno kot namizja pri navadnih računalnikih. Sestavljene so iz različnih ikon oz. bližnjic do aplikacij in pa gradnikov (angl. widget), ki predstavljajo informacije v realnem času (vreme, novice). Nad



Slika 2.8: Grafični izgled vmesnika na Android-u verzije JellyBean

namizjem je statusna vrstica, ki prikazuje informacije o napravi in povezanosti z omrežjem. Grafični izgled vmesnika zadnje verzije Android-a JellyBean, lahko vidimo na Sliki 2.8 [9].

Aplikacije

Aplikacije so ponavadi razvite v programskem jeziku Java s pomočjo Android SDK-ja, možna pa so tudi ostala razvojna okolja. Aplikacije lahko dobimo v posebnih spletnih trgovinah, kot so *Google Play* ali *Amazon Appstore*, lahko pa jih enostavno naložimo s katerekoli strani, ki vsebuje datoteko APK.

Varnost in zasebnost

Aplikacije na Android-u tečejo v posebnem peskovniku (angl. sandbox), ki predstavlja izolirano področje operacijskega sistema, ki nima dostopa do sistemskih virov. Izjemoma je to možno, če uporabnik potrdi posebna dovoljenja za dostop do teh virov, preden namesti aplikacijo. Tak pristop peskov-

Verzija	Ime	API	Delež
1.5	Cupcake	3	0.1%
1.6	Donut	4	0.3%
2.1	Eclair	7	3.1%
2.2	Froyo	8	12%
2.3 - 2.3.2	Gingerbread	9	0.3%
2.3.3 - 2.3.7		10	53.9%
3.1	Honeycomb	12	0.4%
3.2		13	1.4%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	25.8%
4.1	Jelly Bean	16	2.7%

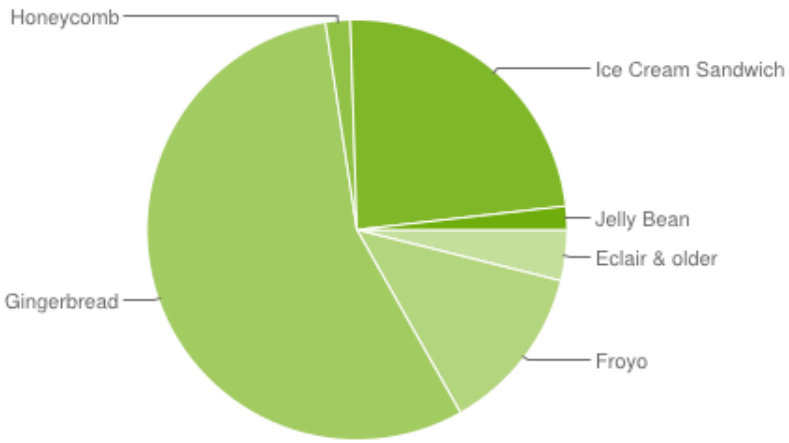
Tabela 2.1: Delež verzij Android

nika in sistema dovoljenj zmanjša verjetnost različnih vpadov ali hroščev v aplikacijah.

Delež uporabe

Android je šel skozi več verzij. Tabela 2.1 prikazuje številke in imena posameznih verzij, nivo API-ja in pa delež porazdeljenosti med uporabnike. Podatki so bili pridobljeni 1. novembra, leta 2012 [1]. V času razvoja aplikacije sem uporabljal verzijo GingerBread (2.2.3).

Slika 2.9 še grafično prikazuje porazdeljenost operacijskega sistema Android na enakih podatkih, kot jih vsebuje tabela.



Slika 2.9: Grafični prikaz deleža verzij Android

Poglavje 3

Razvoj aplikacije

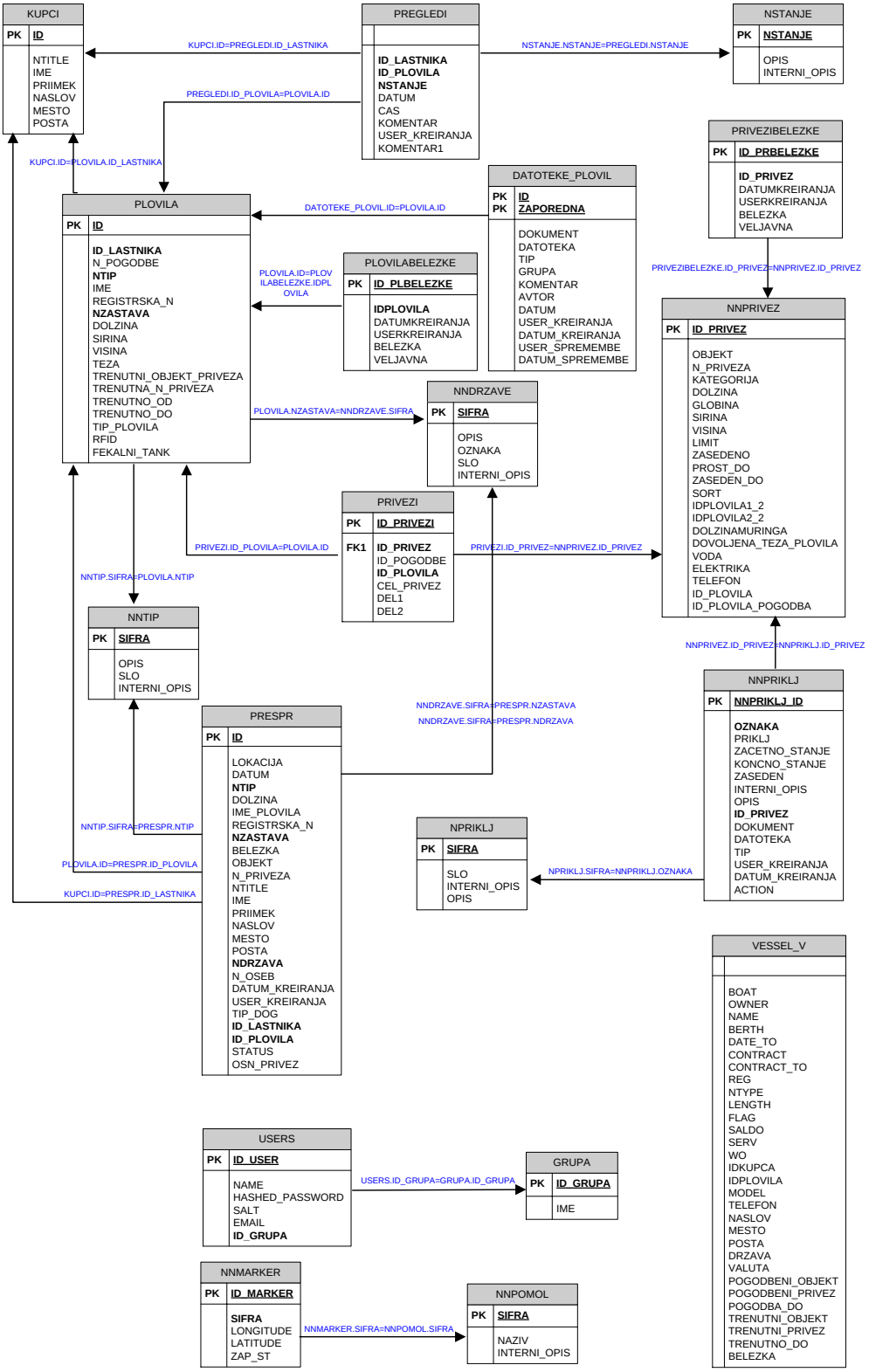
V tem poglavju bo najprej predstavljen podatkovni model z razloženimi atributi posameznih tabel. Temu bo sledil opis razvoja aplikacije po posameznih aktivnostih. Konec poglavja bo namenjen opisu sinhronizacije prek omrežja.

3.1 Podatkovni model

V podjetju IRM d. o. o. že obstaja celotni podatkovni model za namizno aplikacijo *MarinaMaster*. Obstoječa centralna podatkovna baza je Oracleova. Podatkovni model za celotno aplikacijo je kompleksen, zato smo za potrebe mobilne aplikacije vzeli le podmnožico celotnega podatkovnega modela. Prikazan je na Sliki 3.1.

Pri razvoju aplikacije sem potreboval naslednje tabele:

- **PLOVILA** Vsebuje osnovne podatke o plovilih. Za potrebe aplikacije sem potreboval naslednje stolpce:
  - ID *(integer)*: primarni ključ
  - ID_LASTNIKA *(integer)*: identifikator lastnika oz. kupca plovila (tuji ključ tabele KUPCI)



Slika 3.1: Podatkovni model za potrebe mobilne aplikacije

  - NTIP *(varchar)*: identifikator tipa plovila (tuji ključ tabele NNTIP)
  - IME *(varchar)*: ime plovila
  - REGISTRSKA_N *(varchar)*: registrska številka plovila
  - NZASTAVA *(varchar)*: identifikator države, iz katere prihaja plovilo (tuji ključ tabele NNDRZAVE)
  - DOLZINA, SIRINA, VISINA in TEZA *(varchar)*: dejanska dolžina, širina, višina in teža plovila
  - TRENUTNI_OBJEKT_PRIVEZA *(varchar)*: objekt trenutnega priveza
  - TRENUTNA_N_PRIVEZA *(varchar)*: oznaka trenutnega priveza

- **NNPRIVEZ** Vsebuje osnovne podatke o privezih. Za potrebe aplikacije sem potreboval naslednje stolpce:
  - ID_PRIVEZ *(integer)*: primarni ključ
  - OBJEKT *(varchar)*: vrsta priveza (morje, kopno, ...)
  - N_PRIVEZA *(varchar)*: enolična oznaka priveza za določeno marino
  - KATEGORIJA *(varchar)*: oznaka pomola h kateremu spada privez
  - DOLZINA, GLOBINA, SIRINA, VISINA *(varchar)*: dejanska dolžina, globina, širina in višina priveza
  - LIMIT *(integer)*: največje število plovil, ki so lahko na nekem privezu
  - ZASEDENO *(integer)*: število plovil na trenutnem privezu
  - PROST_DO *(date)*: datum do kdaj je privez prost
  - ZASEDEN_DO *(date)*: datum do kdaj je privez zaseden

  - SORT *(integer)*: dogovorjeno število, ki predstavlja lokacijo priveza na določenem pomolu

- **PRIVEZI** Povezovalna tabela med tabelo PLOVILA in tabelo NN-PRIVEZ. Potrebujemo jo zaradi razmerja mnogo : mnogo med obema tabelama. Za potrebe aplikacije sem potreboval naslednje stolpce:

  - ID_PRIVEZI *(integer)*: primarni ključ
  - ID_PRIVEZ *(integer)*: identifikator priveza (tuji ključ tabele NN-PRIVEZ)
  - ID_PLOVILA *(integer)*: identifikator plovila (tuji ključ tabele PLOVILA)

- **PLOVILABELEZKE** Vsebuje podatke o beležkah za določeno plovilo. Za potrebe aplikacije sem potreboval naslednje stolpce:

  - ID_PLBELEZKE *(integer)*: primarni ključ
  - IDPLOVILA *(integer)*: identifikator plovila, na katerega se beležka nanaša (tuji ključ tabele PLOVILA)
  - DATUMKREIRANJA *(date)*: datum kreiranja beležke
  - USERKREIRANJA *(varchar)*: ime uporabnika, ki je beležko kreiral
  - BELEZKA *(varchar)*: vpisana beležka
  - VELJAVNA *(boolean)*: označuje ali je beležka veljavna ali ne

- **PREGLEDI** Vsebuje podatke o komentarjih na pregled plovil. Za potrebe aplikacije sem potreboval naslednje stolpce:

  - ID_LASTNIKA *(integer)*: identifikator lastnika plovila (tuji ključ tabele KUPCI)
  - ID_PLOVILA *(integer)*: identifikator plovila (tuji ključ tabele PLOVILA)

  - NSTANJE *(varchar)*: identifikator stanja plovila (tuji ključ tabele NSTANJE)
  - DATUM *(date)*: datum kreiranja komentarja na pregled
  - CAS *(varchar)*: čas kreiranja komentarja na pregled
  - KOMENTAR *(varchar)*: vpisani komentar
  - USER_KREIRANJA *(varchar)*: ime uporabnika, ki je ustvaril komentar

- **DATOTEKE_PLOVIL** Vsebuje podatke o slikovnem gradivu za plovila. Za potrebe aplikacije sem potreboval naslednje stolpce:
  - ID *(integer)*: primarni ključ
  - ZAPOREDNA *(integer)*: zaporedna številka fotografije istega plovila
  - DOKUMENT *(blob)*: binarni zapis fotografije
  - DATOTEKA *(varchar)*: pot do shranjene fotografije
  - TIP *(varchar)*: tip datoteke (končnica datoteke)
  - KOMENTAR *(varchar)*: vnešen komentar poleg fotografije
  - USER_KREIRANJA *(varchar)*: ime uporabnika, ki je naredil fotografijo
  - DATUM_KREIRANJA *(date)*: datum kreiranja fotografije

- **PRIVEZI_BELEZKE** Vsebuje podatke o beležkah za določen privez. Za potrebe aplikacije sem potreboval naslednje stolpce:
  - ID_PRBELEZKE *(integer)*: primarni ključ
  - ID_PRIVEZ *(integer)*: identifikator priveza, na katerega se beležka nanaša (tuji ključ tabele NNPRIVEZ)
  - DATUMKREIRANJA *(date)*: datum kreiranja beležke

  - USERKREIRANJA *(varchar)*: ime uporabnika, ki je kreiral beležko
  - BELEZKA *(varchar)*: vpisana beležka
  - VELJAVNA *(boolean)*: označuje ali je beležka veljavna ali ne

- **PRESPR** Vsebuje podatke o sprejemu plovila v marino ali odhodu plovila iz marine. Za potrebe aplikacije sem potreboval naslednje stolpce:
  - ID *(integer)*: primarni ključ
  - DATUM *(date)*: datum kreiranja zapisa
  - DOLZINA *(varchar)*: dolžina prispelega plovila
  - IME_PLOVILA *(varchar)*: ime prispelega plovila
  - REGISTRSKA_N *(varchar)*: registrska številka prispelega plovila
  - NZASTAVA *(varchar)*: identifikator države, iz katere prihaja plovilo (tuji ključ tabele NNDRZAVE)
  - BELEZKA *(varchar)*: beležka za prispelo plovilo
  - OBJEKT *(varchar)*: objekt priveza za plovilo
  - N_PRIVEZA *(varchar)*: enolična oznaka priveza
  - IME *(varchar)*: ime lastnika prispelega plovila
  - PRIIMEK *(varchar)*: priimek lastnika prispelega plovila
  - NASLOV *(varchar)*: naslov lastnika prispelega plovila
  - MESTO *(varchar)*: mesto lastnika prispelega plovila
  - POSTA *(varchar)*: pošta lastnika prispelega plovila
  - NDRZAVA *(varchar)*: identifikator države, iz katere prihaja lastnik prispelega plovila
  - N_OSEB *(varchar)*: število oseb za katero je plovilo registrirano
  - DATUM_KREIRANJA *(date)*: datum kreiranja zapisa v to tabelo
  - USER_KREIRANJA *(varchar)*: ime uporabnika, ki je zapis kreiral

  - TIP_DOG *(varchar)*: tip dogodka. Ločimo 4 različne dogodke in sicer PP (prihod pogodbenega plovila), OP (odhod pogodbenega plovila), PT (prihod tranzitnega plovila) in OT (odhod tranzitnega plovila)
  - ID_LASTNIKA *(integer)*: identifikator lastnika odhajajočega plovila (tuji ključ tabele KUPCI)
  - ID_PLOVILA *(integer)*: identifikator odhajajočega plovila (tuji ključ tabele PLOVILA)
  - STATUS *(varchar)*: poseben identifikator, ki označuje, da gre za zapis z naprave Android
  - OSN_PRIVEZ *(varchar)*: osnovni privez. Prispelo plovilo se lahko namreč razteza čez več privezov in v tem primeru OSN_PRIVEZ označuje privez na katerega je uporabnik kliknil, N_PRIVEZA pa vse ostale priveze, čez katere se plovilo razteza

- **NNPRIKLJ** Vsebuje podatke o števcih za priveze. Za potrebe aplikacije sem potreboval naslednje stolpce:
  - NNPRIKLJ_ID *(integer)*: primarni ključ
  - OZNAKA *(varchar)*: oznaka števca (EL, WT, ...)
  - PRIKLJ *(integer)*: številka priključka števca
  - ZACETNO_STANJE *(varchar)*: začetno stanje števca (ponavadi kar 0)
  - KONCNO_STANJE *(varchar)*: končno stanje števca
  - ZASEDEN *(boolean)*: označuje ali je števec zaseden ali ne
  - INTERNI_OPIS *(varchar)*: interni opis števca za potrebe podjetja
  - OPIS *(varchar)*: opis števca
  - ID_PRIVEZ *(integer)*: identifikator priveza, na katerega se števec nanaša (tuji ključ tabele NNPRIVEZ)

  - DOKUMENT *(blob)*: binarni zapis fotografije priveza
  - DATOTEKA *(varchar)*: pot do shranjene fotografije priveza
  - TIP *(varchar)*: tip datoteke (končnica datoteke)
  - USER_KREIRANJA *(varchar)*: ime uporabnika, ki je naredil fotografijo
  - DATUM_KREIRANJA *(date)*: datum kreiranja fotografije
  - ACTION *(varchar)*: številka določene akcije. Uporabnik lahko namreč za neko stanje lahko naredi samo vpis stanja, samo fotografijo, ali pa oboje.

- **VESSEL_V** Pogled, ki je bil ustvarjen nad različnimi tabelami. Vsebuje veliko podobnih podatkov, ki bi jih lahko dobili tudi iz drugih tabel. Nekaj podatkov pa je takih, da bi za njihov izračun potrebovali več tabel (primer takega podatka je SALDO). Zato smo se odločili, da v primerih, ko potrebujemo izračunane podatke, uporabimo podatke iz pogleda. Za potrebe aplikacije sem potreboval naslednje stolpce:

  - IDKUPCA *(integer)*: identifikator lastnika plovila
  - IDPLOVILA *(integer)*: identifikator plovila
  - BOAT *(varchar)*: ime plovila
  - OWNER *(varchar)*: priimek lastnika plovila
  - NAME *(varchar)*: ime lastnika plovila
  - BERTH *(varchar)*: oznaka priveza
  - REG *(varchar)*: registrska številka plovila
  - NTYPE *(varchar)*: tip plovila
  - LENGTH *(varchar)*: dolžina plovila
  - FLAG *(varchar)*: ime države, iz katere prihaja plovilo
  - SALDO *(varchar)*: saldo

  - SERV *(boolean)*: označuje ali ima lastnik odprte storitve
  - WO *(boolean)*: označuje ali ima lastnik delovne naloge
  - TELEFON *(varchar)*: telefonska številka lastnika plovila
  - NASLOV *(varchar)*: naslov stalnega prebivališča lastnika plovila
  - MESTO *(varchar)*: mesto stalnega prebivališča lastnika plovila
  - POSTA *(varchar)*: pošta stalnega prebivališča lastnika plovila
  - DRZAVA *(varchar)*: ime države, iz katere prihaja lastnik
  - POGODBENI_OBJEKT *(varchar)*: označuje, če ima lastnik pogodbo za privez

- **KUPCI** Vsebuje osnovne podatke o lastnikih plovil. Za potrebe aplikacije sem potreboval naslednje stolpce:

  - ID *(integer)*: primarni ključ
  - IME *(varchar)*: ime lastnika
  - PRIIMEK *(varchar)*: priimek lastnika
  - NASLOV *(varchar)*: stalni naslov lastnika
  - MESTO *(varchar)*: mesto stalnega prebivališča lastnika
  - POSTA *(varchar)*: pošta mesta, v katerem prebiva lastnik

- **USERS** Vsebuje podatke o uporabnikih, ki lahko uporabljajo aplikacijo. Ustvarjena je bila naknadno in ni bila del že obstoječega podatkovnega modela. Za potrebe aplikacije sem potreboval naslednje stolpce:

  - ID_USER *(integer)*: primarni ključ
  - NAME *(varchar)*: uporabniško ime
  - HASHED_PASSWORD *(varchar)*: zgoščena vrednost SHA-1 gesla uporabnika

  - SALT *(varchar)*: naključno generirana vrednost za preprečitev napadov na gesla z grobo silo (angl. brute force attack)
  - EMAIL *(varchar)*: elektronski naslov uporabnika
  - ID_GRUPA *(integer)*: rezerviran stolpec za bodočo uporabo v primeru dodelitve vlog za aplikacijo

- **NPRIKLJ** Šifrant števcev z opisi v različnih jezikih. Za potrebe aplikacije sem potreboval naslednje stolpce:
  - SIFRA *(varchar)*: primarni ključ
  - SLO *(varchar)*: opis števca v slovenskem jeziku
  - INTERNI_OPIS *(varchar)*: notranji opis števca za potrebe podjetja

- **NNDRZAVE** Šifrant držav z nazivi in opisi v različnih jezikih. Za potrebe aplikacije sem potreboval naslednje stolpce:
  - SIFRA *(varchar)*: primarni ključ
  - OPIS *(varchar)*: opis države
  - OZNAKA *(varchar)*: dvomestna oznaka posamezne države
  - SLO *(varchar)*: naziv države v slovenskem jeziku
  - INTERNI_OPIS *(varchar)*: notranji opis države za potrebe podjetja

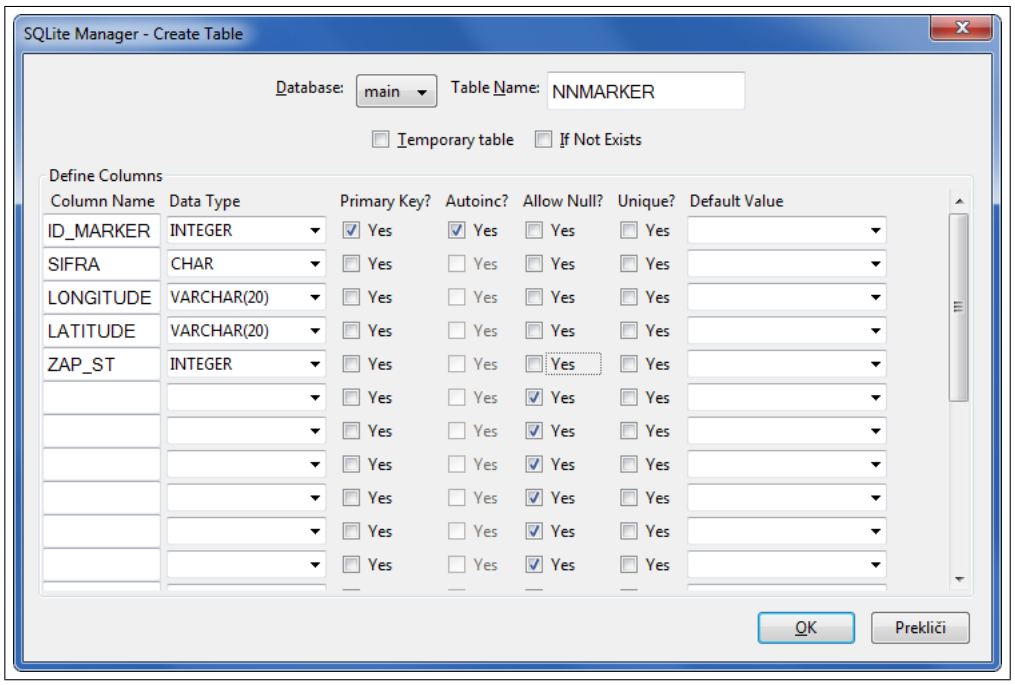
- **NNMARKER** Vsebuje osnovne podatke o označevalcih (angl. marker). Potrebujemo jo za določitev lokacije teh označevalcev na zemljevid. Za potrebe aplikacije sem potreboval naslednje stolpce:
  - ID_MARKER *(integer)*: primarni ključ
  - SIFRA *(varchar)*: oznaka pomola na katerem nahaja označevalec

  - LONGITUDE *(varchar)*: število stopinj zemljepisne dolžine lokacije označevalca
  - LATITUDE *(varchar)*: število stopinj zemljepisne širine lokacije označevalca
  - ZAP_ST *(integer)*: zaporedno število označevalca na nekem pomolu

- **NNPOMOL** Šifrant pomolov, ki vsebuje oznake pomolov določene marine. Za potrebe aplikacije sem potreboval naslednje stolpce:
  - SIFRA *(varchar)*: primarni ključ
  - NAZIV *(varchar)*: praktično enak šifri in predstavlja oznako pomola

- **NNTIP** Šifrant tipov plovila. Vsebuje opise in nazive v različnih jezikih. Za potrebe aplikacije sem potreboval naslednje stolpce:
  - SIFRA *(varchar)*: primarni ključ
  - OPIS *(varchar)*: opis tipa plovila
  - SLO *(varchar)*: opis tipa plovila v slovenskem jeziku
  - INTERNI_OPIS *(varchar)*: notranji opis tipa plovila za potrebe podjetja

- **NSTANJE** Šifrant stanj, v katerih se lahko nahaja plovilo. Za potrebe aplikacije sem potreboval naslednje stolpce:
  - NSTANJE *(varchar)*: primarni ključ
  - SLO *(varchar)*: opis stanja v slovenskem jeziku
  - OPIS *(varchar)*: opis stanja
  - INTERNI_OPIS *(varchar)*: notranji opis stanja za potrebe podjetja



Slika 3.2: Okno za kreiranje nove tabele

3.1.1 Izdelava lokalne podatkovne baze SQLite

Lokalno podatkovno bazo SQLite sem izdelal v že opisanem vtičniku *SQLite Manager* za brskalnik Mozilla Firefox. Bralec si lahko celotno dokumentacijo o SQLite ogleda v [5].

Orodje *SQLite Manager* nam nudi enostavno kreiranje tabel prek pojavnega okna, prikazanega na Sliki 3.2. Znotraj okna lahko določimo imena stolpcev tabele in pripadajoče tipe. Prav tako je možno določiti tudi primarne ključe in privzete vrednosti za stolpce.

Po postopku kreiranja tabel, je te bilo treba napolniti s testnimi podatki. V ta namen sem prek Oracle-ovega programa *SQL Developer* izvozil že obstoječe podatke v formatu s stavki INSERT. To pomeni, da program ustvari datoteko SQL, kjer so vsi podatki zapisani kot stavki INSERT in so kot taki

pripravljeni za prenos na neko drugo bazo.

Pri izvozu podatkov sem naletel na težavo pri datumih, saj je njihov format v bazah Oracle različen od formata v bazah SQLite. V ta namen sem uporabil orodje *Notepad++*, ki ponuja odlično podporo regularnim izrazom. S tem orodjem sem ustvarjene datoteke z napačnim formatom datumov pretvoril v pravilni format za podatkovno bazo SQLite. To pretvorbo lahko naredimo tako, da v orodju najprej izberemo možnost *Replace*. Ko se pojavi okno za zamenjavo nizov, moramo v možnostih za iskalni način izbrati regularni izraz (angl. regular expression).

V polje *Poišči (angl. Find what)* vpišemo naslednji regularni izraz:

```
to_date\(\'([0-9][0-9])\.([0-9][0-9])\.([0-9][0-9])\',\'DD\.MM\.RR\'\)
```

V polje *Zamenjaj z (angl. Replace with)* pa vpišemo naslednji regularni izraz:

```
'20\3-\2-\1'
```

Orodje bo v tem primeru najprej našlo vse nize, ki se začnejo z nizom *to_date*, se nadaljujejo z tremi dvomestnimi številkami, ločenimi s piko, in končajo z nizom v formatu *DD.MM.RR*. Najdene nize bo zamenjalo z datumom oz. nizom, ki se začne z dvomestno številko 20, nato pa mu sledi tretja dvomestna številka iz iskanega niza, zatem pomišljaj, nato druga dvomestna številka, zatem pomišljaj in na koncu še prva dvomestna številka iz prvotnega niza.

Bralec si lahko razlago uporabe regularnih izrazov v orodju *Notepad++* lahko ogleda v [13].

3.1.2 Izdelava razredov v Basic4Android-u

Pri razvoju aplikacije sem hotel uporabiti objektni pristop, zato sem po kreiranju lokalne podatkovne baze za nekatere tabele ustvaril razrede. Iz njih sem kasneje v programu ustvaril primerke teh razredov, kar mi je omogočilo

enostaven dostop do njihovih lastnosti. V Basic4Android-u je razrede možno ustvariti s posebnim razrednim modulom (angl. class module). V nadaljevanju bom opisal primer kreiranja razreda za tabelo *Nnmarker*, njegovo povezavo s podatkovno bazo in način dostopa do prebranih lastnosti razreda. Opisani koncept je podoben za vse ostale tabele, za katere sem ustvaril razrede.

V odseku kode 3.1 je prikazan začetni del razreda *Nnmarker* za pripadajočo istoimensko tabelo. V tem delu najdemo najavo privatnih spremenljivk oz. lastnosti razreda.

```
1 'razredni modul (angl. class module)
2 'najava privatnih spremenljivk
3 Sub Class_Globals
4   Private Id_marker As Int
5   Private Sifra As String
6   Private Longitude As String
7   Private Latitude As String
8   Private Zap_st As Int
9 End Sub
```

Koda 3.1: Lastnosti razreda *Nnmarker*

Koda 3.2 prikazuje primer nastavitvene metode (angl. set method), Koda 3.3 pa pridobitvene metode (angl. get method).

```
1 Public Sub SetLongitude(aLongitude As String)
2   Longitude=aLongitude
3 End Sub
```

Koda 3.2: Metoda SET za nastavitev lastnosti *Longitude*

```
1 Public Sub GetLongitude As String
2   Return Longitude
3 End Sub
```

Koda 3.3: Metoda GET za nastavitev lastnosti *Longitude*

Po postopku kreiranja razredov, sem te lahko uporabil v povezavi z podatkovno bazo. Primer pridobitve vseh vrstic tabele *Nnmarker* in kreiranja objektov za vsako vrstico je zapisan v kodi 3.4. Rezultat vrnemo kot seznam objektov.

```
1  Sub GetAllNnmarker As List
2    'kreiramo seznam rezultatov
3    Dim Results As List
4    Results.Initialize
5
6    Dim Cursor1 As Cursor
7    'prek kurzorja izvedemo sledečo poizvedbo
8      Cursor1 = Globals.SQL1.ExecQuery("SELECT * FROM nnmarker")
9
10   'gremo čez vse rezultate
11     For i = 0 To Cursor1.RowCount - 1
12         Cursor1.Position = i
13
14     'kreiramo začasen objekt, kateremu nastavimo lastnosti
15     'iz trenutne vrstice
16     Dim TmpNnmarkerObj As Nnmarker
17     TmpNnmarkerObj.Initialize
18     TmpNnmarkerObj.SetId_marker(Cursor1.GetInt("ID_MARKER"))
19     TmpNnmarkerObj.SetSifra(Cursor1.GetString("SIFRA"))
20     TmpNnmarkerObj.SetLongitude(Cursor1.GetString("LONGITUDE"))
21     TmpNnmarkerObj.SetLatitude(Cursor1.GetString("LATITUDE"))
22     TmpNnmarkerObj.SetZap_st(Cursor1.GetInt("ZAP_ST"))
23
24     'dodamo začasen objekt v seznam objektov
25     Results.Add(TmpNnmarkerObj)
26     Next
27     Cursor1.Close
28
29   'vrnemo seznam objektov, ki predstavlja rezultat
30   'izvršene poizvedbe
31   Return Results
32 End Sub
```

Koda 3.4: Kreiranje seznama objektov *Nnmarker*

Sedaj lahko iz poljubne lokacije v programu pokličemo metodo *GetAllNnmarker* in tako dostopamo do seznama objektov. Primer dostopa do posameznih objektov, izpisa dveh lastnosti in nastavljanja ene lastnosti je zapisan v Kodi 3.5

```
'pridobimo seznam objektov
LstNnmarker = DBUtils.GetAllNnmarker

'gremo čez celoten seznam
For i = 0 To LstNnmarker.Size - 1
  Dim tmpNnmarkerObj As Nnmarker
  'pridobimo i-ti objekt iz seznama
  tmpNnmarkerObj = LstNnmarker.Get(i)

  'sedaj lahko dostopamo do lastnosti objekta
  'izpiše lastnost Id_marker objekta na i-ti poziciji
  Log(tmpNnmarkerObj.GetId_marker)
  'izpiše lastnost Sifra objekta na i-ti poziciji
  Log(tmpNnmarkerObj.GetSifra)

  'lahko pa tudi nastavljamo lastnosti objekta
  'nastavimo zaporedno številko objekta na i-to pozicijo
  tmpNnmarkerObj.SetZapSt(i)
Next
```

Koda 3.5: Pridobitev posameznih objektov in dostop do lastnosti

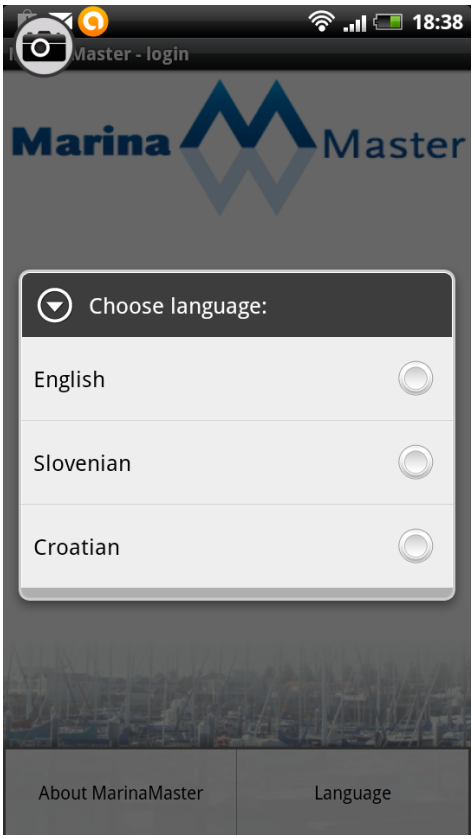
3.2 Grafični uporabniški vmesnik in logika

V nadaljevanju bo opisan postopek razvoja aplikacije po aktivnostih, kjer bom za vsako aktivnost opisal njen grafični izgled, zatem pa še logiko, ki se izvaja v ozadju.

3.2.1 Aktivnost Main

Najprej sem razvil aktivnost *Main*, prek katere je možna prijava v aplikacijo. Prikazana je na Sliki 3.3. Poleg prijave je možna tudi izbira poljubnega jezika in pregled verzije. Možnosti izbire jezika prikazuje Slika 3.4. *Main* je

Slika 3.3: Aktivnost *Main*



Slika 3.4: Aktivnost *Main* z možnostjo izbire jezika

torej prva aktivnost, ki se odpre uporabniku ob zagonu aplikacije. Vsebuje

dve vnosni polji za uporabniško ime in geslo ter potrditveni gumb. Ob kliku na menijski gumb se prikažeta možnosti *About MarinaMaster* za pregled trenutne verzije in *Language* za izbiro poljubnega jezika. Če uporabnik klikne na prvo možnost, se mu prikaže pojavno okno z informacijo o zadnji verziji aplikacije. Ob kliku na drugo možnost pa se uporabniku prikaže pojavno okno s tremi možnimi izbirami jezika. Ti jeziki so angleščina, slovenščina in hrvaščina.

Za prevajanje aplikacije je zadolžen poseben objekt *Translator*, s katerim lahko pridobimo poljuben niz v izbranem jeziku. V ta namen moramo za vsak jezik posebej definirati različne datoteke s končnico *lng*. Za slovenščino moramo tako definirati datoteko *locale_sl.lng*, za hrvaščino *locale_hr.lng*, za angleščino pa *locale_en.lng*. V teh datotekah so posamezni zapisi sestavljeni iz ključa in pripadajoče vrednosti. Ključ je zapisan v angleškem jeziku in ga v programu potrebujemo za sklicevanje na želeno vrednost. Isti ključ mora biti v vseh datotekah poimenovan z istim imenom.
Primer stavka, s katerim lahko prevajamo nize je:

```
Globals.translator.GetText("welcome").
```

Ta stavek nam glede na izbran jezik in ključ pridobi ustrezno vrednost, kar pa predstavlja prevod v izbranem jeziku. V tem konkretnem primeru bomo za jezik slovenščina dobili prevod *Dobrodošli*, za angleščino pa *Welcome*.

Če uporabnik prvič zažene to aktivnost, nastavimo angleški jezik kot privzet. Šele ko uporabnik klikne na eno od možnih izbir jezika, se prek objekta *Translator* izvede prevajanje v izbran jezik.

V tej aktivnosti prek funkcije *InitDatabase* inicializiramo podatkovno bazo. Ta nam prekopira podatkovno bazo iz projektnega imenika (angl. asset folder) na neko zapisljivo lokacijo (ponavadi zunanjo kartico SD). To naredi funkcija *DBUtils.CopyDBFromAssets*, a le če podatkovna baza še ne obstaja. V funkciji *InitDatabase* imamo tudi funkcijo *DeleteFiles*, ki izbriše predho-

dno bazo, tako da jo potem funkcija *DBUtils.CopyDBFromAssets* lahko na novo prekopira.

Uporabniki in prijava v aplikacijo

Uporabniki so bili kreirani na lokalni podatkovni bazi v tabeli *USERS*. Sol (angl. salt) je bila za obstoječe uporabnike ustvarjena z generatorjem varnih naključnih števil (angl. secure random) in je sestavljena iz 80 znakov. Geslo je zapisano kot zgoščena vrednost algoritma SHA-1 nad vpisanim geslom in soljo. Sestavljeno je iz 40 znakov [16].

Ob pritisku na potrditveni gumb *OK*, se prek funkcije *CheckInput* naprej preveri ali je uporabnik sploh vnesel kak podatek. V nasprotnem primeru ga aplikacija prijazno opozori. Ko uporabnik vpiše oba podatka, se pokliče funkcija *CheckLogin*, ki najprej preveri uporabniško ime prek metode *DBUtils.GetUserByEmail*. Če ta funkcija vrne vrednost *False*, to pomeni, da nismo našli uporabnika s tem uporabniškim imenom. V tem primeru zopet opozorimo uporabnika. Če uporabnik s tem uporabniškim imenom obstaja, preverimo še geslo. Za to potrebujemo dve dodatni knjižnjici *ByteConverter* in *Encryption*.

Preverjanje gesla je zapisano v Kodi 3.6

```
1  Dim data(0) As Byte
2  Dim md As MessageDigest
3
4  'pridobimo vpisano geslo in sol iz podatkovne baze
5  data = Bconv.StringToBytes(edtPass.Text & "" & Globals.UserObj.
       GetSalt, "UTF8")
6
7  'generiramo izvleček z algoritmom SHA-1 (geslo in sol)
8  data = md.GetMessageDigest(data, "SHA-1")
9
10 'pretvorimo v šestnajstiško vrednost
11 Dim correctPass As String
```

```
12 correctPass = Bconv.HexFromBytes(data)
13
14 'če je izračunan izvleček enak shranjenemu izvlečku
15 'na podatkovni bazi, potem je geslo pravilno
16 If Globals.UserObj.GetHashed_password.EqualsIgnoreCase(
       correctPass) Then
17   'uspešna prijava, sedaj lahko zaženemo naslednjo aktivnost
18   StartActivity(MainMenu)
19 Else 'napačno geslo
20   'neuspešna prijava
21   'opozorimo uporabnika
22   showUserPassWarn
23 End if
```

Koda 3.6: Preverjanje gesla

Ko uporabnik vnese pravilno uporabniško ime in geslo, lahko zaženemo aktivnost MainMenu, ki je opisana v nadaljevanju.

3.2.2 Aktivnost MainMenu

To je glavna aktivnost aplikacije in vsebuje bližnjice do vseh operacij, ki jih aplikacija podpira. Bralec si jo lahko ogleda na Sliki 3.5. V tej aktivnosti je prisotnih pet gumbov. Prvi služi za sinhronizacijo, drugi za iskanje plovil, tretji za urejanje privezov in plovil, četrti pa za izhod iz aplikacije. V zgornjem desnem kotu imamo še peti gumb za odjavo uporabnika iz aplikacije.

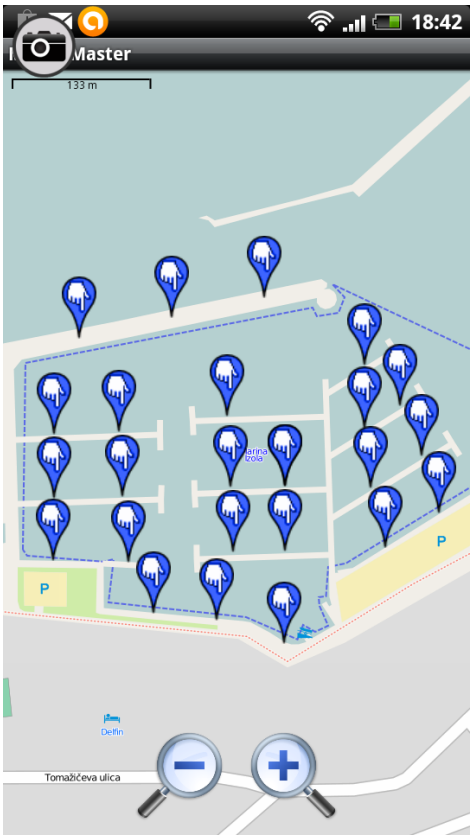
Ob kliku na gumb za iskanje plovil se uporabniku odpre aktivnost *SearchVessels*, ob kliku na gumb za urejanje privezov in plovil pa se uporabniku odpre aktivnost *Map*. V primeru klika na gumb za sinhronizacijo je zadeva malo bolj kompleksna. V enem koraku se namreč povežemo na oddaljeni strežnik, ustvarimo zahtevo za generiranje nove datoteke s stavki INSERT, preberemo arhivirano datoteko iz strežnika, jo dearhiviramo in na koncu napolnimo lokalno podatkovno bazo. Bolj podroben proces sinhronizacije bo opisan v nadaljevanju diplomske naloge. V primeru klika na gumb za iz-

Slika 3.5: Aktivnost MainMenu

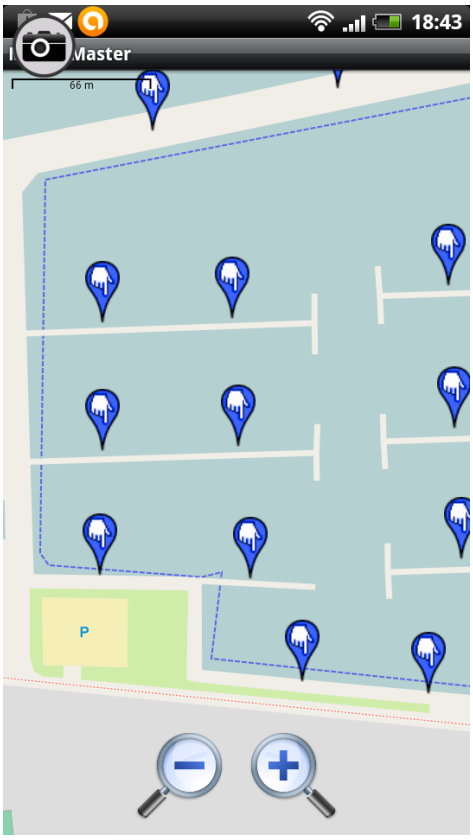
hod iz aplikacije pa najprej preverimo ali so bile narejene kakšne spremembe na lokalni podatkovni bazi. Če so bile narejene, potem vprašamo uporabnika, če jih želi zapisati v centralno podatkovno bazo. Ob potrditvi se nato spremembe zapišejo v centralno podatkovno bazo.

3.2.3 Aktivnost Map

Ta aktivnost vsebuje zemljevid marine Izola z označevalci (angl. marker) postavljenimi na pomole. Aplikacija podpira dva nivoja povečave. Pomanjšan zemljevid marine Izole lahko vidimo na Sliki 3.6, povečan zemljevid pa na Sliki 3.7. Povečavo nastavljamo prek gumbov v obliki povečevalnega stekla v spodjem delu ekrana.



Slika 3.6: Pomanjšan zemljevid marine Izola



Slika 3.7: Povečan zemljevid marine Izola

Kreiranje zemljevida

Za kreiranje zemljevida določene marine s funkcijo povečave/pomanjšave potrebujemo brezplačni program *Maperative*, ki si ga lahko prenesemo s spleta [11]. Program po prenosu enostavno zaženemo, saj ne potrebuje namestitve.

V programu se postavimo na mesto, za katerega hočemo kreirati zemljevid. Ko smo na želenem mestu, povečamo zemljevid skoraj do končne povečave in kliknemo na možnost *Tools*, zatem pa *Generate Tiles*. Ta funkcija naredi več slik PNG za več različnih povečav in jih skopira v mapo *Tiles*. V tej datoteki je več različnih map in sicer za vsak nivo povečave ena. Korensko datoteko arhiviramo s poljubnim imenom, znotraj arhiva pa nato preimenujemo mapo *Tiles* v *Mapnik*. To je nujno potrebno, da aplikacija lahko zazna zemljevide tipa *Mapnik*. Arhivirano datoteko nato premaknemo v projektni imenik (angl. asset folder).

Generiranje označevalcev

Na bazi imamo za določitev označevalcev posebno tabelo z imenom *NNMARKER*, ki je že bila opisana pri podatkovnem modelu. Stolpca *LATITUDE* in *LONGITUDE* predstavljata koordinate GPS. Našel sem jih prek spletnega zemljevida Najdi.si in sicer tako, da sem se za posamezen privez postavil na želeno lokacijo in vpisal pripadajoče koordinate. Na nek pomol lahko postavimo več označevalcev, a jih moramo ustrezno označiti. Označujemo vedno v naraščajočem vrstnem redu od kopnega proti morju. Tabelo *NNMARKER* nato uporabimo v programu za postavitev označevalcev na zemljevid.

Uporaba zemljevida v Basic4Android-u

Za uporabo zemljevida v nepovezanem načinu (angl. offline) potrebujemo dodatno knjižnjico *OSMDroid*, ki jo moramo vključiti v Basic4Android. Sama uporaba zemljevida v praksi je dokaj enostavna.

V kodi se na predhodno ustvarjeno arhivirano datoteko sklicujemo s stavkom:

```
MapView1.SetTileSource("Mapnik") //pove da gre za mape Mapnik OSM
```

Ob inicializaciji aktivnosti moramo nastaviti središče zemljevida. To je pomembno zato, da nas aplikacija zna postaviti v osrednji del zemljevida že ob začetku zagona aktivnosti ali pa ob kliku na menijsko možnost *Center Map*. Tudi koordinate središča lahko dobimo prek zemljevida, le da se sedaj postavimo približno v središče marine Izola in odčitamo pripadajočo lokacijo.

V inicializacijskem delu aktivnosti moramo nastaviti tudi začetni nivo povečave. Za zemljevide tipa Mapnik OSM večinoma obstajajo nivoji povečave od 1 pa do 18, kjer večja številka pomeni večji nivo povečave. Ob razvoju aplikacije smo se odločili, da bomo za marino Izola prikazali le dva nivoja povečave z nivojema 17 (najmanjši) in 18 (največji). Z gumboma oblike povečevalnega stekla krmilimo nivo povečave, s tem da se omejimo le med nivoja 17 in 18. Če uporabnik kljub trenutni največji/najnižji povečavi klikne na gumb za povečavo/pomanjšavo, ga aplikacija prijazno opozori, da je največji/najmanjši nivo povečave že dosežen.

Ob inicializaciji moramo poklicati še metodo *InitMarkers*, ki prebere podatke o označevalcih iz podatkovne baze. Za vse označevalce sem spisal še skupni poslušalec na dogodke z imenom *MarkersEventsOverlay1_Click*, ki se pokliče ob kliku na določen označevalec. Ta metoda shrani izbran označevalec in pokliče aktivnost *Pier*, ki je opisana v nadaljevanju.

3.2.4 Aktivnost Pier

Slika 3.8 prikazuje aktivnost *Pier* z največjo funkcionalnostjo in posledično največjo količino programske kode. Po izbiri določenega označevalca marine Izola v aktivnosti Map, tu dobimo izpisane vse leve in desne priveze, ki so v bližini izbranega označevalca. Izpis privezov je dinamičen, tako da je

Slika 3.8: Aktivnost Pier

ob večjem številu označevalcev za določen pomol, seznam izpisanih privezov manjši in obratno. Področje pomola se namreč razdeli na N delov, kjer N predstavlja število označevalcev postavljenih na pomol.

Ob kratkotrajnem kliku na določen privez se pokliče metoda *makeChoicesList* s sledečo logiko:

- Če je privez prost, je obarvan zeleno. Metoda doda možnosti, ki se nanašajo na upravljanje priveza in sicer: prikaz podrobnosti za privez, vpis beležke za privez in vpis stanja števca. Prikazane so na Sliki 3.9. Na praznem privezu je vpisan tudi datum *PROST_DO*, a le če je ta zapisan tudi v podatkovni bazi.

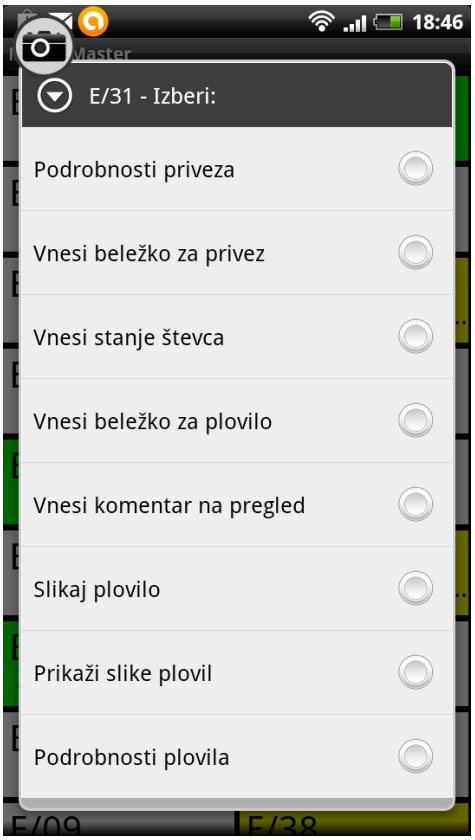
- Če privez ni prost in je na njem samo eno plovilo, je obarvan belo. Metoda poleg možnosti za urejanje priveza prikaže še možnosti za urejanje plovila. Te možnosti so: vpis beležke za plovilo, vpis komentarja na plovilo, fotografiranje in prikaz fotografij plovila ter prikaz podrobnosti plovila. Prikazane so na Sliki 3.10.

- Če privez ni prost in je na njem več plovil, je obarvan rumeno. V tem primeru vsebuje le ime prvega plovila in tri pike. Ob kratkotrajnem kliku na tak privez, se nam ob vsaki možnosti za upravljanje s plovili prikažejo še možnosti za izbiro plovila. To delo opravlja funkcija *MakeChoicesVesselsList*. Podobno velja tudi za števce, kjer funkcija *MakeChoicesMeterList* prikaže vse možne števce na izbranem privezu, ki jih lahko urejamo.

Ob dolgotrajnem kliku na določen privez se pokliče metoda *makeChoicesListLongClick* s sledečo logiko:

- Če je privez prost, metoda doda dve možnosti in sicer *Pregled OK* ter *Plovilo je na tem privezu*. Primer pojavnega okna za ta primer si bralec lahko ogleda na Sliki 3.11. Prva možnost služi označitvi uspešnega pregleda priveza. Po dogovoru to pomeni, da se v tabelo *PRIVEZI-BELEZKE* zapiše beležka z vsebino "Check OK". Druga možnost je

Slika 3.9: Prikazane možnosti ob kratkotrajnem kliku na prost privez

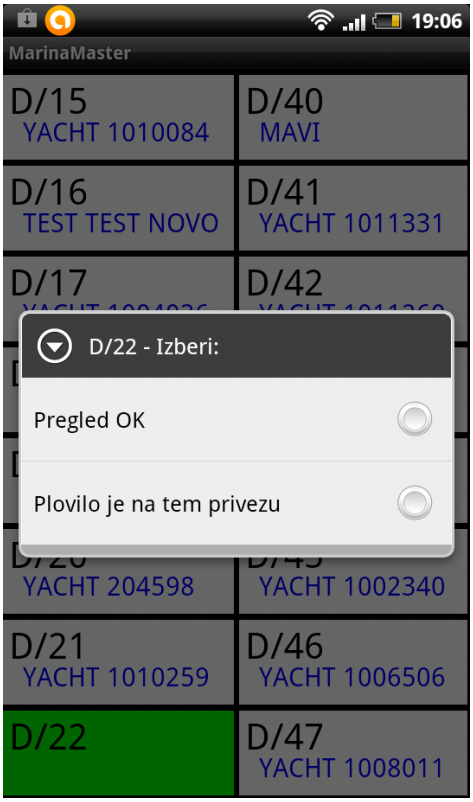


Slika 3.10: Prikazane možnosti ob kratkotrajnem kliku na zaseden privez

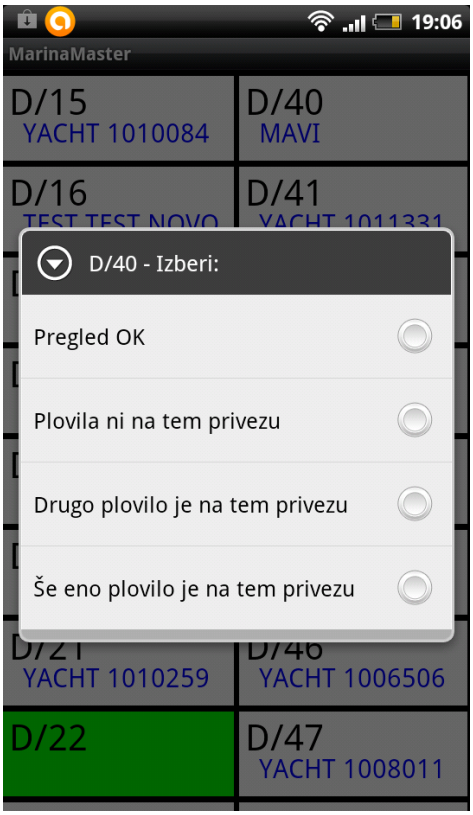
namenjena označitvi prihoda plovila na prazen privez. Ob kliku na to možnost se odpre nova aktivnost *VesselPrespr*, ki bo opisana v nadaljevanju.

- Če je privez zaseden, metoda doda štiri možnosti in sicer *Pregled OK*, *Plovila ni na tem privezu*, *Drugo plovilo je na tem privezu* ter *Še eno plovilo je na tem privezu*. Primer pojavnega okna za ta primer si bralec lahko ogleda na Sliki 3.12. Prva možnost služi označitvi uspešnega pregleda priveza in plovila na njem. Po dogovoru to pomeni, da se poleg vpisa v tabelo *PRIVEZIBELEZKE*, sedaj tudi v tabelo *PREGLEDI* zapiše vrednost "Check OK" kot komentar. Druga možnost

je namenjena označitvi, da plovila ni na privezu, čeprav je v aplikaciji vpisano. V tem primeru se v tabelo *PRESPR* zapiše ustrezen dogodek, privez pa postane prost (obarva se zeleno). Tretja možnost je namenjena označitvi prihoda drugega plovila na privez namesto vpisanega. Četrta možnost pa je namenjena označitvi prihoda še enega plovila poleg že vpisanega. Ob kliku na zadnji dve našteti možnosti se odpre nova aktivnost *VesselPrespr*, ki bo opisana v nadaljevanju.



Slika 3.11: Prikazane možnosti ob dolgotrajnem kliku na prost privez



Slika 3.12: Prikazane možnosti ob dolgotrajnem kliku na zaseden privez

V aktivnosti so dodane še naslednje menijske možnosti, zapisane v Kodi 3.7

```
1 Activity.AddMenuItem(Globals.translator.GetText("
      notes_vessel_menu"),"nmuNotesVessel")
```

```
2 Activity.AddMenuItem(Globals.translator.GetText("
    check_vessel_menu"),"mnuCheckVessel")
3 Activity.AddMenuItem(Globals.translator.GetText("
    notes_mooring_menu"),"mnuNotesMooring")
4 Activity.AddMenuItem(Globals.translator.GetText("
    notes_meter_menu"),"mnuNotesMeter")
```

Koda 3.7: Menijske možnosti v aktivnosti Pier

Ob kliku na določeno menijsko možnost se nam izpišejo vse beležke oz. komentarji plovil, privezov ali števcev, ki smo jih urejali na tem privezu.

3.2.5 Aktivnost MooringDetails

Slika 3.13 prikazuje aktivnost, ki se nam prikaže ob kliku na možnost *Podrobnosti priveza* v prejšnji aktivnosti *Pier*. Vsebuje osnovne podatke o privezu, kot so: oznaka, dolžina, širina, višina, itd.

Aktivnost je sestavljena iz gradnika *ScrollView*, ki je podoben gradniku *ListView*, le da nam ponuja precej več možnosti za prilagoditev izgleda po lastnih željah. Gradnik *ScrollView* nadaljnje vsebuje več gradnikov *Panel*, za vsako vrstico dva. Ti pa naprej zopet služijo kot vsebovalniki (angl. container) za napise oz. gradnike *Label*.

3.2.6 Aktivnost MooringNotes in MooringNotesAll

Aktivnost *MooringNotes* je namenjena vpisovanju beležk na privez in je prikazana na Sliki 3.14. Do nje pridemo, če v aktivnosti *Pier* kliknemo na možnost *Vnesi beležko za privez*. Prek vnosnega polja uporabnik zapiše ustrezno beležko na izbran privez in klikne na gumb *Shrani*. Postopek shranjevanja v podatkovno bazo bo opisan v nadaljevanju.

Slika 3.15 prikazuje aktivnost *MooringNotesAll*, do katere pridemo s pritiskom na menijsko možnost *Beležke privezov* v aktivnosti *Pier*. Vsebuje ta-

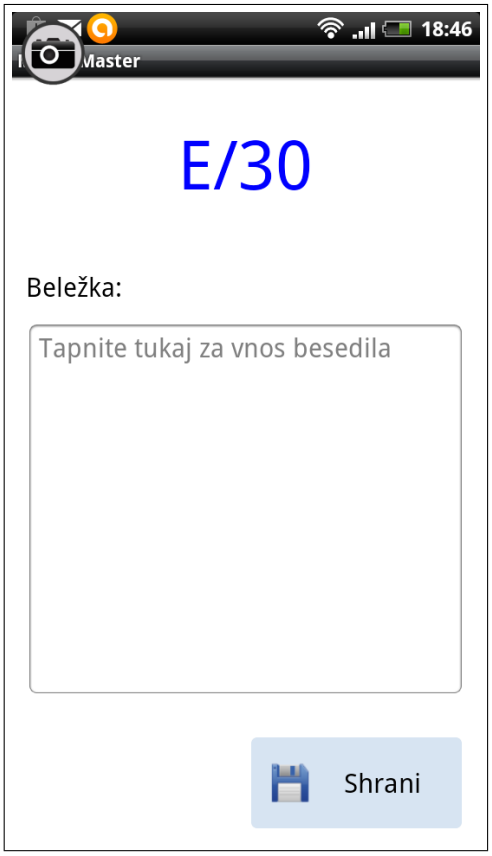
Slika 3.13: Aktivnost MooringDetails

belo, kjer je vsaka vrstica sestavljena iz naslednjih stolpcev: oznaka priveza, datum kreiranja beležke, uporabnik kreiranja beležke in pa sama beležka. Postopek pridobitve podatkov in njihov prikaz bo predstavljen v nadaljevanju.

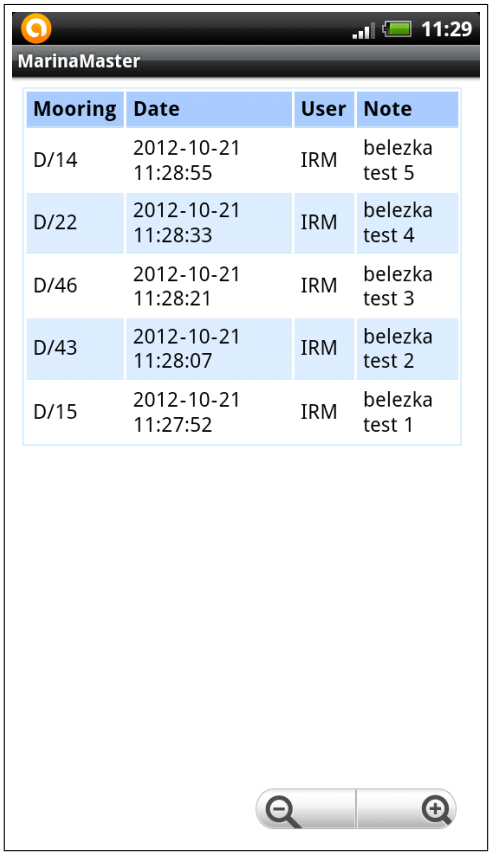
Postopek shranjevanja v lokalno podatkovno bazo

Pred samim shranjevanjem podatkov je potrebno seveda preveriti uporabniški vnos. To delo opravlja funkcija *CheckInput*, ki preveri ali je uporabnik izpolnil vnosno polje. V nasprotnem primeru ga aplikacija prijazno opozori.

Postopek shranjevanja v podatkovno bazo bo opisan samo enkrat, saj koncept velja za vse aktivnosti, pri katerih je potrebno zapisovanje v bazo. V Kodi 3.8 je zapisan del funkcije za shranjevanje podatkov v podatkovno bazo z ustreznimi komentarji.



Slika 3.14: Aktivnost MooringNotes



Slika 3.15: Aktivnost MooringNotesAll

```
1  'shranimo današnji datum v niz
2  DateTime.DateFormat = "yyyy-MM-dd"
3  Dim now As Long
4  now = DateTime.now
5  Dim dateStr As String
6  dateStr = DateTime.date(now)
7
8  'shranimo trenuten čas v niz
9  Dim timeStr As String
10 timeStr = DateTime.Time(now)
11
12 'združimo niza za datum in čas
```

```
13 Dim date As String
14 date = dateStr & " " & timeStr
15
16 'shranimo prijavljenega uporabnika
17 Dim user As String
18 user = Globals.UserObj.GetName
19
20 'shranimo vse vrednosti v tabelo PRIVEZIBELEZKE
21 'vrednosti podamo kot tabelo objektov
22 DBUtils.InsertPriveziBelezke(Array As Object(Null, NnprivezObj.
       GetId_privez, _
23 date, user, edtNote.Text, "Y", "Y"))
24
25 'uporabniku sporočimo, da je bilo zapisovanje uspešno
26 ToastMessageShow(Globals.translator.GetText("saved"), False)
```

Koda 3.8: Funkcija priprave podatkov in njihovega shranjevanja v bazo

Pridobitev podatkov in prikaz

Aktivnost *MooringNotesAll* vsebuje poseben gradnik *WebView*, ki je v osnovi namenjen prikazu spletnih vsebin. Sam sem ga uporabil za prikaz podatkov v obliki tabele s pomočjo jezika HTML in CSS. Koda 3.9 prikazuje postopek pridobitve vsebine HTML celotne tabele *PRIVEZIBELEZKE* glede na izbran privez in polnjenje gradnika *WebView* s pridobljeno vsebino.

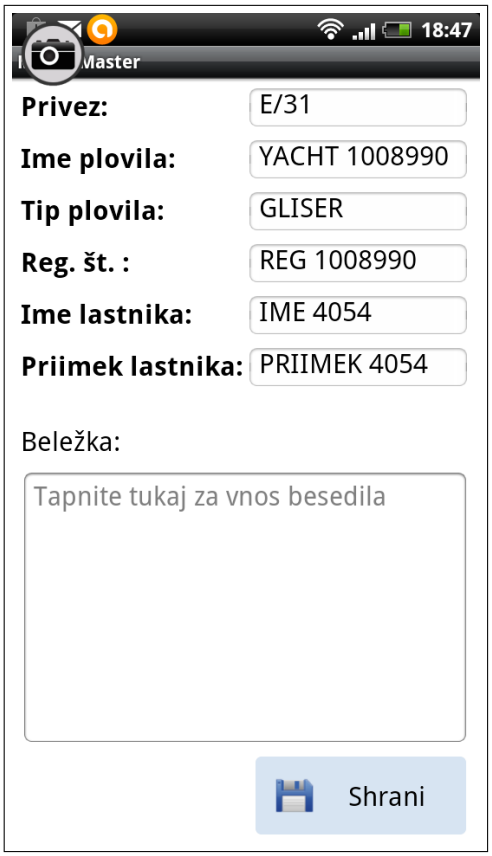
```
1 Dim Content As String
2 'pridobimo vsebino HTML celotne tabele PRIVEZIBELEZKE glede na
      izbran privez
3 Content = DBUtils.GetPriveziBelezkeTableContent(NnprivezObj.
      GetKategorija)
4
5 'če rezultat ni prazen
6 If Content <> 0 Then
7   'napolnimo gradnik WebView z vsebino, ki je zapisana kot HTML
8   webResults.LoadHtml(Content)
9 Else
10  'napolnimo z vsebino "Ni rezultatov"
```

```
11    webResults.LoadHtml(Utils.GetNoResults)
12  End If
```

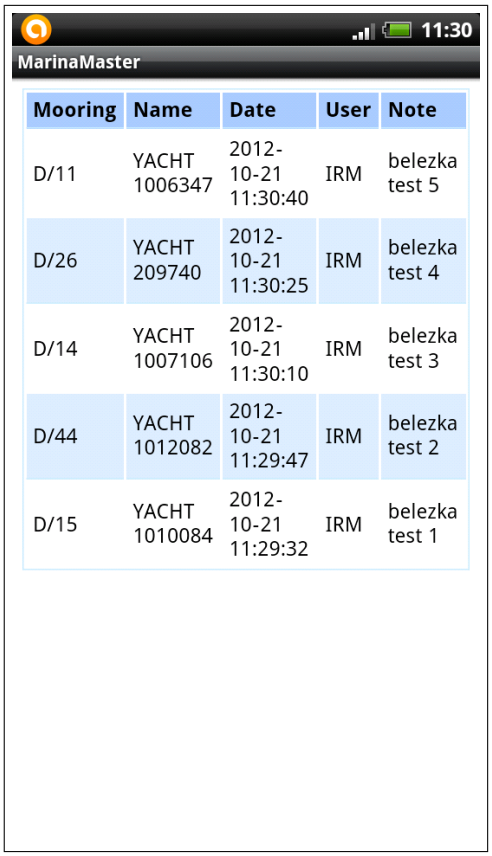
Koda 3.9: Pridobitev in prikaz podatkov izbranega priveza

3.2.7 Aktivnost VesselNotes in VesselNotesAll

Slika 3.16 prikazuje aktivnost *VesselNotes*, do katere pridemo s klikom na možnost *Vnesi beležko na plovilo* v aktivnosti *Pier*. Tukaj lahko uporabnik vpiše beležko za plovilo. Slika 3.17 pa prikazuje aktivnost *VesselNotesAll*, ki se prikaže ob kliku na menijsko možnost *Beležke plovil*. Koncept shranjevanja



Slika 3.16: Aktivnost VesselNotes



Slika 3.17: Aktivnost VesselNotesAll

v podatkovno bazo in prikaz podatkov je v bistvu enak kot pri že opisanih

aktivnostih *MooringNotes* in *MooringNotesAll*, le da tukaj namesto beležke za privez vpisujemo beležko za plovilo.

3.2.8 Aktivnost CheckComments in CheckCommentsAll

Aktivnost *CheckComments* prikazuje Slika 3.18 in je namenjena vpisu komentarjev na preglede plovil in določitvi stanja, v katerem se plovilo nahaja. Poleg vnosnega polja imamo zato pri tej aktivnosti še dodaten spustni meni, prek katerega lahko izberemo stanje plovila.

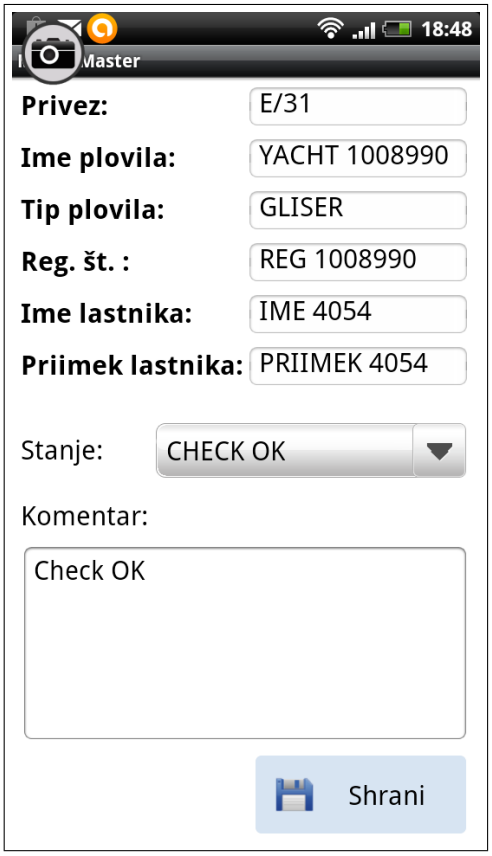
Slika 3.19 prikazuje aktivnost *CheckCommentsAll*, ki se prikaže ob kliku na menijsko možnost *Pregledi plovil* v aktivnosti *Pier*. Koncept shranjevanja v podatkovno bazo in prikaz podatkov je v bistvu enak kot pri že opisanih aktivnostih *MooringNotes* in *MooringNotesAll*, le da tukaj namesto beležke za privez vpisujemo komentar na pregled plovila. Poleg komentarja se v bazo zapiše še šifra stanja plovila.

3.2.9 Aktivnost MeterNotes in MeterNotesAll

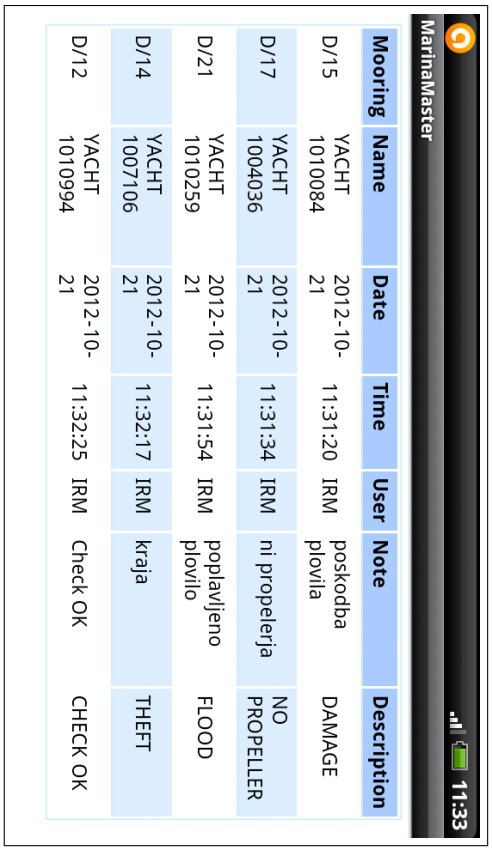
Aktivnost *MeterNotes* prikazuje Slika 3.20, prek katere lahko uporabnik vpiše stanje števca, ki ga je predhodno izbral v aktivnosti *Pier*. Poleg vpisa stanja je možno tudi fotografiranje izbranega priključka s pritiskom na gumb *Dodaj sliko*. Uporabnik lahko vpiše le stanje ali le fotografira priključek, lahko pa stori tudi oboje. To označuje stolpec *ACTION*, ki predstavlja eno od treh možnosti:

1. uporabnik je vnesel stanje in fotografijo
2. uporabnik je vnesel samo stanje
3. uporabnik je naredil samo fotografijo

Koncept shranjevanja je enak že opisanemu, le da se tukaj glede na akcijo, v podatkovno bazo shrani še stanje priključka ali pa fotografija, vključno z



Slika 3.18: Aktivnost CheckComments



Slika 3.19: Aktivnost CheckCommentsAll

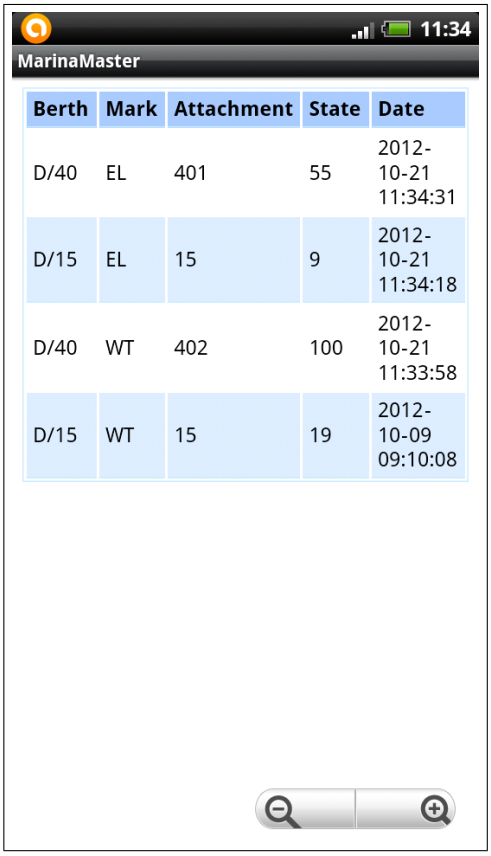
številko, ki označuje akcijo.

Slika 3.21 prikazuje aktivnost *MeterNotesAll*, do katere pridemo ob kliku na menijsko možnost *Stanje števcev* v aktivnosti *Pier*. Prikazuje osnovne podatke o števcih in privezih, za katere smo vpisali števce.

3.2.10 Aktivnost Camera

Ta aktivnost je namenjena izključno samo fotografiranju plovil ali števcev. Da omogočimo delovanje kamere potrebujemo dodatno knjižnjico *ACL*. Aktivnost vsebuje gradnik *Panel* na katerega projeciramo zajeto sliko. Ta

Slika 3.20: Aktivnost MeterNotes



Slika 3.21: Aktivnost MeterNotesAll

je dinamično razširjen čez celoten zaslon. Ko je kamera pripravljena, se pokliče funkcija *Camera1_Ready*. V njej nastavimo lastnosti, kot so orientacija, kvaliteta, itd. Ko zajamemo fotografijo, se pokliče funkcija *Camera1_PictureTaken*, ki je zapisana v Kodi 3.10. Ta funkcija shrani fotografijo v trenutni imenik projekta kot datoteko *Image.jpg*.

```
1 'funkcija, ki se kliče po zajetju fotografije
2 'fotografija je shranjena v tabeli Data
3 Sub Camera1_PictureTaken (Data() As Byte)
4     camera1.StartPreview
5
6   'inicializirmo izhodni tok
7     Dim out As OutputStream
```

```
8       out = File.OpenOutput(File.DirDefaultExternal, "Image.jpg",
            False)
9
10    'zapišemo sliko (tabelo Data) v izhodni tok (fotografija Image
          .jpg)
11      out.WriteBytes(Data, 0, Data.Length)
12      out.Close
13  End Sub
```

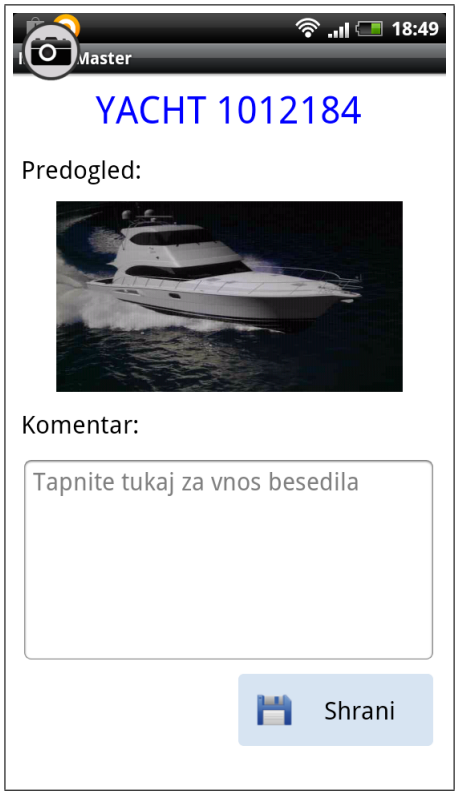
Koda 3.10: Shranjevanje posnete fotografije

Ko neka aktivnost potrebuje shranjeno fotografijo za prikaz (npr. *CameraNotes*), jo najprej skopira v neko drugo datoteko, preimenuje, zapiše v bazo in na koncu izbriše.

3.2.11 Aktivnost CameraNotes

Slika 3.22 prikazuje aktivnost *CameraNotes*, ki je namenjena fotografiranju plovil in vnašanju komentarjev. Koncept shranjevanja je podoben kot pri že opisanih aktivnostih, posebnost je le postopek shranjevanja fotografij v podatkovno bazo. Ta del prikazuje Koda 3.11 z ustreznimi komentarji.

```
1  Dim fileName As String
2  Dim firstPart As String
3
4  'sestavimo novo ime na slikovno datoteko
5  firstPart = PlovilaObj.GetIme.Replace("/", "_")
6  fileName = firstPart & " - " & zapSt
7
8  'prekopiramo Image.jpg v nova.jpg,
9  'kjer je "nova" ime preimenovane datoteke
10 File.Copy(Camera.ImageDir, Camera.ImageFileName, Camera.ImageDir
       , fileName)
11
12 Dim InputStream1 As InputStream
13 InputStream1 = File.OpenInput(Camera.ImageDir, fileName)
14
15 Dim OutputStream1 As OutputStream
```



Slika 3.22: Aktivnost CameraNotes

```
16 OutputStream1.InitializeToBytesArray(1000)
17
18 File.Copy2(InputStream1, OutputStream1)
19
20 'deklariramo tabelo byte-ov za shranitev fotografije
21 Dim Buffer() As Byte
22 Buffer = OutputStream1.ToBytesArray
23
24 'fotografijo zapišemo kot tabelo byte-ov v bazo
25 DBUtils.InsertDatoteke_plovil(Array As Object(PlovilaObj.GetId,
       zapSt, Buffer, _
26 fileName, imgType, 1, edtComment.Text, user, date, user, date,
       user, date, "Y"))
27
28 'zbrišemo fotografijo, saj ta že obstaja v bazi
29 File.Delete(Camera.ImageDir, fileName)
```

Koda 3.11: Shranjevanje posnete fotografije v podatkovno bazo

3.2.12 Aktivnost ImageShower in ImageFullScreen

Slika 3.23 prikazuje aktivnost *ImageShower*, do katere pridemo s klikom na možnost *Prikaži slike plovil* v aktivnosti *Pier*. Namenjena je prikazu fotografij izbranega plovila s pomočjo gradnika *ScrollView*. Pod vsako fotografijo se prikaže še ime plovila in pa njegova zaporedna številka fotografije.

Če uporabnik klikne na eno izmed prikazanih fotografij, se zažene aktivnost *ImageFullScreen*, ki je prikazana na Sliki 3.24. Ta aktivnost je namenjena le prikazu izbrane fotografije čez celoten zaslon. Posebnost aktivnosti *ImageFullScreen* je pretvorba vrednosti iz podatkovne baze v dejansko fotografijo. Ta del prikazuje Koda 3.12.

```
1 'objekt za tabelo DATOTEKE_PLOVIL, vsebuje tudi tip BLOB
2 'v katerem je zapisana fotografija
3 Dim TmpDatoteke_plovilObj As Datoteke_plovil
4 TmpDatoteke_plovilObj.Initialize
```

Slika 3.23: Aktivnost ImageShower

Slika 3.24: Aktivnost ImageFullScreen

```
5  TmpDatoteke_plovilObj = LstImages.Get(i)
6
7  'tabela byte-ov, ki služi kot medpomnilnik (angl. buffer)
8  Dim Buffer() As Byte
9  'pridobimo objekt BLOB
10 Buffer = TmpDatoteke_plovilObj.GetDokument
11
12 'inicializirmo vhodni tok
13 Dim InputStream1 As InputStream
14 InputStream1.InitializeFromBytesArray(Buffer, 0, Buffer.Length)
15
16 'objekt s katerim lahko rišemo grafiko
17 Dim Bitmap1 As Bitmap
```

```
18 'inicializiramo z vhodnim tokom
19 Bitmap1.Initialize2(InputStream1)
20 InputStream1.Close
21
22 'v seznam vseh fotografij dodamo trenutno fotografijo
23 Bitmaps.Add(Bitmap1)
```

Koda 3.12: Nalaganje fotografije iz podatkovne baze v sliko

3.2.13 Aktivnost VesselPrespr

Slika 3.25 in Slika 3.26 prikazuje aktivnost, ki se nam odpre ob izbirah naslednjih možnosti v aktivnosti *Pier*:

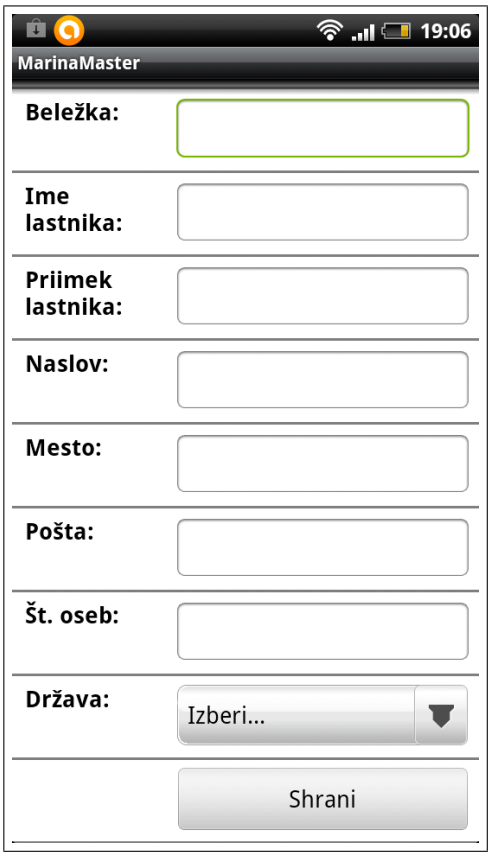
- *Plovilo je na tem privezu*
- *Drugo plovilo je na tem privezu*
- *Še eno plovilo je na tem privezu*

Ob kliku na katerokoli možnost se najprej pojavi še pojavno okno, kjer izberemo vrsto plovila. Izbiramo lahko med pogodbenim plovilom in tranzitnim plovilom. V vseh treh primerih se po kliku na gumb *Shrani* v tabelo *Prespr* zapišejo vrednosti, ki jih je uporabnik vpisal v vnosna polja aktivnosti. V primeru prihoda pogodbenega plovila so podatki v vnosnih poljih že izpolnjeni. Edini obvezni podatki za vnos so: *Ime plovila*, *Dolžina* in *Datum*. Poleg tega lahko uporabnik določi čez koliko privezov se prispelo plovilo razteza. To stori s klikom na izbiro *Več kot 1 privez*, kjer se prikaže seznam vseh sosednjih privezov poleg izbranega. Uporabnik lahko nato izbere poljubno število sosednih privezov (ali nobenega) in potrdi izbiro.

V stolpec *TIP_DOG* tabele *PRESPR* se glede na izbiro vrste plovila zapiše ena izmed vrednosti: "PP" (prihod pogodbenega plovila) ali "PT" (prihod tranzitnega plovila). V primeru odhoda plovila pa se zapiše ena izmed vrednosti: "OP" (odhod pogodbenega plovila) ali "OT" (odhod tranzitnega

plovila). V stolpec *OSN_PRIVEZ* se v primeru izbire več sosednjih privezov vedno zapiše oznaka priveza, na katerega je uporabnik kliknil. V pripadajočih vrsticah stolpca *N_PRIVEZA* pa so po vrsti zapisani vsi sosednji izbrani privezi.

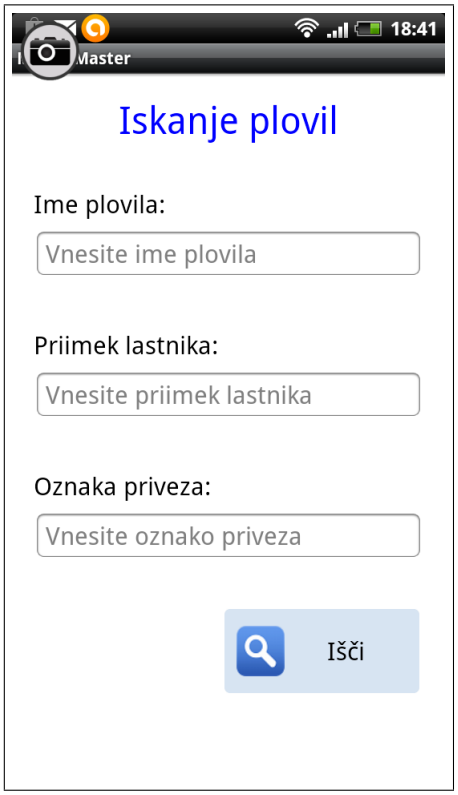
Slika 3.25: Aktivnost VesselPrespr



Slika 3.26: Aktivnost VesselPrespr (nad.)

3.2.14 Aktivnost VesselSearch

Slika 3.27 prikazuje aktivnost, do katere pridemo, če v glavnem meniju (aktivnost *MainMenu*) kliknemo na gumb *Iskanje plovil*. Kot že ime pove, je ta aktivnost namenjena iskanju plovil. Vsebuje tri vnosna polja za iskanje po imenu plovila, priimku lastnika in oznaki priveza. Ko uporabnik pritisne na



Slika 3.27: Aktivnost VesselSearch

gumb *Išči*, se nad podatkovno bazo naredi ustrezna poizvedba in shrani vse rezultate v seznam. Poizvedba je napisana na način, da se ob neprisotnosti določenega kriterija, ta ne upošteva. Rezultate najdemo tudi v primeru, če napišemo samo nekaj začetnih črk. Tako lahko uporabnik npr. za priimek lastnika napiše le niz "brit", poizvedba pa bo našla vse zapise, ki imajo začetni del priimka enak vpisanemu (npr. Britovšek).

Prek poizvedbe naložimo rezultate v seznam, zatem pa pokličemo aktivnost *VesselResults*.

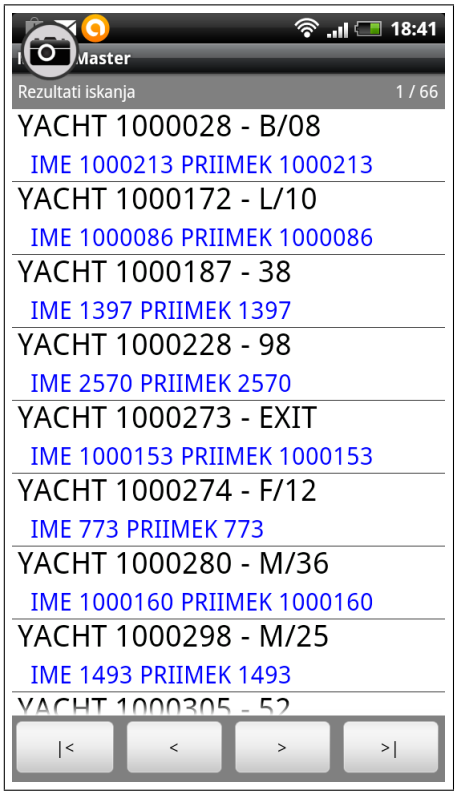
3.2.15 Aktivnost VesselResults

Slika 3.28 prikazuje aktivnost *VesselResults*, ki se zažene ob pritisku na gumb *Išči* v aktivnosti *VesselSearch*. Namenjena je prikazu rezultatov plovil, ki ustrezajo vpisanim kriterijem.

Aktivnost vsebuje gradnik *ListView*, ta pa vsebuje seznam plovil, katerih število določa konstanta *NumOfRecToShow*. Če poizvedba ustvari manj rezultatov, kot jih določa ta konstanta, potem onemogočimo vse gumbe za navigacijo med stranmi. Če pa je rezultatov več, jih ustrezno razdelimo na več strani in zopet omogočimo gumbe za navigacijo. V zgornjem desnem kotu aktivnosti je zapisana številka trenutne strani, zraven pa število vseh strani.

Za navigacijo med stranmi imamo na razpolago štiri različne gumbe. S klikom na prvi ali zadnji gumb se premaknemo na začetno ali končno stran. S klikom na drugi ali tretji gumb pa se premikamo po eno stran nazaj ali naprej.

Ob kliku na ustrezno plovilo v gradniku *ListView* se nam odpre aktivnost *OneVesselResults*.



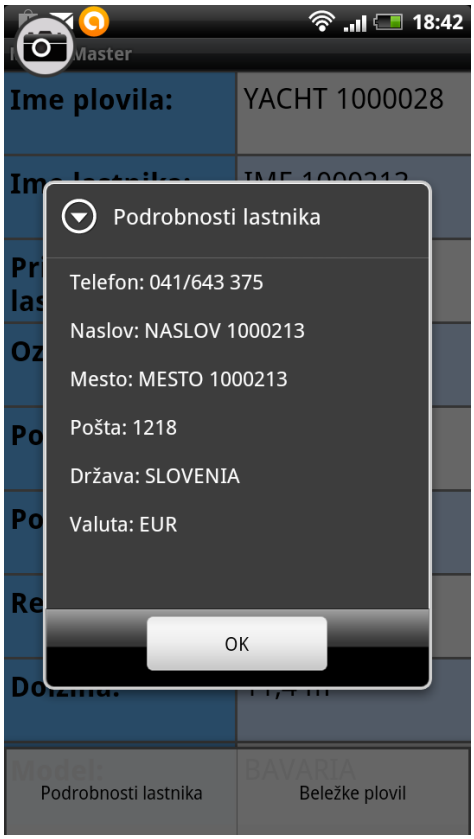
Slika 3.28: Aktivnost VesselResults

3.2.16 Aktivnost OneVesselResults

Slika 3.29 prikazuje aktivnost *OneVesselResults*, ki se zažene ob kliku na določeno plovilo v aktivnosti *VesselResults*. Namenjena je prikazu osnovnih podatkov o izbranem plovilu, njegovem lastniku, privezu, pogodbah, itd. Rezultati so prikazani v gradniku *ScrollView* na enak način, kot pri aktivnostih za prikaz podrobnosti o privezih in plovilih.

V aktivnost sta dodani še dve menijski možnosti za prikaz podrobnosti o la-

Slika 3.29: Aktivnost OneVesselResults



Slika 3.30: Pojavno okno s podrobnostmi lastnika

stniku in za prikaz beležk plovil. Če uporabnik klikne na menijsko možnost za prikaz podrobnosti lastnika, se mu prikaže pojavno okno s podatki o lastniku kot so: telefon, naslov, mesto, pošta, država, itd. Pojavno okno prikazuje

Slika 3.30. Ob kliku na menijsko možnost za prikaz beležk pa se enostavno pokliče že opisana aktivnost *VesselNotesAll*, ki naloži vse beležke izbranega plovila in jih prikaže v gradniku *WebView*.

3.3 Sinhronizacija prek omrežja

Po zaključku dela z grafičnim uporabniškim vmesnikom in logiko na lokalni podatkovni bazi, sem se lotil še sinhronizacije prek omrežja. Ta poteka v obe smeri, torej iz centralne podatkovne baze v lokalno in obratno. V nadaljevanju bom opisal oba pristopa.

Sinhronizacija centralne podatkovne baze z lokalno

Ideja te sinhronizacije je v tem, da posebna skripta PHP na strežniški strani kreira stavke INSERT za vsako tabelo, ki jo potrebujemo v aplikaciji. Te stavke zapiše v posebno tekstovno datoteko *INSERT_ALL_PHP.txt* in jo na koncu arhivira. S tem pridobimo na hitrosti prenosa, saj je datoteka po arhiviranju manjša in potrebuje tudi manj časa za prenos.

Primer omenjene skripte PHP s pripadajočimi komentarji je zapisan v Kodi 3.13. Koda ne vsebuje implementacij klicanih funkcij.

```
1  <?php
2  //povežemo se z lokalno (v tem primeru centralno) podatkovno
       bazo
3  $conn = oci_connect("izola", "izola", '192.168.0.30/ORC1');
4
5  //ime za tekstovno datoteko, kamor bomo zapisovali stavke INSERT
6  $masterFile = "INSERT_ALL_PHP.txt";
7
8  //izbrišemo datoteko, v primeru če že obstaja
9  unlink($masterFile);
10
11 //odpremo tekstovno datoteko
12 $fh = fopen($masterFile, 'a') or die("can't open file");
```

```
13
14 //za vsako želeno tabelo kreiramo stavke INSERT in jih zapišemo
       v datoteko
15 appendPrivezibelezke();
16 appendPregledi();
17 appendPlovilabelezke();
18 appendDatoteke_plovil();
19 appendNnpriklj();
20 appendUsers();
21 appendNndrzave();
22 appendNnmarker();
23 appendNntip();
24 appendNstanje();
25 appendNnprivez();
26 appendPrivezi();
27 appendKupci();
28 appendVessel_v();
29 appendPlovila();
30
31 //arhiviramo datoteko
32 $files_to_zip = array('INSERT_ALL_PHP.txt');
33 $result = create_zip($files_to_zip,'INSERT_ALL_PHP.zip');
34
35 //zapremo datoteko in povezavo do podatkovne baze
36 fclose($fh);
37 oci_close($conn);
38
39 //indikator aplikaciji, da se je postopek izvršil brez napak
40 echo "SUCCESS";
41
42 //implementacija zgornjih funkcij
43 //...
44 ?>
```

Koda 3.13: PHP skripta na strežniški strani

Primer kreiranja stavkov INSERT za tabelo *Privezibelezke* je zapisan v Kodi 3.14. Podobno moramo narediti za vse tabele, ki jih potrebujemo v aplikaciji.

```
1  function appendPriveziBelezke(){
2    global $fh;
3    global $conn;
4
5    //pridobimo vse stolpce iz tabele PRIVEZIBELEZKE
6    $query = "SELECT * FROM PRIVEZIBELEZKE";
7    $stid = oci_parse($conn, $query);
8    oci_execute($stid);
9
10   //začetek vsakega stavka INSERT za to tabelo
11   $insertStart= "INSERT INTO PRIVEZIBELEZKE (ID_PRBELEZKE,
         ID_PRIVEZ, " .
12             "DATUMKREIRANJA, USERKREIRANJA, BELEZKA, VELJAVNA)
                   VALUES (";
13
14   //za vsako vrstico v tabeli kreiramo stavek INSERT iz vseh
         stolpcev
15   while (($row = oci_fetch_array($stid, OCI_BOTH +
         OCI_RETURN_NULLS))) {
16     $dateToParse = $row['DATUMKREIRANJA'];
17     $newDate = parseDate($dateToParse);
18
19     $insertStatement = $insertStart . $row['ID_PRBELEZKE'] . ",
           ";
20     $insertStatement .= $row['ID_PRIVEZ'] . ", '";
21     $insertStatement .=  $newDate . "', '";
22     $insertStatement .= $row['USERKREIRANJA'] . "', '";
23     $insertStatement .= $row['BELEZKA'] . "', '";
24     $insertStatement .= $row['VELJAVNA'] . "');";
25
26     //zapišemo sestavljen stavek INSERT v datoteko
27     fwrite($fh, $insertStatement . "\r\n");
28   }
29 }
```

Koda 3.14: Funkcija za kreiranje stavkov INSERT za tabelo *Privezibelezke*

Na strani aplikacije izvedemo ta tip sinhronizacije s klikom na gumb *Sinhroniziraj* v glavnem meniju (aktivnost *MainMenu*). Ob kliku najprej preverimo, če so bile že narejene kakšne spremembe na lokalni podatkovni bazi. Če sprememb ni bilo, lahko nadaljujemo. V nasprotnem primeru pa vprašamo uporabnika ali želi potrditi zapis sprememb ali ne. Če možnost potrdi, potem se najprej izvede sinhronizacija v obratni smeri. Ta je opisana v nadaljevanju.

V primeru uspešne potrditve za izvedbo sinhronizacije v smeri centralne v lokalno podatkovno bazo, aplikacija najprej izbriše predhodne datoteke. Takoj zatem prebere arhivirano datoteko iz strežnika, jo razširi in v transakciji SQL izvrši vse stavke INSERT. Opisana funkcionalnost je zapisana še v Kodi 3.15.

```
1  Sub FillDatabase(Job As HttpJob)
2    'izbrišemo predhodno arhivirano (.zip) datoteko
3    If File.Exists(File.DirDefaultExternal, "INSERT_ALL_PHP.zip")
         = True Then
4      File.Delete(File.DirDefaultExternal, "INSERT_ALL_PHP.zip")
5    End If
6
7    'izbrišemo predhodno tekstovno datoteko
8    If File.Exists(File.DirDefaultExternal, "INSERT_ALL_PHP.txt")
         = True Then
9      File.Delete(File.DirDefaultExternal, "INSERT_ALL_PHP.txt")
10   End If
11
12   'pridobimo arhivirano datoteko iz strežnika
13   'to delo opravi servisna storitev v ozadju
14   Dim out As OutputStream
15   out = File.OpenOutput(File.DirDefaultExternal, "INSERT_ALL_PHP
         .zip", False)
16
17   'prekopiramo dobljeno datoteko v zunanji pomnilnik
18   File.Copy2(Job.GetInputStream, out)
19   out.Close
20
21   'razširimo datoteko
```

```
22    If Globals.Zip1.ABUnzip(File.DirDefaultExternal & "/
          INSERT_ALL_PHP.zip", File.DirDefaultExternal & "/") Then
23      'če je bilo razširjanje uspešno
24      'napolnimo podatkovno bazo iz dobljene datoteke
25      DBUtils.FillDatabase
26      ProgressDialogHide
27    End If
28  End Sub
```

Koda 3.15: Funkcija za pridobitev datoteke in polnjenje podatkovne baze

Sinhronizacija lokalne podatkovne baze s centralno

Za to smer sinhronizacije delamo obraten postopek od prejšnjega. Uporabnik ga sproži s pritiskom na gumb *Izhod*, kjer aplikacija naprej preveri, če so bile narejene spremembe na lokalni podatkovni bazi. To se zgodi tudi ob kliku na gumb *Sinhroniziraj*, če uporabnik še ni zapisal sprememb. V primeru da so spremembe bile narejene, aplikacija uporabnika vpraša ali želi sedaj zapisati spremembe v centralno podatkovno bazo. Če uporabnik potrdi zapis sprememb, se izvede postopek, ki je opisan v nadaljevanju.

Na strani aplikacije najprej kreiramo datoteke s stavki INSERT ali UPDATE, toda tokrat le za tiste tabele, ki smo jih dejansko spreminjali. Končnih datotek tukaj ni potrebno arhivirati, saj ponavadi vsebujejo zelo malo stavkov in so zato že privzeto dovolj majhne za hiter prenos.

Primer zgornje opisane funkcionalnosti za eno tabelo je zapisan še v Kodi 3.16. Podoben postopek je potrebno narediti za vse tabele, ki smo jih spreminjali.

```
1  Sub GenerateInsertFileAll
2    Dim fileName As String
3    fileName = "INSERT_ALL.txt"
4
5    'izbrišemo vse predhodne tekstovne datoteke
6    DeleteAllFiles
```

```
7
8    'deklariramo objekt zapisovalca, ki bo zapisoval stavke INSERT
9    Dim TextWriter1 As TextWriter
10   'inicializiramo ga na predhodno ustvarjeno ime datoteke
11     TextWriter1.Initialize(File.OpenOutput(File.
           DirDefaultExternal, fileName, True))
12
13   Dim Counter As Int
14   'preštejemo vse vrstice, ki so bile dodane ali spremenjene
15   Counter = Globals.SQL1.ExecQuerySingleResult("SELECT COUNT(*)
       FROM privezibelezke WHERE TO_UPDATE='Y'")
16
17   'če je bila vsaj ena vrstica dodana/spremenjena
18   If Counter > 0 Then
19     'generiramo datoteko INSERT_PRIVEZIBELEZKE z stavki INSERT
20     'za tabelo PRIVEZIBELEZKE
21     GenerateInsertFileForPrivezibelezke
22
23     'iz generirane datoteke vsebino prepišemo v
24     'glavno datoteko INSERT_ALL.txt
25     AppendTo(TextWriter1, "INSERT_PRIVEZIBELEZKE.txt")
26   End If
27
28   'podobno naredimo še za vse ostale tabele
29    '...
30
31   TextWriter1.Close
32 End Sub
```

Koda 3.16: Funkcija kreiranja datoteke s stavki INSERT na aplikacijski strani

Na strežniški strani sem spisal skripto PHP, ki prebere poslano datoteko stavkov INSERT in jih izvede v transakciji. Primer te skripte je zapisan v Kodi 3.17.

```
1 <?php
2   //povežemo se z lokalno podatkovno bazo
3   $conn = oci_connect("izola", "izola", '192.168.0.30/ORC1');
```

```
4
5   //odpremo vhodni tok za datoteko, ki jo pošlje aplikacija
6   $file = file('php://input');
7
8   //za vsako vrstico v datoteki (stavek INSERT)
9   foreach ($file as $line_num => $line) {
10    $stid = oci_parse($conn, $line);
11    //izvedemo stavek INSERT v transakciji
12    oci_execute($stid, OCI_NO_AUTO_COMMIT);
13  }
14
15  //potrdimo transakcijo
16  oci_commit($conn);
17
18  //zapremo povezavo z bazo
19  oci_close($conn);
20
21  //indikator aplikaciji, da se je postopek
22  //uspešno izvedel
23  echo "SUCCESS";
24 ?>
```

Koda 3.17: Strežniška skripta za zapisovanje sprememb v centralno bazo

Poglavje 4

Zaključek

Razvoj aplikacije je trajal dober mesec in pol ter je šel skozi več verzij. Po večkratnem testiranju je končni rezultat diplomske naloge delujoča aplikacija z izpolnjenimi skoraj čisto vsemi zahtevami. Edina zahteva, ki ni bila realizirana, je zapis fotografij v centralno podatkovno bazo.

Pri razvoju seveda ni šlo brez težav. Prva težava se je pojavila pri zaslonih drugačnih velikosti od tiste, na kateri sem testiral aplikacijo. Nekatere aktivnosti se namreč na takih zaslonih ne prikazujejo lepo in je ponavadi kakšna komponenta malenkost skrita (npr. gumb *Shrani*). To bi bilo možno rešiti z boljšim načrtovanjem že opisanih načrtovalskih skript (angl. designer script).

Težave so se pojavile tudi pri različnih verzijah Android-a. Aplikacija je bila namreč testirana tudi na mobilnih napravah z Android-om verzij 2.2 in 2.3.7, kjer pa ni delovala. Razlog, da aplikacija ni delovala na verziji 2.2 je v tem, da pri teh verzijah obstaja hrošč, kjer ne smemo dodajati nestisnjenih datotek večjih od 1 MB v projektni imenik (angl. asset folder) [7]. Čeprav sem večkrat poizkusil rešiti problem s preimenovanjem končnic datotek v *png* ali *mp3*, mi še vedno ni uspelo usposobiti delovanja aplikacije.

Pri verziji Android-a 2.3.7 pa se je pojavila težava drugačne vrste. Apli-

kacija se je namreč ob zagonu sesula, češ da ni mogla najti določene tabele v podatkovni bazi. Izkazalo se je, da je šlo za napako v neujemanju verzij *SQLite*. Sistem smo iz verzije 2.3.7 nato nadgradili na verzijo 4.0.1 in aplikacija je zopet začela delovati.

Razvoj aplikacije se bo seveda še nadaljeval. Zanjo je namreč načrtovana funkcionalnost za prevajanje v kitajski jezik. Načrtovana je tudi funkcionalnost beleženja, kjer se beležijo vse uporabniške akcije in nato shranjujejo v podatkovno bazo. Prav tako bo zagotovo tudi dodana kakšna funkcionalnost v zvezi z urejanjem marine. Načrtuje se tudi povečanje območja pokritosti aplikacije v tem smislu, da bi namesto ene marine pokrivala več marin. Možnosti je seveda še neomejeno in samo čas bo pokazal kaj vse bomo še implementirali v aplikacijo.

Literatura

[1] Delež verzij Android. Dostopno na:
http://developer.android.com/about/dashboards/index.html

[2] Dodatek SQLite Manager za brskalnik Firefox. Dostopno na:
https://addons.mozilla.org/sl/firefox/addon/sqlite-manager

[3] Dokumentacija za Basic4Android. Dostopno na:
http://www.basic4ppc.com/android/documentation.html

[4] Dokumentacija za jezik PHP. Dostopno na:
http://php.net/docs.php

[5] Dokumentacija za podatkovno bazo SQLite. Dostopno na:
http://www.sqlite.org/docs.html

[6] Domača spletna stran Basic4Android. Dostopno na:
http://www.basic4ppc.com

[7] Hrošč v Android-u verzije 2.2. Dostopno na:
http://stackoverflow.com/questions/5176240/
android-inputstream-ioexception-large-file-android-2-2

[8] Odjemalec B4A-Bridge. Dostopno na:
http://goo.gl/QSytM

[9] Operacijski sistem Android. Dostopno na:
http://www.android.com

[10] Podjetje IRM d. o. o. Dostopno na:
http://www.irm.si/si/company

[11] Program Maperative. Dostopno na:
http://maperitive.net

[12] Predstavitev produkta MarinaMaster. Dostopno na:
http://www.marinamaster.si/en/product-information/short-presentation

[13] Razumevanje regularnih izrazov z Notepad++. Dostopno na:
http://blog.creativeitp.com/posts-and-articles/editors/understanding-regex-with-notepad

[14] Sintaksa jezika Basic4Android. Dostopno na:
http://www.basic4ppc.com/android/files/UserGuide.zip

[15] Wikipedia, "Android (operacijski sistem)". Dostopno na:
http://en.wikipedia.org/wiki/Android_(operating_system)

[16] Wikipedia, "SHA-1". Dostopno na:
http://en.wikipedia.org/wiki/SHA-1

Slike

1.1 Tabela privezov z njihovimi osnovnimi podatki 2
1.2 Izpis stanja vseh privezov 3

2.1 Programski tok po rutinah 6
2.2 Grafični izgled orodja SQLite Manager 8
2.3 Razvojno orodje Basic4Android 10
2.4 Nabor možnosti ob kliku na zavihek *Project* 11
2.5 Nabor možnosti ob kliku na zavihek *Tools* 13
2.6 Grafični izgled orodja Designer in Abstract Designer 16
2.7 Logotip operacijskega sistema Android 18
2.8 Grafični izgled vmesnika na Android-u verzije JellyBean . . . 19
2.9 Grafični prikaz deleža verzij Android 21

3.1 Podatkovni model za potrebe mobilne aplikacije 24
3.2 Okno za kreiranje nove tabele 34
3.3 Aktivnost *Main* 39
3.4 Aktivnost *Main* z možnostjo izbire jezika 39
3.5 Aktivnost MainMenu 43
3.6 Pomanjšan zemljevid marine Izola 44
3.7 Povečan zemljevid marine Izola 44
3.8 Aktivnost Pier 47
3.9 Prikazane možnosti ob kratkotrajnem kliku na prost privez . . 49
3.10 Prikazane možnosti ob kratkotrajnem kliku na zaseden privez 49
3.11 Prikazane možnosti ob dolgotrajnem kliku na prost privez . . 50

3.12 Prikazane možnosti ob dolgotrajnem kliku na zaseden privez . 50
3.13 Aktivnost MooringDetails 52
3.14 Aktivnost MooringNotes 53
3.15 Aktivnost MooringNotesAll 53
3.16 Aktivnost VesselNotes 55
3.17 Aktivnost VesselNotesAll 55
3.18 Aktivnost CheckComments 57
3.19 Aktivnost CheckCommentsAll 57
3.20 Aktivnost MeterNotes 58
3.21 Aktivnost MeterNotesAll 58
3.22 Aktivnost CameraNotes 60
3.23 Aktivnost ImageShower 62
3.24 Aktivnost ImageFullScreen 62
3.25 Aktivnost VesselPrespr 64
3.26 Aktivnost VesselPrespr (nad.) 64
3.27 Aktivnost VesselSearch 65
3.28 Aktivnost VesselResults 67
3.29 Aktivnost OneVesselResults 68
3.30 Pojavno okno s podrobnostmi lastnika 68

Tabele

2.1 Delež verzij Android 20

Koda

2.1 Izsek kode iz skripte *Designer* 17
3.1 Lastnosti razreda *Nnmarker* 36
3.2 Metoda SET za nastavitev lastnosti *Longitude* 36
3.3 Metoda GET za nastavitev lastnosti *Longitude* 36
3.4 Kreiranje seznama objektov *Nnmarker* 37
3.5 Pridobitev posameznih objektov in dostop do lastnosti 38
3.6 Preverjanje gesla 41
3.7 Menijske možnosti v aktivnosti Pier 50
3.8 Funkcija priprave podatkov in njihovega shranjevanja v bazo . 53
3.9 Pridobitev in prikaz podatkov izbranega priveza 54
3.10 Shranjevanje posnete fotografije 58
3.11 Shranjevanje posnete fotografije v podatkovno bazo 59
3.12 Nalaganje fotografije iz podatkovne baze v sliko 61
3.13 PHP skripta na strežniški strani 69
3.14 Funkcija za kreiranje stavkov INSERT za tabelo *Privezibelezke* 70
3.15 Funkcija za pridobitev datoteke in polnjenje podatkovne baze 72
3.16 Funkcija kreiranja datoteke s stavki INSERT na aplikacijski strani 73
3.17 Strežniška skripta za zapisovanje sprememb v centralno bazo . 74