UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Benjamin Hüll

Primerjava razvoja mobilnih aplikacij za platformi Android in iOS

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentorica: doc. dr. Mojca Ciglarič

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01744/2011
Datum: 01.04.2011

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

Kandidat: **BENJAMIN HUELL**

Naslov: **PRIMERJAVA RAZVOJA MOBILNIH APLIKACIJ ZA PLATFORMI ANDROID IN IOS**

COMPARING MOBILE APPLICATION DEVELOPMENT FOR ANDROID AND IOS PLATFORMS

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Analizirajte lastnosti najpogosteje uporabljanih mobilnih naprav in platform, na katerih te naprave delujejo. Osredotočite se zlasti na operacijska sistema Android in iOS. Opišite izbrano problemsko domeno in analizirajte problem, ki ga navaja naročnik. Naredite načrt ciljne aplikacije za obe platformi ter izberite primerna razvojna orodja in okolja. Aplikacijo implementirajte v obeh različicah. Nazadnje kritično primerjajte razvoj za obe platformi z vidika razvijalca in rezultate - obe aplikaciji z vidika uporabnika.

Mentor:

M. Ciglarič

doc. dr. Mojca Ciglarič

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani **Benjamin Hüll**,

z vpisno številko **63040052**,

sem avtor diplomskega dela z naslovom:

Primerjava razvoja mobilnih aplikacij za platformi Android in iOS

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom
  doc. dr. Mojce Ciglarič
- 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 3.10.2011 Podpis avtorja:

Zahvala

Zahvaljujem se staršem in sestri, ki so me spodbujali in podpirali med študijem. Posebna zahvala gre mentorici, doc. dr. Mojci Ciglarič za pomoč in svetovanje pri izdelavi diplomske naloge. Zahvalil bi se tudi sodelavcem v podjetju IUS-SOFTWARE, ki so mi pomagali in svetovali pri razvoju aplikacije. Največja zahvala pa gre Sonji, ki me je vedno spodbujala in verjela vame tudi takrat, ko sem bil v dvomih.

Kazalo

Povzetek **1**

Abstract **2**

1 Uvod **3**

2 Opis mobilnih naprav **5**
2.1 Naprave, ki uporabljajo operacijski sistem Android 5
2.2 Naprave, ki uporabljajo operacijski sistem iOS 9

3 Opis platform **13**
3.1 Platforma Android 13
3.2 Platforma iOS 15

4 Opis delovnega okolja **17**
4.1 Delovno okolje za naprave z operacijskim sistemom Android . . 17
4.1.1 Eclipse 17
4.2 Delovno okolje za naprave z operacijskim sistemom iOS 18
4.2.1 Xcode4 18

5 Opis problema **21**

6 Tehnologija **23**
6.1 Arhitektura sistema IUS-TIME 23
6.2 Aplikacijski strežnik 23
6.3 Spletna storitev REST 25
6.3.1 AndroidProvider 27
6.4 Opis podatkovne baze in model podatkovne baze 28
6.4.1 Podatkovna baza Ius-Time 28
6.4.2 Podatkovna baza PortalCMS 29

6.5 Opis aplikacije 33
6.6 Uporabniški vmesnik 35
6.6.1 Razvoj uporabniškega vmesnika za androidno napravo . 35
Aktivnosti 35
Izdelava uporabniškega vmesnika 36
Viri 44
Pravila pri podajanju kvalifikatorjev 46
Naslavljanje virov 49
Menu 50
Pogovorna okna 51
Graditelj gest 54
6.6.2 Razvoj uporabniškega vmesnika za iOS napravo 54
Izdelava uporabniškega vmesnika 56
Pogovorna okna 62
Viri 63
Razred za prepoznavo gest 64

7 Primerjava razvoja za Androida in iOS napravo 67

8 Zaključek 71

Literatura 73

Dodatki 75

A Navodila za uporabo aplikacije - primer za Androida 75

Slike 79

Tabele 81

Seznam uporabljenih kratic in simbolov

A2DP	Advanced Audio Distribution Profile (tehnologija za prenos zvoka preko bluetooth povezave)
ADT	Android Development Tools (Androidova razvojna orodja)
AMOLED	Active-matrix organic light-emitting diode (tehnologija za zaslone)
API	Application Programming Interface (programski vmesnik)
EDGE	Enhanced Data rates for Global Evolution (teh. za prenos podatkov)
GB	Gigabyte (oznaka za kapaciteto pomnilnika)
IDE	Integrated development environment (integrirano razvojno okolje)
iOS	operacijski sistem za Appleove naprave
MB	Megabyte (oznaka za kapaciteto pomnilnika)
Mb/s	Megabit per second (oznaka za prenos podatkov)
MHz	Megahertz (enota za frekvenco ure procesorja)
NFC	Near-Field Communication
OS	Operating System (operacijski sistem)
REST	Representational State Transfer
SDK	Software Development Kit (orodja za razvoj programske opreme
SMS	Short Message Service (storitev kratkih sporočil)
SOAP	Simple Object Access Protocol
UI	User Interface (uporabniški vmesnik)
UMTS	Universal Mobile Telecommunications System (teh. za večji prenos podatkov)

URI	Uniform Resource Identifier
WSDL	Web Services Description Language
XML	Extensible Markup Language (razširljiv označevalni jezik

Povzetek

V diplomski nalogi smo razvili mobilno aplikacijo IUS-INFO za operacijski sistem Android. Zraven smo še razvili spletno storitev, ki deluje po omejitvah REST, in skrbi za dotok podatkov na mobilno aplikacijo. Predstavili smo tudi arhitekturo sistema IusTime in opisali podatkovne baze, do katerih dostopamo s pomočjo spletne storitve. Razvoj uporabniškega vmesnika na Androidu smo primerjali še z razvojem na operacijskem sistemu za tablični računalnik iPad, ki ga poganja operacijski sistem iOS. Pogledali smo si različna delovna okolja, ki nam jih ponujajo proizvajalci posameznih operacijskih sistemov. Primerjali smo orodja za izdelavo gest, orodja za obveščanje uporabnika o spremembah med izvajanjem aplikacije in način dostopa do raznih virov (slike, xml datoteke) . Nato smo še opisali izzive, s katerimi smo se srečali ob razvoju na dveh različnih platformah. Končni cilj diplomske naloge je delujoča mobilna aplikacija za Androidne mobilne telefone.

Ključne besede:

Android, iOS, mobilna naprava, RESTful, aplikacija, uporabniški vmesnik

Abstract

For our thesis, we have developed a mobile application IUS-FOR for the Android operating System. We have also developed a REST web service that enables us to send data to the mobile application, described the IusTime system architecture and the databases which we access with the help of our web service. We have compared the development of applications for Android OS to the development of applications for iOS powered tablet computer iPad. In addition, we have examined different development environments (IDE) that are available from the developers of each operating system. We have compared tools for recognizing gestures, user notification tools and resource access methods (pictures, XML files). Finally, we have described the challenges we have met during the development on two different platforms. The final aim of this thesis is a development of a working mobile application for Android mobile devices.

Key words:

Android, iOS, mobile device, RESTful, application, user interface

Poglavje 1

Uvod

Svet se seli na mobilne naprave. Lahko bi rekli, da se je že preselil, saj ima skoraj vsako podjetje (ki ima spletno stran) bodisi prirejeno spletno stran za mobilne naprave bodisi pa kar narejeno namensko aplikacijo, ki omogoča predstavitev podjetja, ponuja storitve, ki so jih prej ponujali le preko spletnih stranih ali pa ponujajo prirejeno vsebino svojih storitev mobilnim napravam. Vse to jim omogoče ne le širjenje trga, ampak tudi ohranja trenutno bazo uporabnikov, saj slednjim omogoča dostopnost svojih storitev od vsepovsod. Lep primer takega poslovanja so časopisi, ki imajo aplikacije na tabličnih računalnikih. Svojim bralcem omogočajo zastonjski dostop do revij, časopisov na katere so naročeni v fizični obliki, ponujajo naročnino na digitalne izdaje oz. nakup posameznih številk. Te možnosti so zelo priročne, če se odpravimo na počitnice in ne moremo dostopati do dejanskega časopisa, ki nam ga vsako jutro dostavijo na dom.

Na trgu obstaja mnogo različnih mobilnih naprav, od mobitelov do tabličnih računalnikov, na katerih tečejo različni operacijski sistemi. Podjetje Apple proizvaja tako strojno opremo kot programsko opremo za svoje naprave. Med njegove izdelke spadajo iPad, iPhone in iPod na katerih teče operacijski sistem iOS, na drugi strani pa sta operacijska sistema Android in Windows Phone 7. Slednjega je razvil Microsoft, prvega pa Google. Obe firmi sta razvili le operacijski sistem in prodajata licenco proizvajalcem strojne opreme. Seveda imata obadva določena pravila o tem, kako mora biti mobilna naprava zgrajena. Pri tem je Microsoft strožji[18] od Googla, saj le ta dovoljuje več rešitev.

Poglavje 2

Opis mobilnih naprav

2.1 Naprave, ki uporabljajo operacijski sistem Android

Operacijski sistem Android ni omejen le na mobitele, ampak se uporablja tudi v drugih napravah. Med drugim poganja tablične računalnike, med katerimi je bolj znan Samsungov[1] Galaxy Tab. Androidov operacijski sistem najdemo tudi v e-bralnikih oz. e-bralniku, saj trenutno poganja le NOOKcolor, e-bralnik založnika Barnes and Noble's[2]. Naj omenim še internetne televizije, ki jih prav tako poganja Android, čeprav le te ne spadajo ravno med mobilne naprave. Na Švedskem so razvili t.i. skandinavsko androidno televizijo (ang. Scandinavia Android TV[3]), Google[4] pa je s pomočjo drugi podjetij (Intel[5] , Sony[6] , Logitech[7]) razvil tv platformo Google TV. Vidimo lahko, da je število in različnost podprtih naprav ogromno. Katero koli podjetje lahko zakupi licenco in razvije napravo, ki jo poganja Android. Največja prednost je seveda lažji razvoj programske opreme, saj naj bi program, ki smo ga napisali za operacijski sistem Android, delal na vseh napravah, pa naj bo to mobitel, tablični računalnik ali netbook. Odprtost operacijskega sistema je bil glavni cilj Googla, v nasprotju s podjetjem Apple[8], kjer moramo kupiti licenco za razvoj iOS aplikacij, če

[1] http://www.samsung.com/uk/
[2] http://www.barnesandnoble.com/nook/index.asp
[3] http://www.peopleoflava.com/
[4] http://www.google.si/
[5] http://www.intel.com/
[6] http://www.sony.si
[7] http://www.logitech.com/
[8] http://www.apple.com/

želimo testirati programe na napravah. Če te licence ne kupimo, nam ostane preizkus na simulatorju.

Androidove naprave lahko uporabljajo zaslone s poljubno resolucijo, vendar obstajajo določene omejitve []:

- Zaslon mora imeti diagonalo najmanj 6,35 cm (2,5 inč)
- Gostota mora biti najmanj 100 dpi (dots per inch - pikslov na inč)
- Razmerje mora biti med 1.333 (4:3) in 1.779 (16:9)
- Tehnologija za prikaz mora uporabljati kvadratne piksle
- Naprava mora imeti zaslon na dotik

Prav tako mora naprava podpirati spremembo orientacije (položno ali pokončno) zaslona, saj lahko aplikacija zahteva en ali drug način pri izvajanju. Ker pa pri nekoliko večjih napravah to ni možno, saj jih težko zavrtimo (npr. televizija), lahko to rešimo s pomočjo ïletterboxing¨, kjer se uporabi le del zaslona, ko uporabljamo aplikacijo, ki zahteva pokončni način delovanja. Proizvajalec se lahko odloči, da naprava vsebuje kakšno poljubno strojno opremo, vendar mora proizvajalec poskrbeti za API (Application Programming Interface) oz. programski vmsenik na tak način kot ga predpisuje dokumentacija za Android SDK. Uporabniku morajo biti vedno na voljo naslednje funkcije: Home, Menu in Back. Prva nas vrže nazaj na osnovni zaslon, druga nam prikaže menu z raznimi možnostmi, zadnja funkcija pa nas vrže nazaj na prejšnjo aktivnost. Google priporoča, da te funkcije implementiramo s pomočjo namenskih gumbov, dopušča pa možnost vgradnje s pomočjo programov, gibov, zaslona na dotik, dokler so vedno na voljo in ne motijo delovanja aplikacije ali prikaza na zaslonu. Prav tako priporočajo namenski gumb za iskanje. Google torej dopušča kar precej svobode pri oblikovanju uporabniškega vmesnika, na drugi strani pa je Microsoft[9], ki za svoj operacijski sistem Windows Phone 7[10] zahteva točno določeno število gumbov in kako se uporabljajo.

Glavna naprava za katero bom razvijal aplikacijo in na kateri bom tudi testiral, je mobitel Nexus One[11]. Razvil ga je Google kot vodilni model za operacijski sistem Android. Proizvajalo ga je tajvansko podjetje HTC[12]. Prodajati

[9] http://www.microsoft.com/
[10] http://www.microsoft.com/windowsphone/en-ww/default.aspx
[11] https://sites.google.com/a/pressatgoogle.com/nexusone/
[12] http://www.htc.com/europe/

so ga začeli 5. januarja 2010. Medtem ko se je Apple povezal s telekomunikacijskim ponudnikom AT&T[13] in z njihovo pomočjo tržil mobitel iPhone[14], je Google prodajal Nexus One preko svoje spletne strani. Posebnost Nexusa je bila, da telefon ni bil zaklenjen na določenega operaterja. Prodaja preko spleta se ni obnesla, zato so le-to ukinili in začeli s ponudbo v trgovinah. Tehnične značilnosti si lahko ogledamo v tabeli 2.1 [1]. Naslednik mobitela Nexus One se imenuje Nexus S[15]. Google je tokrat združil moči s podjetjem Samsung. Posebnost telefona je vgrajena tehnologija NFC (Near Field Communication), ki je visokofrekvenčna komunikacijska tehnologija kratkega dosega in omogoča izmenjavo podatkov na razdalji do 10 cm [10].

Slika 2.1: Mobitel Nexus One

[13]http://www.att.com/
[14]http://www.apple.com/iphone/
[15]http://www.google.com/nexus/

Velikost	Dimenzije Masa	119 x 59.8 x 11.5 130 g
Zaslon	Tip Velikost	AMOLED zaslon na dotik, 16M barv 480x800, 9cm zaslon. Podpira dotik z več prsti. Merilec pospeška za obračanje slike. Nadzor z dotikom. Navigacija s sledilno kroglico.
Zvok	Tipi opozoril Zvočnik Podpora za slušalke	Vibriranje, MP3 podpora Da. Da.
Spomin	Telefonski imenik Zapis klicev Notranji Zunanji	Skoraj neskončen. Skoraj neskončen. 512 MB RAM, 512 MB ROM Podpira microSD kartice, do 32 GB.
Podatki	GPRS EDGE 3G WLAN Bluetooth USB	Class 10, 32-38 kbps Class 10, 236.8 kbps HSDPA 7.2 Mbps; HSUPA, 2 Mbps Wi-Fi 802.11 a/b/g v.2.1 z A2DP (Advanced Audio Distributioon Profile) microUSB 2.0
Kamera	Glavni podatki Značilnosti Video	5 MP, 2560x1920, avtofokus, LED flash Geotaggin - dodajanje meta podatkov 720x480, 20 slik/sekundo
Značilnosti	CPE Sporočila Brskalnik Radio Igre GPS Java	1GHz procesor Scorpion, GPE, Adreno 200, čipovje Qualcomm QSD8250. SMS(nitni prikaz), MMS, elektronska pošta, takojšnja sporočila. HTML. Privzeto zaklenjen. Da, lahko jih dodatno naložimo z interneta. Da. Java MIDP emulator. Mikrofon za zmanjševanje hrupa v ozadju. Digitalni kompas. Gumb za iskanje. Privzeto naložene aplikacije, Google Search, Maps, Gmail, Youtube, Google Talk.

Baterija	Stanje pripravljenosti Čas pogovorov Predvajanje glasbe	Li-Ion 1400 mAh Do 290 ur (2G) ali do 250 ur (3G). Do 10 ur na 2G ali do 7 ur na 3G omrežju. Do 20 ur.
Gumbi		Gumb HOME - nas vrne na glavno namizje Gumb BACK - nas vrne na prejšnjo aktivnost. Gumb SEARCH - iskanje po internetu Gumb MENU - nam pokaže opcijski menu

Tabela 2.1: Tehnične značilnosti mobitela Nexus One

2.2 Naprave, ki uporabljajo operacijski sistem iOS

Število različnih naprav, ki jih poganja Appleov operacijski sistem iOS, je manjše kot pri Androidu. Če odštejemo vse iteracije in različne ali novejše verzije, dobimo v osnovi le tri oz. štiri različne naprave z operacijskim sistemom iOS. Te naprave so iPhone, iPad, iPod in AppleTV. Vse te naprave proizvaja Apple. Ostalim podjetjim ne prodajajo ali podeljujejo licenc za uporabo iOS. Če se osredotočimo na iPad, opazimo, da ima za interakcijo na voljo le en gumb. Klik na gumb nas vrže iz aplikacije nazaj na namizje, dvo-klik pa nam prikaže program, ki smo ga nazadnje odprli. Pri aplikacijah z več aktivnostmi moramo sami poskrbeti za vračanje na prejšnjo aktivnost, medtem ko imajo Androidne naprave že vgrajen gumb BACK za tako opravilo. iPad je prišel na trg aprila 2010. Izšel je v dveh verzijah, prva je za brežično povezavo uporabljala le Wi-Fi, druga, seveda tudi dražja, pa se je lahko povezala v svet tudi s 3G. Obstajali so tudi različni modeli glede na velikost diska, slednjega pa se na žalost ne da razširiti, ker iPad ne podpira nikakršnih dodatnih spominskih kartic. Na iPadu je že vnaprej nameščenih kar nekaj uporabnih aplikacij, med drugim Safari za brskanje po internetu, Mail za branje elektornske pošto, Photos za gledanje slik, YouTube aplikacija za brskanje po zbirki filmčkov na YouTube strani, iTunes za predvajanje glasbe, iBooks za kupovanje in branje knjig. Za slovenski trg so na žalost na voljo le knjige, ki so v javni rabi in so zastonj. Apple nudi še App Store aplikacijo, s pomočjo katere brskamo in kupujemo vse ostale programe. Za prenos slike, glasbe, filmov in tudi programov, moramo iPad priključiti na računalnik, kjer se nato poveže s programom iTunes. Tega postopka se moramo poslužiti tudi, ko želimo name-

stiti popravke in posodobitve. Neposrednega dostopa do datotečnega sistema nimamo, nasprotno od naprav, ki jih poganja Android. Eno leto po izdaji iPada je izšla nova generacija tabličnega računalnika iPad 2. Naprava ima kar nekaj dodatkov, in sicer dve kameri, sprednjo in zadnjo, hitrejši procesor, več spomina in žiroskop. Prav tako so se nekoliko zmanjšale njegove dimenzije. Oba izdelka sta požela velik kritični uspeh in odlično prodajo. Prav prodaja pa nudi razvijalcem veliko število strank, saj lahko v App Storeu prodajamo svoje aplikacije. Tehnične specifikacije si lahko ogledate v tabeli 2.2 [5].

Slika 2.2: Tablični računalnik iPad. Prva generacija.

Velikost	Dimenzije Masa	242.8 x 189.7 x 13.4 mm 680g (Wi-fi model), 730g (Wi-fi+3G model)
Zaslon	Velikost Resolucija	Diagonala 24,6 cm 1024x8+9. Zaslon odporen na prstne odtise.
Brezžičnost	Model Wi-Fi	Wi-Fi (802.11 a/b/g/n) Bluetooth 2.1. + EDR tehnologija. UMTS (le 3G model) GSM/EDGE (le 3G model)
Spomin	Notranji Shramba Zunanji	256 MB DDR RAM 16, 32 ,64 GB Ne podpira.
	CPE	1 GHz Apple A4

Značilnosti

	Senzorji	Merilec pospeška. Senzor za luč. Magnetomer.
Baterija	Video Audio Stanje pripravljenosti	LiPo baterija 10 ur. 140 ur. 1 mesec.
Gumbi		Gumb Home (za vrnitev na namizje) Gumb za urejanje glasnosti. Gumb za bodisi zaklep zaslona bodisi utišanje zvoka (izberemo v nastavitvah)

Tabela 2.2: Tehnične značilnosti tabličnega računalnika iPad.

Poglavje 3

Opis platform

3.1 Platforma Android

Android je operacijski sistem za mobilne naprave. Narejen je na osnovi operacijskega sistema Linux. Razvilo ga je podjetje z enakim imenom. Leta 2005 je podjetje kupil Google, ki je ravno vstopal na trg mobilnih naprav. Z nakupom je prevzel razvoj in razvojno ekipo.

Namen podjetja Google je bil, da je Android odprtokoden in zastonj, zato je večino izvorne kode izdal pod licenco Apache. V praksi to pomeni, da lahko kdor koli, ki želi uporabljati Android oz. razvijati aplikacije zanj, najde izvorno kodo na internetu. Proizvajalci naprav lahko dodajajo lastne razširitve in priredijo operacijski sistem ter se s tem razlikujejo od ostalih naprav na tržišču. Seveda morajo slediti strojnim zahtevam, ki sem ji že opisal v poglavju o mobilnih napravah.

Razvojni model, ki ga ponuja Android, zaradi svoje preprostosti privlači veliko proizvajalcev. Še posebej zanima podjetja, ki jih je prizadel fenomen Apple-ovega mobitela iPhone. Slednji je povzročil revolucijo na trgu pametnih telefonov, kar je mnoga podjetja prisililo v iskanje novih rešitev, kako naj oživijo svoje izdelke, za katere so do sedaj sami razvijali operacijske sistem. Primer takih podjetji sta Motorola in Sony Ericsson. Slednji in mnogo drugih so rešitev videli ravno v Androidu. Le tega bo podjetje uporabilo za operacijski sistem, strojno opremo pa bodo še naprej razvijali sami.

Prednost pri izbiri Androida je v enotnem pristopu razvoja aplikacij. Razvijalci lahko napišejo program, ki bo tekel na številnih različnih napravah. Proizvajalci mobilnih naprav vidijo ravno v Androidu možnost, da se zoperstavijo pohodu iPhona in pridobijo nazaj delež trga.

Do danes je izšlo že kar nekaj verzij Androida. Kot zanimivost naj omenim,

Verzija	**Kodno ime**
1.1	/
1.5	Cupcake
1.6	Donut
2.0/2.1	Eclair
2.2	Froyo
2.3	Gingerbread
3.0	Honeycomb
3.3	Ice Cream Sandwich

Tabela 3.1: Številke in imena verzij Androida

da se vsaka verzija imenuje po neki sladici. Imena pa si sledijo po abecedi. Tabela 3.1 vsebuje izdaje po številki verzije in njeno kodno ime.

Pri verzijah moramo omeniti, da je izdaja Honeycomb (3.0 - 3.2) na voljo le za tablične računalnike, na pametnih telefonih je možna nadgradnja le do verzije Gingerbread (2.3). To povzroča razdrobljenost trga, kar pa nameravajo popraviti z naslednjo izdajo, ki bo Ice Cream Sandwich. Slednja predstavlja neko smiselno celoto, ki je sestavljena iz prejšnjih izdaj (gingerbread in honeycomb).

Android ima kljub odprtokodnosti in možnosti prirejanja uporabniškega vmesnika vgrajenih veliko uporabnih funkcij:

- Shramba - za shranjevanje podatkov uporablja SQLite, preprostejšo relacijsko bazo
- Povezljivost - podpira GSM/EDGE, brezžični internet, bluetooth, UMTS, CDMA
- Sporočila - podpira SMS in MMS sporočila
- Brskalnik - ima vgrajen internetni brskalnik
- Podpora medijskim vsebinam - video (H.264, MPEG-4, AMR), avdio(MP3, AAC, MIDI), slike (PNG, JPEG, GIF, BMP)
- Podpora strojni opremi - senzor za pospešek, kamera, digitalni kompas, GPS, senzor za zaznavanje bližine
- Podpora multi touch zaslonom
- Podpira multi-tasking aplikacij

- Podpira predvajanje in prikazovanje vsebin spisanih v Flash.
- Podpora tethering - telefon spremeni v brezžično točko in tako deli svojo internetno povezavo z drugimi napravami

Androidov operacijski sistem je na grobo razdeljen v pet delov, ki potekajo preko štirih plasti:

- Linuxovo jedro - jedro, ki je osnova za Androida. Ta plast vsebuje gonilnike za vse vgrajene komponente v napravi.
- Knjižnice - tukaj je shranjena koda, ki zagotavlja glavne funkcije v operacijskem sistemu Android. Npr. SQLite knjižnica vsebuje podporo podatkovnim bazam, knjižnjica WebKit zagotavlja funkcionalnosti za brskanje po internetu.
- Izvajalno okolje Android - množica knjižnjic, ki omogoča razvijalcem pisanje aplikacij s pomočjo programskega jezika Java. Prav tako vključuje navidezni stroj Dalvik, ki je bil razvit specifično za Android in optimiziran za mobilne naprave. Aplikaciji omogoči, da se zagnja kot samostojen proces z lastno instanco Dalvik stroja.
- Aplikacijsko ogrodje - Izpostavi razne zmožnosti operacijskega sistema Android, da jih lahko razvijalci uporabijo v svojih aplikacijah.
- Aplikacije - V to najvišjo plast spadajo aplikacijo, ki so na voljo ob nakupu naprave in aplikacije, ki jih naložite z Androidove spletne trgovine in namestite na mobilno napravo. Aplikacija, ki sem jo napisal v okviru diplomske naloge, se nahaja na tej plasti.

3.2 Platforma iOS

iOs je Apple-ov operacijski sistem za mobilne naprave. Do junija se je imenoval iPhone OS, ker ga je do takrat uporabljal le njihov lastni telefon iPhone. Danes ga uporabljajo štirje izdelki, in sicer iPhone, iPad, iPod touch in Apple TV. Druga podjetja ga ne uporabljajo, ker Apple ne prodaja licenc za uporabo.

iOS so prvič pokazali leta 2007 na konferenci MacWorld (Cohen, 2007), kjer so ga predstavili le kot OS X. Takrat še ni podpiral aplikacij tretjih oseb, Steve Jobs je to argumentiral, češ da lahko razvijalci naredijo spletne aplikacije, ki bi se obnašale enako kot avtohtone aplikacije. Nekaj mesecev kasneje so le napovedali, da bodo razvijalska orodja kmalu na voljo. Marca 2008 so izdali

Verzija	**Najnovejša naprava, ki jo podpira**
3.1.3	iPhone in iPod Touch (1.generacija)
4.2.1	iPhone 3G in iPod Touch (2. generacija)
4.2.10	iPhone 3GS, iPhone 4, iPod Touch (3. In 4. generacija), iPad 1 in iPad 2
4.3.5	Eclair
5.0 (Beta)	iPhone 3GS, iPhone 4, iPod Touch (3. In 4. generacija), iPad 1 in iPad 2

Tabela 3.2: Verzije operacijskega sistema iOS

SDK in preimenovali operacijski sistem v iPhone OS (iOS, 2011). Naslednje preimenovanje v iOS je bilo leta 2010, potem ko so izdali še tablični računalnik iPad. Tabela 2 vsebuje verzije iOS in naprave, katere podpira.

V tabeli lahko vidimo, da je v delu že verzija 5.0, ki bo prinesla mnogo novosti, med drugim storitev iCloud (shranjevanje podatkov v oblaku), iMessage (storitev, ki omogoča pogovarjanje preko naprav iOS), storitev Twitter je integrirana v iOS. Medtem ko so nadgradnje pri Androidu do sedaj bile vse brezplačne, so bile v preteklosti Appleove nadgradnje plačljive.

iOS lahko razdelimo na štiri dele oziroma plasti:

- Cocoa Touch - je programski vmesnik, s katerim gradimo aplikacije za iOS naprave
- Medijske/Aplikacijske storitve
- Osnovne storitve
- Mac OS X jedro

Cocoa Touch ponuja ogrodja za razvoj aplikacij za mobilne naprave. Osredotoča se na dotikabilne grafične vmesnike. Tako UIKit ponuja orodja za razvoj grafičnih, dogodkovno vodenih programov. UIKit nam omogoča dostop do kontrol za grafični vmesnik, kot so gumbi in različni pogledi. Prav tako lahko nadziramo aplikacijo z merilnikom pospeška in različnimi gestami, ki jih izvajamo na zaslonu na dotik.

Poleg ogrodja za uporabniški vmesnik nam Cocoa Touch ponuja ogrodja za upravljanje z videom in zvokom, grafiko in animacijo, ogrodje za uporabniške aplikacije kot so Maps, App Store, Imenik (Address Book, Map Kit, Store Kit). Poznamo še ogrodji za upravljanje s podatki in za omreženje.

Pri diplomski nalogi se bom osredotočil na ogrodje UIKit za uporabniški vmesnik.

Poglavje 4

Opis delovnega okolja

4.1 Delovno okolje za naprave z operacijskim sistemom Android

4.1.1 Eclipse

Za razvoj aplikacije za Androida sem uporabil razvijalno okolje Eclipse (verzijo Helios), ki ga predlaga tudi Google. Eclipse podpira razvoj aplikacij v več programskih jezih, npr. Java, C, C++, Python itd.

Na voljo so verzije za različne operacijske sisteme (Windows, MacOS, Linux) kot tudi različne verzije za 32-bitno in 64-bitno arhitekturo. Medtem ko so verzije Androida poimenovane po raznih sladicah, so verzije Eclipsa poimenovane v povezavi z astronomijo. Tako so prve tri verzije imele imena po Jupitrovih lunah, naslednje pa po znanstveniku Galileju, grškemu sončnemu bogu Heliosu, najnovejš,a že izdana verzija se imenuje Indigo po, verzijo, ki je napovedana za konec junija 2012, pa so izglasovali uporabniki sami.

Po namestitvi programa Eclipse moramo namestiti tudi komplet za razvoj programske opreme za Androida (ang. Android SDK). Ta vsebuje razhroščevalnik, knjižnjice, emulator, dokumentacijo, primere in navodila.

Za razvoj v razvojnem okolju Eclipse moramo še namesiti še razširitev ADT (Android Development Tools). Z njim lahko ustvarjamo in iščemo napake v Androidovih aplikacijah. Prav tako nam omogoči, da:

- Ustvarimo nove projekte za Android aplikacije
- Dostopamo do emulatorjev in naprav
- Prevajamo in razhroščujemo aplikacije

- Izvažamo aplikacije v t.i. Android Packages (apk), katere lahko nato zaženemo na mobilni napravi. Tako namestimo aplikacijo in jo lahko začnemo uporabljati oz. testirati.
- Ustvarimo lahko digitalne certifikate za podpisovanje naših aplikacij.
- Gradimo uporabniške vmesnike na dva načina. Lahko pišemo XML datoteko ali pa z miško postavljamo posamezne komponente na prikazan zaslon.

4.2 Delovno okolje za naprave z operacijskim sistemom iOS

4.2.1 Xcode4

Xcode 4 je program, ki vsebuje vsa potrebna orodja za razvoj aplikacij za MacOS in iOS. Z verzijo 4 smo dobili tudi vgrajen Interface Builder (urejevalnik za uporabniški vmesnik), ki je bil pred tem samostojen program. Program je že privzeto narejen za razvoj aplikacij za iOS, tako da nam ni potrebno namestiti nobenih dodatnih vtičnikov, kot je bilo to potrebno pri Eclipsu. Program je na voljo brezplačno, le tistim uporabnikom, ki so plačali letno članarino (99 dolarjev) za iOS razvojni program. Članstvo v tej skupini nam omogoča testiranje narejenih aplikacij na mobilni napravi (iPad, iPhone in iPod) in objavljanje aplikacij v spletni trgovini, ki je dostopna preko aplikacije AppStore. Če nismo včlanjeni v program, nam je na voljo verzija Xcode 3, ki je brezplačen za vse ali pa nam je na voljo četrta verzija za štiri evre. Slednjo možnost smo izbrali mi, saj želimo le primerjati razvoj uporabniškega vmesnika in še nismo pripravljeni na razvoj in izdajo aplikacije na Applovih mobilnih napravah. Program nam omogoča, da:

- Ustvarimo nov projekt, kjer že na začetku določimo, da delamo na iPad aplikaciji
- Uporabimo že predpripravljene razporeditve
- Uporabimo vgrajene emulatorje za vse mobilne naprave
- Prevajamo in razhroščujemo aplikacije
- Potegnemo povezave od kontrol uporabniškega vmesnika do izvorne kode. Tako z miškinim klikom ustvarimo potrebne metode.

- Primerjamo različne verzije programske kode

Poglavje 5

Opis problema

Cilj diplomske naloge je razviti prototipno mobilno aplikacijo za podjetje IUS-SOFTWARE[1]. Podjetje se ukvarja z naslednjimi dejavnostmi:

- oblikovanje, razvoj, upravljanje, vzdrževanje in distribucijo podatkovnih baz pravne in poslovne narave
- razvoj informacijsko-tehnoloških in programskih rešitev za povezovalne sisteme, ki omogočajo celovito upravljanje z vsebinami nestrukturiranih dokumentov (besedila, upravljanje z vsebinami, upravljanje z dokumenti)
- zastopstvo tujih družb za posredovanje sorodnih zbirk podatkov in izdelavo programske opreme

Glavni izdelki podjetja so naslednji spletni portali:

- Portal IUS-INFO, pravni in poslovni informacijski portal, je komercialna baza podatkov na spletu, ki vsebuje največji izbor pravnih informacij iz Slovenije in Evropske unije na enem mestu.
- Portal FinD-INFO[2], finance in davki; je najsodobnejši portal in temelji na tehnologiji IUS-INFO. Portal FinD-INFO ponuja celovit pregled finančnih, računovodskih, davčnih in pravnih informacij ter omogoča enostavno iskanje podlag za rešitev strokovnih vprašanj.
- Hrvaški Portal IUS-INFO[3], pravni in poslovni informacijski portal, je komercialna baza podatkov na spletu, ki vsebuje največji izbor hrvaških pravnih informacij in pravnih informacij Evropske unije na enem mestu.

[1] http://www.iusinfo.si/
[2] http://www.findinfo.si/
[3] http://www.iusinfo.com.hr/

V uvodu sem že omenil, da se vse seli na mobilne naprave. Podjetja morajo ponujati svoje storitve "na poti". Za diplomsko nalogo sem se lotil izdelave mobilne aplikacije podjetja IUS-SOFTWARE. Podjetje se ukvarja s pravno informacijskimi storitvami, ponuja čistopise zakonov, novice in kolumne, ki so pomembne za stranke, katere so pretežno pravniki. Naše storitve trenutno ponujamo preko dveh spletnih portalov. V okviru diplomske naloge bom razvil aplikacijo, ki bo dostopna na Androidovih mobilnih telefonih. Aplikacija bo omogočala branje novic in kolumn na mobilni napravi, uporabo raznih izračunov (npr. izračun potnih stroškov), ter brskanje po zakonih.

Posebno pozornost bomo morali nameniti uporabniškemu vmesniku, saj rezultatov ne bo prikazoval velik monitor, ampak devet centimetrski zaslon na mobitelu. Kako naj prikažem na majhnem zaslonu zakon, ki obsega tisoč členov in zasede 70 strani? Poleg aplikacije za mobitel z operacijskim sistemom Android bom naredil še primerjavjo z orodji za razvoj aplikacije na iPadu, tabličnem računalniku, ki ga poganja iOS. Ne samo, da se bomo ukvarjali z drugim programskim jezikom, delovnim okoljem, velika sprememba bo v velikosti zaslona. iPad ima večji zaslon kot mobitel. Lahko bi sicer naredil samo povečano verzijo aplikacije, vendar to ne bi bila pametna izraba "prostora", ki ga imamo na voljo.

Pri razvoju bom uporabil različna orodja. Aplikacijo za Androida bom napisal v programskem jeziku Java[4] v razvojnem okolju Eclipse[5]. Prototip za iPada bom naredil na računalniku MacAir v okolju Xcode 4[6] in programskem jeziku objective-C. Za dostop do podatkov, ki jih bo aplikacija prikazovala, bom napisal spletno storitev REST v okolju .NET s pomočjo Visual Studia 2010[7] in programskega jezika C#, ki ga za razvoj uporabljamo tudi v podjetju Ius Software. S spletno storitvijo bom dostopal do relacijskih podatkovnih baz Microsoft SQL Server[8].

[4]http://www.java.com/en/
[5]http://www.eclipse.org/
[6]http://developer.apple.com/technologies/tools/whats-new.html
[7]http://www.microsoft.com/visualstudio/en-us
[8]http://www.microsoft.com/sqlserver/en/us/default.aspx

Poglavje 6

Tehnologija

6.1 Arhitektura sistema IUS-TIME

Arhitekturo sistema IUS-TIME razdelimo na tri sloje:

- Podatkovni sloj
- Poslovni sloj
- Predstavitveni sloj

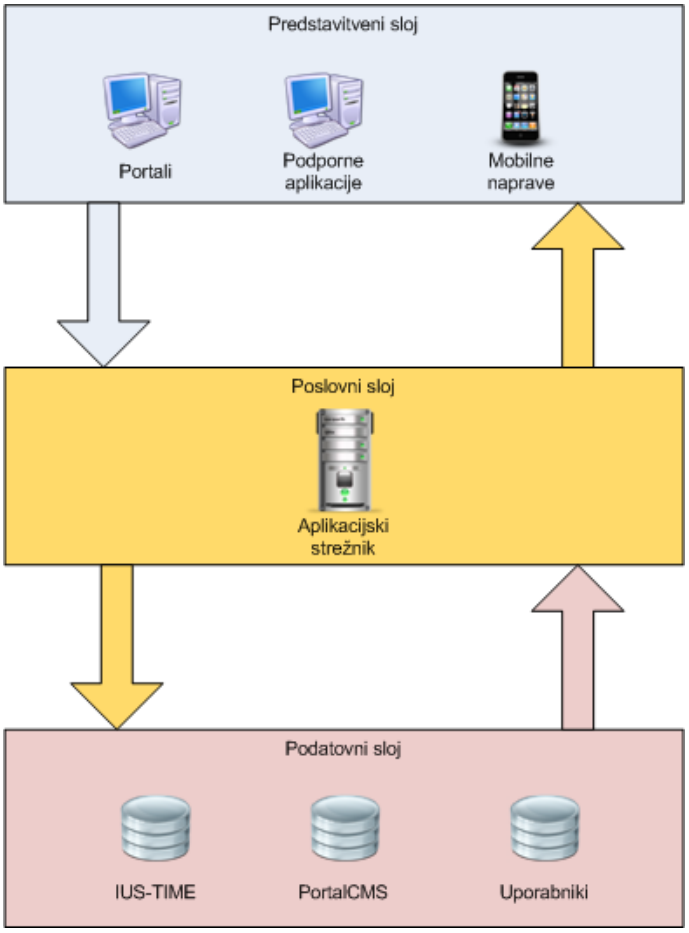
Podatkovni sloj predstavljajo podatkovne baze, kamor shranjujemo raznovrstne dokumente, od zakonov, člankov do novic in kolumn in seveda tudi šifrante.

Do podatkovnega sloja dostopa **poslovni sloj**, kjer imamo aplikacijski strežnik, ki služi kot vmesnik med podatki in aplikacijami oz. klienti, ki te podatke uporabljajo. Poslovni sloj vsebuje funkcije, ki nam dostop do podatkov olajšuje ter skrbi, da so vsi dostopi standardizirani. Omogoča tako branje kot pisanje podatkov. Naša mobilna aplikacija IUS-INFO trenutno potrebuje samo funkcijo branja. Poslovni sloj mora poskrbeti, da dobimo željeni podatek.

Na **predstavitvem sloju** se nahajajo naši portali (IUS-INFO, FinD-INFO), podporne aplikacije, ki jih uporabljamo znotraj podjetja, pa tudi naša nova mobilna aplikacija. Nivo predstavlja vse kliente, ki preko poslovnega sloja dostopajo do podatkov na podatkovnem sloju.

6.2 Aplikacijski strežnik

Aplikacijski strežnik spada med vmesno programsko opremo oz. middleware. Predstavlja vez med podatki in aplikacijami-klienti. Vsebuje veliko različnih



Slika 6.1: Arhitektura IUS-TIME

Vir	GET	PUT	POST	DELETE
Seznam: http://.../News	Vrne seznam vseh elementov	Zamenjamo celoten seznam	Dodamo nov vpis v seznam	Zbrišemo celoten seznam
Določen element: http://.../News/21	Vrne novico, ki ima id. številko 21	Novico zamenjamo ali ustvarimo novo	/	Izbrišemo novico iz seznama

Tabela 6.1: HTTP metode za spletno storitev RESTful

knjižnic, ki ta dostop še dodatno olajšajo. Tako nam je na voljo knjižnjica, ki vsebuje vse entitete (*BusinessEntities*), knjižnico za branje podatkov za spletno stran (*PortalProvider*), *DocumentProvider* skrbi za delo z dokumenti itn. Za oddaljeni dostop do strežnika poskrbita knjižnici *ServiceProviders* in *DataAccessComponents*. Najpomembnejši novi ponudnik za nas je **Android-Provider**, ki skrbi za branje podatkov, katere želimo prikazati uporabniku na mobilni napravi. Trenutno ime sicer nakazuje, da lahko ponudnika uporabimo le za androidne naprave, vendar to ni res. Uporabimo ga lahko za uporabo na vseh prenosnih telefonih, tudi če uporabljajo operacijski sistem Windows Phone 7, iOS ali WebOS. Prav tako uporabljamo ponudnika **Calculations**, ki nam vrne rezultat izračuna glede na podane podatke.

6.3 Spletna storitev REST

Ponudnik je narejen kot RESTful spletna storitev. Za dostop do strežnika uporabljamo naslednje elemente:

- URI naslov za dostop do storitve (http://www.iusinfo.si)
- Množico operacij, katere realiziramo s pomočjo HTTP metod (GET, POST, DELETE,PUT)
- Tipe podatkov, ki jih spletna storitev podpira (JSON, XML)

Tabela 6.1 prikazuje, kaj naredijo HTTP metode. Pri razvoju aplikacije sem uporabljal le metodo GET za branje podatkov.

REST pomeni REpresentational State Transfer in predstavlja množico omejitev, ki ob uporabi ustvarijo arhitekturni stil. Omejitve, ki vodijo v RESTful sistem zahtevajo, da je sistem:

- Osnovan po načinu klient-strežnik
- Brez stanja - ni potrebe po vzdrževanju seje. Iz tega sledi, da je vsak klic strežnika neodvisen od prejšnjih in prihodnjih klicov.
- Shranjuje v predpomnilniku.
- Z enotnim dostopom - vsak vir mora imeti edinstven naslov in veljavno točko dostopa.
- Skalabilen.
- Zmožen vrniti kodo na zahtevo

Prednosti uporabe REST spletnih storitev so naslednje:

- Storitev prenaša manj podatkov, npr. xml sporočila nimajo dodatne navlake (nimamo dodatnih oznak kot sta Envelope in
- Body pri SOAP spletnih storitvah.
- Berljivi rezultati za ljudi
- Preproste za narediti, ne potrebujemo nobenih orodjarn (ang. toolkitov)
- Preprostost dostopa - dostopa lahko s katerega koli brskalnika
- Varnostna pravila so zlahka določljiva. Za klic GET metode vemo, da je vedno varen, ker že po definiciji ne morem spreminjati podatkov.

Poglejmo si en preprost primer HelloWorld spletne storitve SOAP in nato še primer za RESTful spletno storitev. Pri prvi izgledata zahteva in odgovor takole:

Zahteva:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="http://duke.org/hello">
 <soapenv:Body>
 <ns1:hello>
 <name>Luka</name>
 </ns1:hello>
 </soapenv:Body>
</soapenv:Envelope>
```

Odgovor:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="http://duke.org/hello">
 <soapenv:Body>
 <ns1:helloResponse>
 <return>Živijo, Luka !</return>
 </ns1:helloResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

Poleg tega moramo za klic ustvariti SOAP klienta na podlagi WSDL opisa. Ustvariti moramo potrebne razrede, nastaviti možnosti (atribute) in napisati programsko kodo za generirane objekte. Na drugi strani imamo storitev REST, kjer moramo poznati samo naslov spletne storitve (http://www.iusinfo.si) in naslov vira, ki ga želimo imeti. Polni klic zgornje metode bi izgledal takole:

http://www.iusinfo.si/.../AndroidProviderSI.svc/REST/name/Luka

V odziv pa dobimo naslednji odgovor (v obliki xml datoteke):

```
<ns:helloResponse>Živijo, Luka</ns:helloResponse>
```

Glede na prednosti REST spletnih storitev smo se pri razvoju mobilne aplikacije odločili za uporabo REST spletne storitve. Glavna prednost je seveda manjša količina podatkov pri prenosu, kajti mobilna omrežja pri tej postavki niso poceni.

6.3.1 AndroidProvider

V našem ponudniku imamo definirane metode za pridobivanje novic, kolumn, zakonov in izračunov. Spodaj imamo primer metode za izračun avtorskega honorarja in definicijo vmesnika, kjer določimo metodo (GET), URI naslov ter obliko, v kateri naj vrne rezultate (v našem primeru bo to xml datoteka) in katera metoda se uporabi:

Metoda:

```
public Dictionary<string, string> IzracunAvtHon(string BrutoZnesek,
string Procent)
{
        // Deklaracija razreda //
        AvtorskiHonorar MavtHon = new AvtorskiHonorar();
```

```
        //Vhodni parametri //
        MavtHon.Bruto = BrutoZnesek;      // Primer : "1000"
        MavtHon.Procent = Procent;        // Primer : "1000"
        // Izracun vrne Dictionary v obliki Key = Value
        return MavtHon.Izracunaj();
}
```

Vmesnik:

```
[OperationContract]
[WebInvoke
   (Method = "GET",
    ResponseFormat = WebMessageFormat.Xml,
    BodyStyle = WebMessageBodyStyle.Bare,
    UriTemplate = "IzracunAvtHon?BrutoZnesek={BrutoZnesek}&
    Procent={Procent}")]
[FaultContract(typeof(BE.FaultContracts.BusinessLogicFault))]
List<BE.Calculations> IzracunAvtHon(string BrutoZnesek,
string Procent);
```

6.4 Opis podatkovne baze in model podatkovne baze

V aplikaciji prikazujem novice in kolumne, ki pripadajo podatkovni bazi PortalCMS (slika 6.3), prav tako prikazujem zakone oz. člene zakonov, ki se nahajajo v bazi Ius-Time (slika 6.2). Obe bazi sta ogromni, vendar ne uporabljamo vseh tabel, zato bom prikazal le tiste dele, ki jih, in so bistveni za našo mobilno aplikacijo.

6.4.1 Podatkovna baza Ius-Time

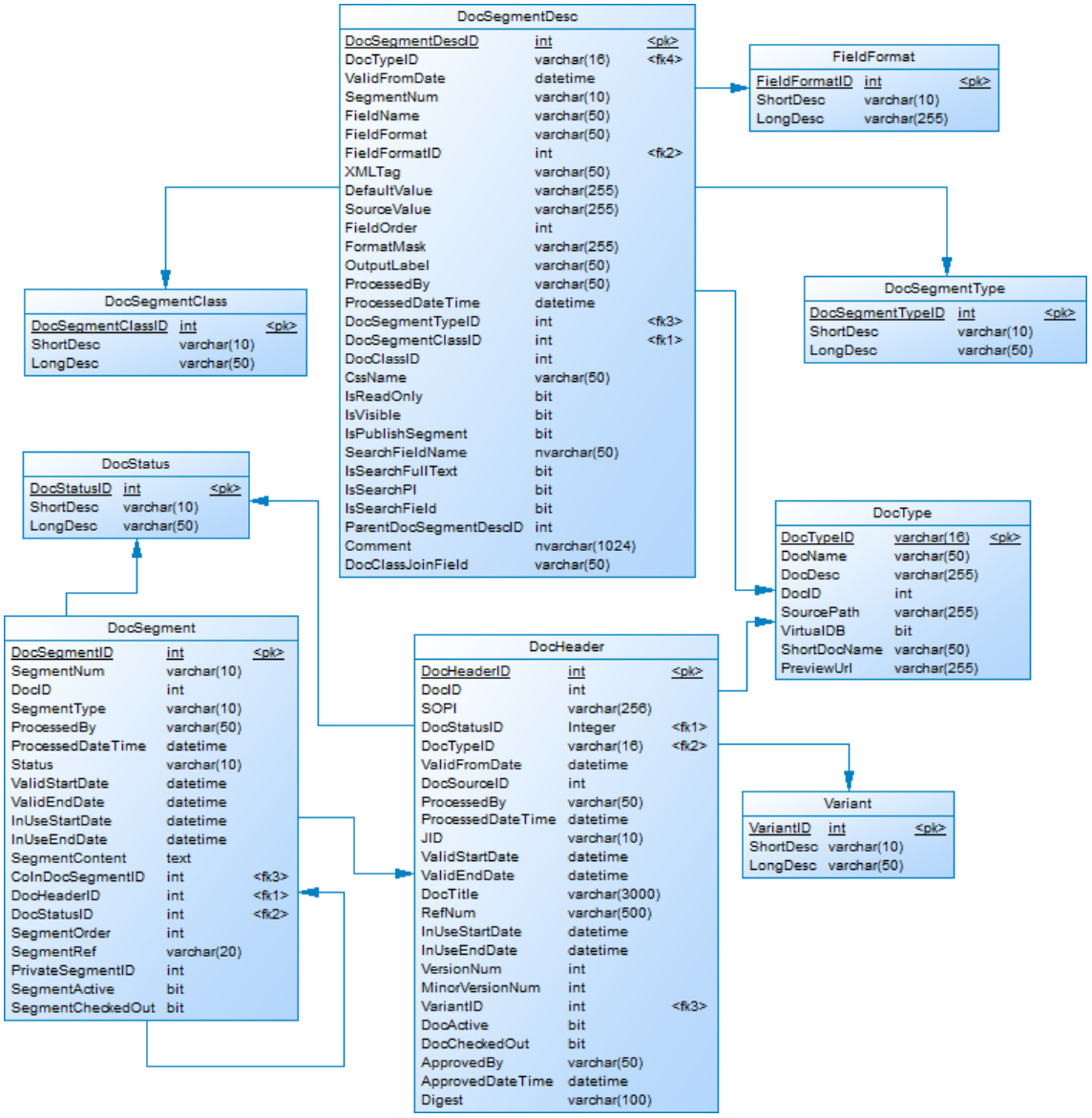
Pri tej bazi sta najpomembnejši tabeli ***DocHeader*** in ***DocSegment***. DocHeader vsebuje glavne podatke o dokumentu in o vseh njegovih verzijah. Atribut *DocHeaderID* predstavlja verzijo dokumenta, *DocID* pa originalni dokument (če sta atributa enaka, potem gledamo originalno verzijo dokumenta). *SOPI* je zunanji enolični identifikator, *DocTypeID* nam pove vrsto dokumenta. V našem primeru moramo gledati za zakoni. Šifranti za vse vrste dokumentov se nahajajo v tabeli ***DocType***, kjer so tudi atributi za bolj podroben opis. Na našem portalu si je možno ogledati vse pretekle verzije dokumentov. Te možnosti trenutna verzija naše aplikacije še ne bo imela, zato je za nas pomemben tudi atribut *ValidEndDate*. Če je datum v tem atributu enak 9999-31-12 potem gre za trenutno veljavno verzijo, katero želimo prikazati. Prav

tako moramo pogledati, če je verzija aktivna (*DocActive*). *DocStatus* pa nam pove status dokumenta ("Urejen". "V obdelavi"itn.). *DocTitle* vsebuje naslov dokumenta (npr. Zakon o upravnem postopku (ZUP)). Posamezne verzije dokumenta so tudi oštevilčene, kar zapišemo v atributa *VersionNum* in *MinorVersionNum*. Prvi atribut pomeni spremembo v vsebini dokumenta, drugi pa predstavlja tehnično verzijo dokumenta. *DocCheckedOut* nam pove, ali je dokument trenutno v obdelavi v programu za urejanje dokumentov.

Ko želimo prikazati določen člen nekega zakona, iščemo v tabeli ***DocSegment***. Atribut *DocSegmentID* predstavlja enolični identifikator zapisa. *SegmentNum* nam pove, za katero vrsto zapisa gre, vrste zapisov so shranjene v tabeli ***DocSegmentDesc***, v našem primeru iščemo zapise za člen. Atribut *SegmentContent* hrani vsebino zapisa, ki jo želimo izpisati na zaslon. Glede na atribut *SegmentNum* je zapis lahko naslov člena (npr. 1.člen ali 2.člen (področje veljavnosti), če vsebuje člen tudi naslov), vsebina člena ali celo naslov poglavja v zakonu (npr. I.SPLOŠNE ODLOČBE). V ***DocSegment*** poiščemo pravilno verzijo člena (najnovejšo) preko tujega ključa *DocHeaderID*, ki predstavlja najnovejšo verzijo dokumenta. Na voljo imamo tudi *DocID*, ki nas napoti do originalne verzije. *SegmentOrder* hrani zapise za pravilen vrstni red pri izpisu vseh členov, ki pripadajo nekemu zakonu. *SegmentRef* nam pove, kdaj se začnejo podatki o iskanem členu. V *SegmentActive* zapišemo podatek o aktivnosti segmenta. Prav tako imamo atributa *ValidStartDate* in *ValidEndDate*, ki povesta, v katerem časovnem obdobju je člen veljaven oz. je v uporabi. SegmentCheckedOut pove, ali je segment trenutno v obdelavi v programu za urejanje dokumentov. Podatke pridobimo iz baze s pomočjo bazne procedure, ki jo pokličemo znotraj aplikacijskega strežnika.

6.4.2 Podatkovna baza PortalCMS

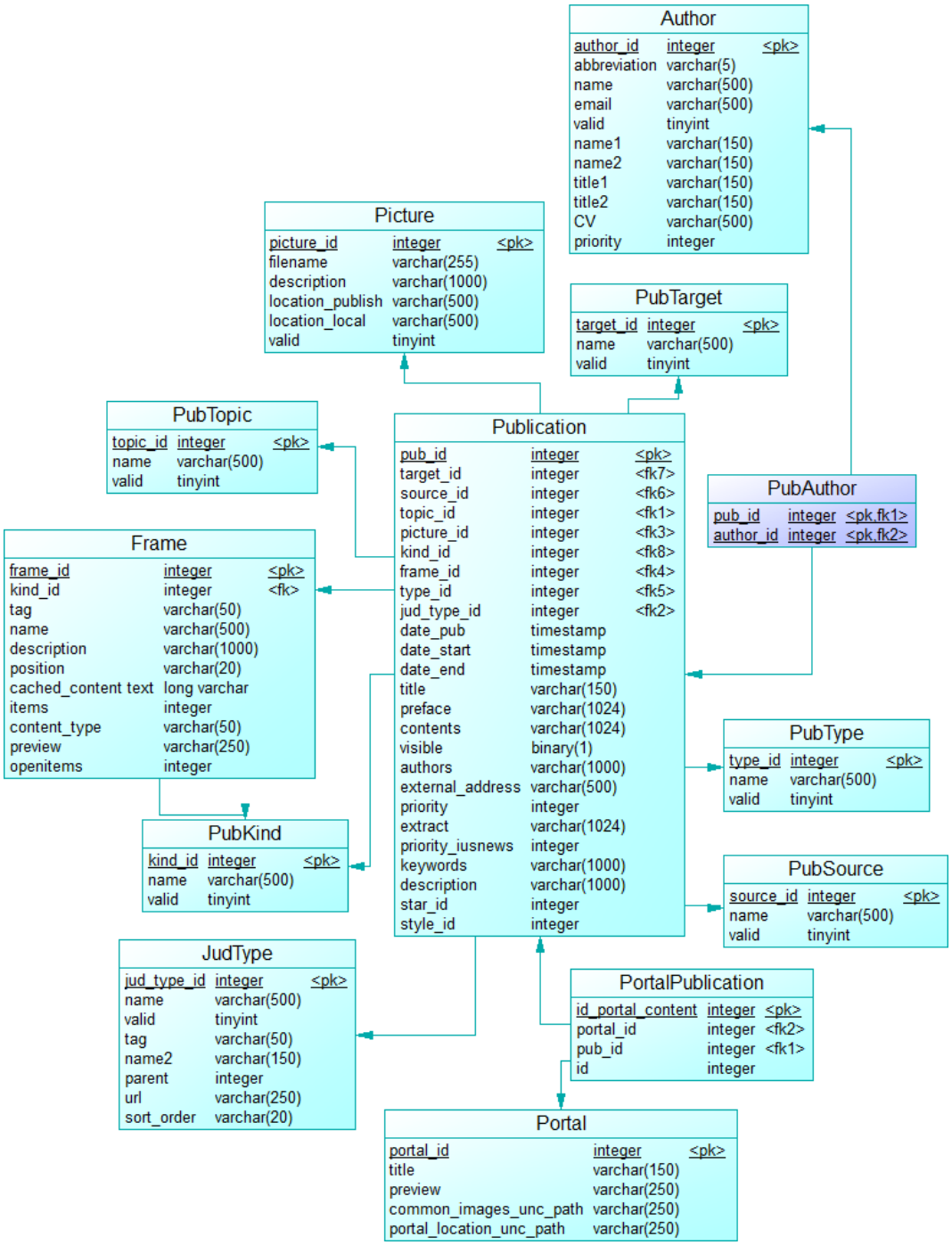
V bazi PortalCMS shranjujemo novice, kolumne, uporabnike itd. V tabeli ***Publication*** se nahajajo novice in kolumne. Pri prikazu novic oz. kolumn poiščemo najnovejših petnjast člankov in jih nato prikažemo na zaslonu. Ker sta prikaza ločeni aktivnosti v aplikaciji moramo biti pozorni na *type_id*, ki nam pove, ali gre za novico ali kolumno. *Pub_id* je primarni ključ tabele ***Publication***, v atribut *authors* pa zapišemo kratice avtorjev članka. Vsebino dobimo v stolpcih *title*, *preface* in *contents*. *Title* vsebuje naslov članka ali kolumne, v *preface* je shranjen uvod v članek, ki ga prikažemo tudi pri predogledu, ostala vsebina pa se nahaja v atributu *contents*. Atribut *visible* je binarni in nam pove, ali je članek sploh lahko viden na portalu. *Priority_iusnews* pa določa prioriteto članka. Če je prioriteta članka višja od navadne, se novica pojavi



Slika 6.2: Podatkovna baza IUS-TIME

višje v urejenem seznamu. Opisi vseh vrst člankov se nahajajo v tabeli ***PubType***, kjer članek opisuje atribut name.

Pri kolumnah prikazujemo tudi nekaj podatkov o avtorju in njegovo sliko, zato moramo dostopati do tabel ***Author*** in ***Picture***. V prvi tabeli preberemo atribute za ime (*name*), elektronski naslov (*email*), naziv avtorja (*title1*) in delovno mesto (*title2*). V tabeli Picture najdemo pravilno vrstico s pomočjo ključa *picture_id*, iz nje pa preberemo *location_publish*, ki hrani zapis o lokaciji slike na strežniku, od koder jo nato tudi prenesemo na mobitel. Pomemben je še dostop do tabele ***PortalPublication***, kjer izvemo ali gre novica na portal IUS-INFO ali FinD-INFO. Aplikacijo pišemo po vzoru prvega portala, zato prikazujemo le novice in kolumne namenjene za ta portal. Šifranti portalov pa so shranjeni v tabeli ***Portal***.



Slika 6.3: Podatkovna baza PortalCMS

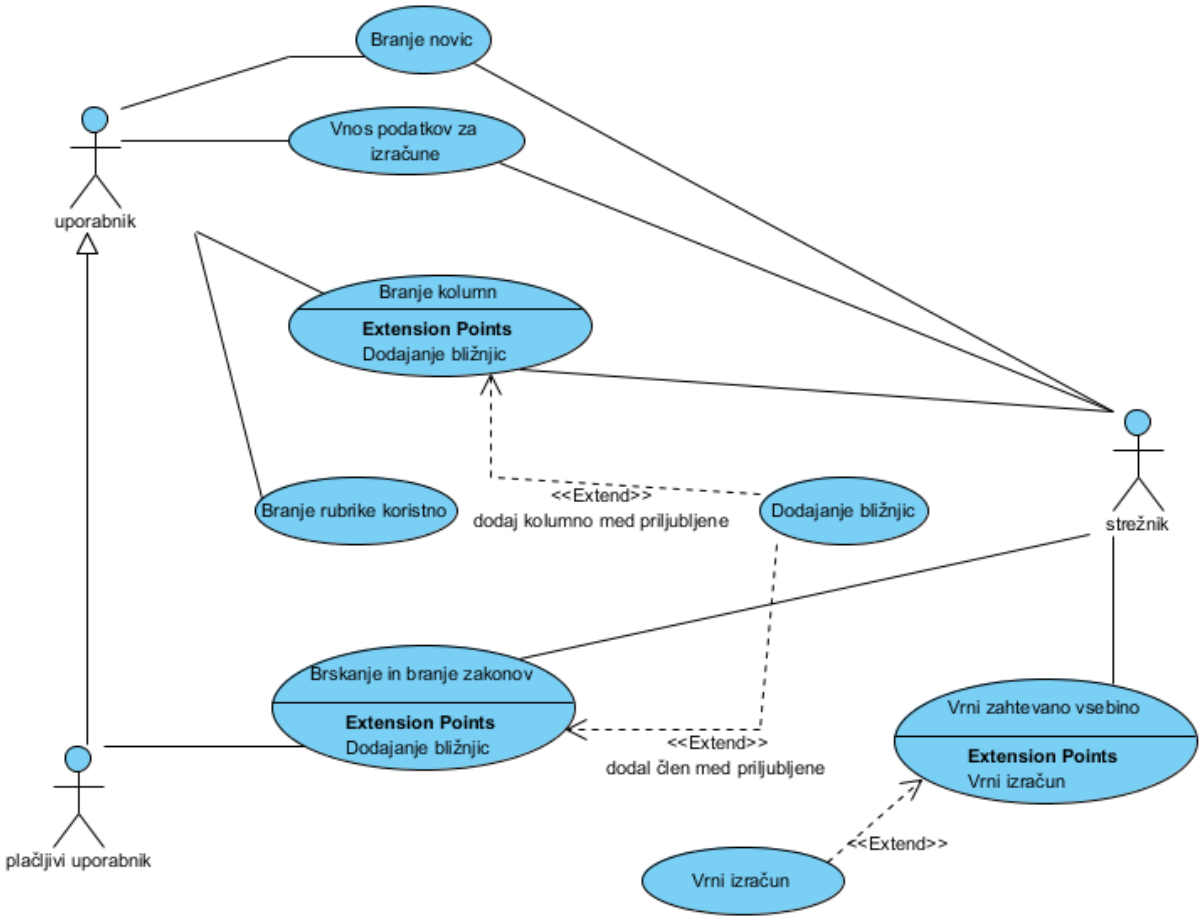
6.5 Opis aplikacije

Prvi korak pri načrtovanju aplikacije je bil določitev funkcij, ki jih želimo imeti na mobilni napravi. Na voljo smo imeli mnogo različnih vsebin, upravljanje z vsebinami (čistopisi zakonov), raznimi izračuni, webinarji. Iz množice vsebin kot so novice, kolumne, zakonodaja, sodna praksa, strokovna literatura, uradna glasila smo izbrali prve tri. Nato smo še izbrali tri vrste izračunov in sicer avtorski honorar, potne stroške in otroški dodatek. Izbrali smo jih na podlagi dejstva, da imajo najmanj potrebnih vnosov za izračun, saj nočemo imeti več kot pet vnosnih kontrol znotraj ene aktivnosti. Dodali smo še dve funkciji, ki sta se nam zdeli primerni za mobilno napravo. Prva je rubrika Koristno, kjer bomo imeli razne bližnjice do zakonov, vsebin in izračunov. Druga funkcija je rubrika Bližnjice, kjer bo lahko uporabnik dodal bližnjice do priljubljenih vsebin oz. do katerih najpogosteje dostopa.

Prav tako smo morali ločiti med uporabniki, ki imajo naročene naše storitve, in tistimi, ki jih nimajo. Slednji na spletni strani nimajo dostopa do vsebin iz zakonodaje (oz. registrirani uporabniki lahko odprejo tri dokumente na mesec), zato bo tudi v mobilni aplikaciji rubrika Zakonodaja na voljo le plačljivim uporabnikom, vendar šele po uspešnem testiranju in lansiranju aplikacije. Slika 6.4 prikazuje diagram primera uporabe (ang. use case diagram).

V naši aplikaciji smo definirali več aktivnosti (več o aktivnostih v naslednjem poglavju):

- MainActivity - prikaže začetno okno, s te aktivnosti lahko pridemo na vse ostale
- NewsListActivity - prikaže seznam novic
- NewsContentActivity - skrbi z branje novic
- ColumnListActivity - prikaže seznam kolumn
- ColumnContentActivity - branje kolumn
- KoristnoActivity - imamo aktivnost z zavihki, kjer vsebin pod izbranim zavihkom predstavlja neko drugo aktivnost
- FavouritesActivty - seznam shranjenih bližnjic
- LawListActivity - seznam zakonov
- LawContentActivity - branje členov zakonov



Slika 6.4: Diagram primera uporabe

- CalcActivity - aktivnost z zavihki, kjer imamo na voljo tri različne izračune. Vsak izračun predstavlja svojo aktivnost

6.6 Uporabniški vmesnik

6.6.1 Razvoj uporabniškega vmesnika za androidno napravo

Aktivnosti

Aplikacijo za Androida začnemo programirati z aktivnostjo (ang. Activity). Aktivnost lahko enačimo z okni pri razvoju namiznih programov. Vsako aplikacijo sestavlja ena ali več aktivnosti. Ena izmed teh mora predstavljati glavno aktivnost, s katero lahko zaženemo vse ostale. Ko zaženemo novo aktivnost, se prejšnja ustavi, vendar se njeno stanje ohrani na skladu, ki deluje po principu zadnji noter, prvi ven (ang. Last In, First Out oz. LIFO). Na prejšnjo aktivnost se vrnemo s pomočjo gumba BACK. Pri tem se trenutna aktivnost uniči. Vsako aktivnost, ki jo definiramo, moramo, če jo seveda želimo zagnati in pokazati na mobilni napravi, deklarirati v datoteki AndroidManifest.xml. Tukaj je primer deklaracije:

```
<activity
android:name="com.example.helloandroid.NewsListActivity"
android:label="IUS–INFO novice"
android:theme="@android:style/Theme.NoTitleBar">
```

Kaj pomenijo napisani atributi?

- *android:name* - tukaj določimo pot oz. ime aktivnosti
- *android:label* - tukaj določimo napis, ki gaželimo imeti v naslovni vrstici
- *android:theme* - določimo stil aktivnosti, prikazana vrstica pomeni, da ne želimo imeti prikazane privzete naslovne vrstice

Če te deklaracije nimamo v manifestu, se aktivnost ne bo zagnala in aplikacija bo podala napako, da je ne najde. V manifestu deklariramo verzijo aplikacije, verzijo razvojnega paketa, ki ga uporabljamo, in dovoljenja za aplikacijo. Dovoljenje izgleda takole:

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

Zgornji vrstici določata, da aplikacija za svoje delovanje potrebuje dostop do interneta in dostop do razširitvene kartice, na katero lahko piše in z nje tudi bere. Na ta dovoljenja smo opozorjeni ob namestitvi aplikacije na mobilno napravo.

Slika 6.5 prikazuje življenjski cikel aktivnosti. Iz slike je razvidno, katere metode imamo na voljo in kdaj se kličejo. Ob prvem zagonu aktivnosti se pokliče metoda **onCreate()**. V tej metodi poskrbimo za naložitev pogledov (npr. ListView), naložimo sezname in jih povežemo z gradniki (npr. Naložimo seznam novic in ga povežemo z ListView, da lahko vidimo celoten seznam novic na zaslonu). Sledi ji aktivnost **onStart()**, ki se kliče tik preden postane aktivnost vidna uporabniku. Slednji lahko sledi bodisi metoda **onResume()** bodisi **onStop()**. Prva se kliče v primeru, ko prihaja aktivnost v ospredje, druga pa v primeru, če se skrije v ozadje. Metoda **onPause()** se kliče, kadar hoče sistem poklicati v ospredje drugo aktivnost oz. aplikacijo (npr. Nekdo vas kliče, medtem ko igrate igro). Tukaj poskrbimo za shranitev raznih podatkov in ustavimo animacije. Metoda ne sme biti preveč časovno zahtevna, saj aktivnost, ki želi priti v ospredje, čaka na zaključek le-te. Že omenjena metoda **onStop()** se kliče tudi v primeru, ko se aktivnost zaključuje oz. uniči. Sledi ji lahko metoda **onRestart()**, če se aplikacija le potisne v ozadje, ali metoda **onDestroy()**, če se aktivnost dokončno uniči. Naj omenimo še, da se v primeru spremembe orientacije zaslona (mobitel obrnemo za 90 stopinj) takoj pokliče metoda **onDestroy()** in takoj za njo še metoda **onCreate()**, kar nam lahko pokvari prikazane podatke. Zato moramo poskrbeti za shranitev stanja ali pa z vrstico *android:screenOrientation="portrait"* v datoteki AndroidManifest.xml zaklenemo zaslon na le en možen prikaz.

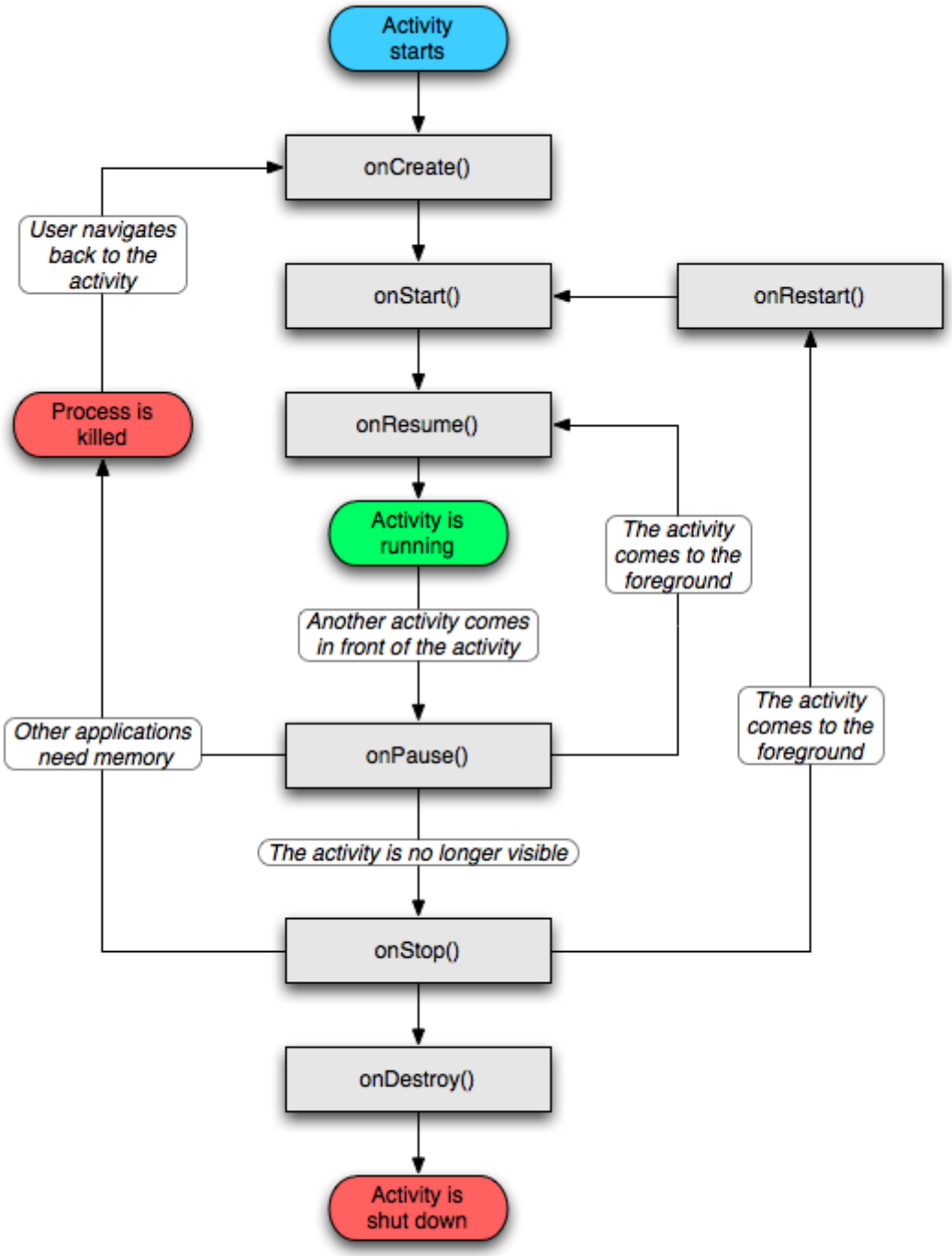
Vsaka aplikacija ima lahko več aktivnosti, primer imamo na sliki 6.6. Novo aktivnost zaženemo z naslednjim klicem:

```
Intent intent =
        new Intent(MainActivity.this, KoristnoActivity.class);
startActivity(intent);
```

Pri kreiranju namere (ang. Intent) podamo argumenta, kjer prvi predstavlja aktivnost, v kateri se nahajamo, drugi pa aktivnost, ki jo želimo zagnati. S klicem *startActivity* pa novo aktivnost dejansko zaženemo.

Izdelava uporabniškega vmesnika

Uporabniškega vmesnika se lahko lotimo na tri načine. Prvi je programski, kjer najprej ustvarimo objekt za razporeditev (npr. LinearLayout) in ga določimo za aktivnost s klicem *setContentView(view)*. Nato lahko dodajamo razne kon-



Slika 6.5: Življenjski cikel aktivnosti pri aplikaciji za Androida

Slika 6.6: Sprožitev nove aktivnosti

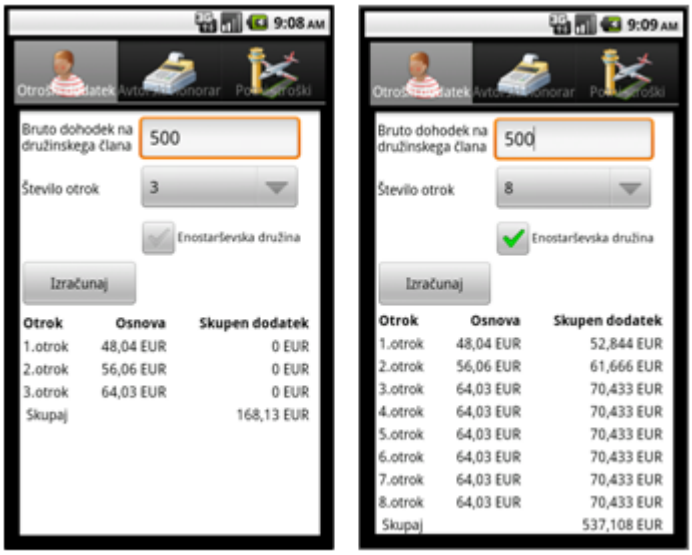
trole (npr. TextView) in jim določamo lastnosti, ki bodo vidne na zaslonu. Primer:

```
LinearLayout nKids = new LinearLayout(getBaseContext());
nKids.setOrientation(LinearLayout.VERTICAL);

TextView textView = new TextView(getBaseContext());
textView.setTextColor(Color.BLACK);
textView.setText("Izpis besedila");
textView.setTypeface(Typeface.DEFAULT, Typeface.BOLD);
nKids.addView(textView);
setContentView(nKids);
```

LinearLayout predstavlja t.i. ViewGroup, skupino pogledov, v katero dodajamo ostale poglede, ki so mu podrejeni. ViewGroup je osnovni razred za razporeditve in ostale kontrole, ki lahko delujejo kot vsebniki (ang. containers). Možnih je kar nekaj:

- LinearLayout - določimo lahko navpično ali vodoravno razporeditev podrejenih pogledov.
- AbsoluteLayout - tukaj lahko določimo x in y koordinate za natančno postavitev pogledov, vendar je takšno stanje težje vzdrževati.
- FrameLayout - uporaben, če želimo na zaslonu prikazati le en element (npr. sliko).



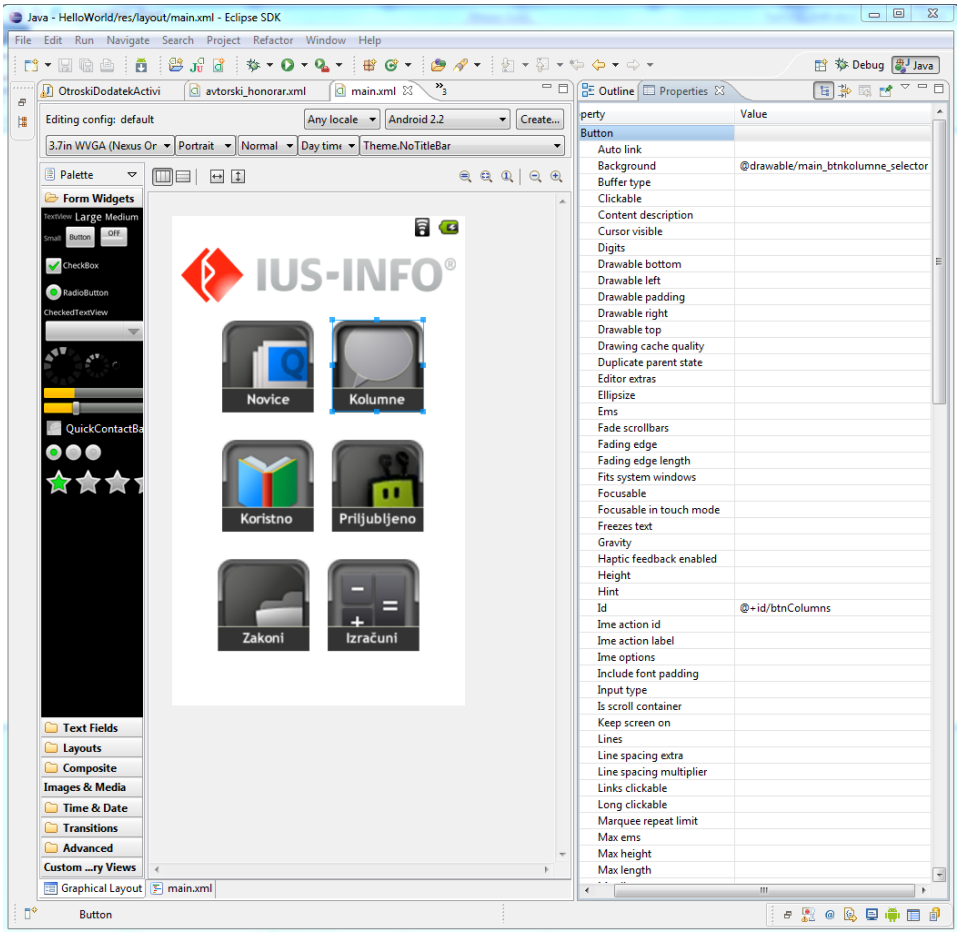
Slika 6.7: Primer programskega dodajanja elementov

- RelativeLayout - položaj podrejenega pogleda lahko podamo glede na položaj drugega podrejenega pogleda ali pa ga določimo glede na nadrejeni element.

V zgornjem primeru smo ustvarili pogled TextView, mu določili besedilo, barvo besedilo in odebeljeni stil. Nato smo pogled dodali v razporeditev nKids. S končnim klicom setContentView(nKids) povežemo razporeditev in aktivnost. Programiranje na tak način je zamudno, vendar včasih priročno. V našem primeru je koristno, ko hočemo prikazati rezultate izračuna otroškega dodatka, kjer je število rezultatov odvisno od podanega števila otrok. Slika 6.7 prikazuje izpis pri različno podanem številu otrok.

Naslednja dva načina sta medsebojno povezana. Uporabniški vmesnik lahko naredimo s pomočjo grafičnega urejevalnika. Medtem ko postavljamo elemente na zaslon, se nam gradi datoteka XML. Ročno urejanje slednje predstavlja tretji način izdelave uporabniškega vmesnika. Slika 6.8 prikazuje urejevalnik. V njem lahko določimo naslednje lastnosti izgleda:

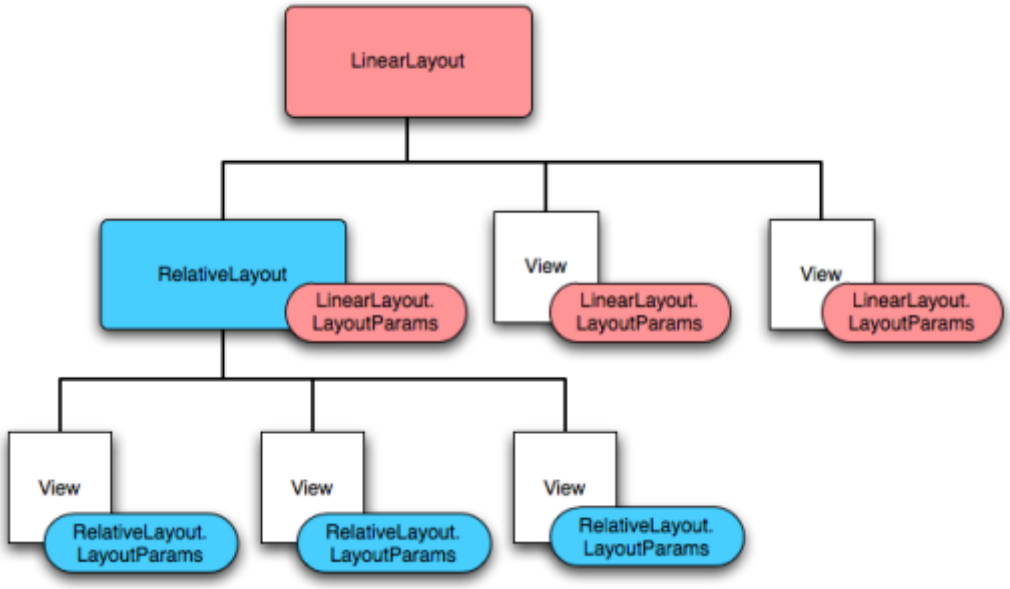
- Izbira velikosti ekrana v inčih
- Pokončna ali položna postavitev
- Tema izgleda (ali gre za osnovni zaslon, vrsto dialoga, vnos podatkov, prikaz z naslovno vrstico ali brez, itd.)



Slika 6.8: Urejevalnik uporabniškega vmesnika

- Izberemo tudi verzijo Androida, za katero delamo aplikacijo

Na sliki 6.8 so na levi strani vidne kontrole, ki jih lahko dodajamo na zaslon. Na voljo imamo osnovne kontrole, kot so TextView (labela za izpis besedila), Button (gumb), RadioButton in CheckBox (izbirni gumbi), EditText (okno za vpis besedila), pri katerem imamo že v naprej pripravljene različne možnosti oz. prednastavljene lastnosti. Takoj lahko postavimo okno za vpis gesla, okno, ki sprejema le vpis številk, datuma, časa, eletronskega naslova itn. Naslednje kontrole so razne razporeditve (Layouts), ki sem jih že naštel zgoraj. Sledijo jim kontrole za medijske vsebine (ImageView za slike, VideoView za video, Gallery za zbirko slik), za izbiranje časa in datuma (DatePicker za datum, TimePicker za čas, prav tako lahko dodamo ali analogni ali digitalni prikaz



Slika 6.9: Prikaz hieararhije pogledov in odvisnosti parametrov za razporeditev od nadrejenega pogleda

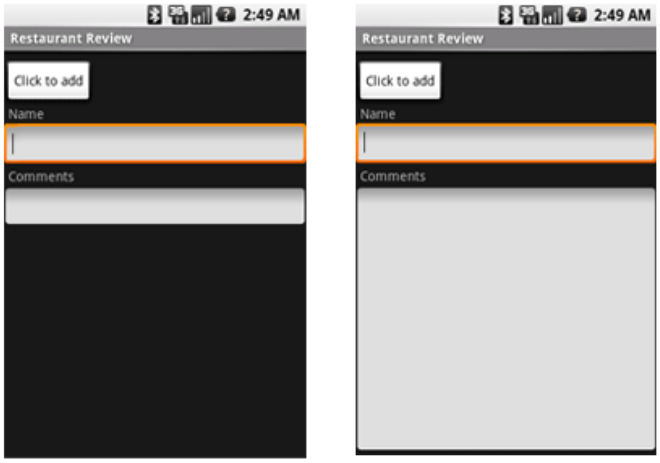
ure). Nato imamo na voljo kontrole za animirano menjavanje pogledov, t.i. switcher-ji oz. menjalniki. Na koncu naj omenim še kontrolo GestureOverlayView, s pomočjo katere lahko zaznamo prednastavljene geste in posledično sprožimo zahtevane akcije. Dober primer je poteg s prstom z desne proti levi, kar sproži prikaz naslednje novice na seznamu. Geste, ki jih želimo uporabiti, lahko naredimo kar znotraj emulatorja, na katerem je nameščena aplikacija Gestures Builder. Podamo ime geste in jo oblikujemo z miško. Vse poteze, ki smo jih naredili, se shranijo v datoteko gestures na emulatorjevi kartici. Če želimo gesto dejansko uporabiti v naši aplikaciji, jo moramo prenesti na disk in jo dodati projektu med vire. Prenesemo jo s pomočjo ukaza pull v ukazni vrstici (ang. Command prompt). Ko dodamo element, lahko urejamo njegove lastnosti v oknu *Properties* (si. lastnosti). Obvezno moramo nastaviti lastnosti za višino (Layout height) in širino (Layout width). Določimo ju lahko številčno ali pa vpišemo privzeti vrednosti *MATCH_PARENT* ali *WRAP_CONTENT*. Prva določi velikost na podlagi nadrejenega pogleda, druga pa na podlagi lastne vsebine. Vsak pogled, ki pripada razredu ViewGroup vsebuje podrazred LayoutParams, kjer se definirata velikost in pozicija za vsak podrejeni pogled. Slednje je prikazano na slike 6.9. Vse kontrole, ki jih dodamo, se zapišejo v datoteko R.java. Kadar želimo spreminjati lastnosti kontrole znotraj kode,

moramo za dostop do nje uporabiti klic *findViewById(R.id.ime_kontrole)*. Dostop je možen, ko določimo ime kontrole v lastnosti *Id*. Spodaj je seznam lastnosti, ki jih imajo praktično vse kontrole in se najpogosteje uporabljajo v naši aplikaciji:

- Layout gravity - določimo položaj znotraj nadrejenega elementa za podrejenega
- Layout height - določimo višino elementa
- Layout margin - določimo, kolikšen naj bo dodaten prazen prostor okoli kontrole na vseh štirih straneh
- Layout margin_bottom(top, left, right) - določimo prazen prostor okoli kontrole za vsako stran posebej
- Layout weight - določimo delež zasedenosti kontrole na zaslonu (primer različnih vrednosti prikazuje slika 6.10
- Layout width - določimo širino elementa
- Visibility - določimo vidnost elementa
- Style - določimo povezavo do lastnega stila, če ga imamo
- Clickable - ali je omogočen klik na kontrolo
- Background - določimo ozadje kontrole (ponavadi dodamo kakšno sliko)
- Id - določimo ime elementa, preko katerega se sklicujemo nanj v programski kodi. Ime je še posebej pomembno pri razporeditvi RelativeLayout, kjer se elementi razporedijo glede na drug drugega.

Tretji in zadnji način ustvarjanja uporabniškega vmesnika je ročno urejanje xml datoteke. Začnemo s korenskim elementom, ki predstavlja neko razporeditev (nekaj razporeditev smo opisali zgoraj). Slednjemu nato dodajamo podrejene elemente in jim določamo lastnosti (v tem primeru jim lahko rečemo kar atributi). Delni izpis xml datoteke, v kateri smo definirali začetno stran naše mobilne aplikacije:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
```



Slika 6.10: Prikaz različnih vrednosti za lastnost weight, na levi strani ima spodnje okno za tekst vrednost 0, na desni pa 1

```
          android:layout_height="fill_parent"
          android:orientation="vertical"
          android:background="@color/White">
          <LinearLayout
android:layout_height="wrap_content"
                    android:background="@drawable/logo_iusinfo2"
                    android:layout_width="wrap_content"
... />
     <LinearLayout
          android:layout_height="wrap_content"
          android:layout_width="wrap_content"
          android:id="@+id/linearLayout3">
       <Button
android:layout_height="100dip"
                    android:layout_width="100dip"
                    android:id="@+id/btnNews"
                    ....
android:background="@drawable/main_btnnovice_selector">
</Button>
       <Button
android:layout_height="100dip"
                    android:layout_width="100dip"
                    android:id="@+id/btnColumns"
                    ....
                    android:background=
                    "@drawable/main_btnkolumne_selector">
</Button>
```

Slika 6.11: Slika začetnega okna IUS-INFO aplikacije

```
    </LinearLayout>
....
```

Končni rezultat zgornjega izpisa je prikazan na sliki 6.11.

Viri

Vsi zunanji viri, katere uporabljamo za izdelavo grafičnega vmesnika, so shranjeni v direktoriju *res*. Tabela 6.2 prikazuje, katere vrste virov so nam na voljo in v katero podmapo jih shranimo. Zgornja tabela prikazuje osnovne mape, ki jih uporabimo za dostop do potrebnih virov za oblikovanje uporabniškega vmesnika. Ima pa orodje priročno možnost za izdelavo vmesnika za več različnih naprav, ne da bi morali za vsako napravo imeti lastni projekt. To orodje so alternativni viri.

Alternativne vire naredimo po naslednjem ključu. Ustvarimo novo podmapo v mapi res v obliki ¡ime_vira¿-¡ime_kvalifikatorja¿. Vnesemo lahko več kvalifikatorjev, le te pa moramo med seboj ločiti s pomišljajem. Operacijski sistem Android zazna konfiguracijo mobilne naprave in potegne vire iz lokacije, ki se najbolje ujemajo z napravo. Poglejmo najprej mapo drawable, kamor lahko shranimo vse ikone, ki jih imamo v aplikaciji. Če jo delamo le za telefone z določeno resolucijo, več kot mape drawable ne potrebujemo. Ko pa želimo aplikacijo uporabiti tudi na napravah z nižjo ali višjo resolucijo, potrebujemo

Podmapa	**Tip vira**
anim/	Xml datoteke, ki definirajo animacije
color/	Xml datoteke, ki definirajo različne barve
drawable	Slike (png, jpg, gif) ali xml datoteke, v katerih definiramo: - Slike - Seznam stanj - Oblike (kvadrat, krog) - Tranzicijske animacije
layout/	Xml datoteke, ki definirajo uporabniški vmesnik.
menu/	Xml datoteke, ki definirajo animacije
raw/	Tukaj shranimo geste, ki smo jih naredili z graditeljem gest
values/	Xml datoteke, v katerih imamo shranjene razne vrednosti (barve, števila, nize). Tukaj lahko v xml datoteko shranimo več vrednosti. Zaradi jasnosti je najboljše, če ima vsak tip vrednosti svojo datoteko. Primer: - Color.xml za barvne vrednosti - String.xml za shranjevanje nizov - Styles.xml za definicije stilov - Arrays.xml za sezname
xml/	Tukaj shranimo razne druge xml datoteke, ki ne spadajo nikamor drugam

Tabela 6.2: Tipi virov

tudi ikone z nižjo oz. višjo resolucijo. V tem primeru ustvarimo nove podmape med viri, in sicer:

- drawable-hdpi/icon.png
- drawable-ldpi/icon.png
- drawable-mdpi/icon.png

Kot lahko vidimo, imamo v vsaki podmapi datoteko z enakim imenom. Mobilna naprava vzame tisto, ki pripada njeni konfiguraciji. Naprave z nizko ločljivostjo bodo vzele ikono iz podmape ldpi (ang. low density per inch, slo. nizka gostota slikovnih pik), višje s hdpi(high density per inch) in srednje z mdpi(medium density per inch) podmape. Pomemben je tudi kvalifikator za orientacijo zaslona pri podmapi layout. Možnosti sta dve in sicer port (ang. Portrait, slo. pokončno) in land (ang. Landscape, slo. ležeče). Enostavno naredimo dva različna uporabniška vmesnika, enega v pokončnem položaju, drugega v ležečem, za pravilno uporabo bo poskrbel operacijski sistem Android.

Pravila pri podajanju kvalifikatorjev

Pri podajanju več kvalifikatorjev je pomemben vrstni red, ki mora slediti zaporedju, kakršen je v tabeli 6.3, v kateri so opisani tudi vsi možni kvalifikatorji.

Konfiguracija	**Vrednosti kvalifikatorjev**	**Opis**
MCC in MNC	Primer: mcc293-mnc41	MCC predstavlja kodo države, MNC pa kodo omrežja. V primeru številka 293 predstavlja Slovenijo, 41 pa operaterja Mobitel[1].
Jezik in regija	Primer: En Fr En-rUS Fr-rCA	Jezik definiramo s pomočjo dvo-črkovne kode ISO-639-1[2] , sledi ji pa dvo-črkovna koda ISO-3166-1[3] za državo, oznaka r pred to kodo označuje, da gre za regijo.

[1]http://maps.mobileworldlive.com
[2]http://www.loc.gov/standards/iso639-2/php/code_list.php
[3]http://www.iso.org/iso/iso-3166-1_decoding_table.html

Najmanjša širina	sw<N>dp Primeri: sw320dp sw720dp	Kolikšna je najmanjša širina ekrana, ki je na voljo. Upošteva tudi elemente grafičnega vmesnika, ki so vedno na vrhu, tako da je najmanjša širina manjša od dejanke širine zaslona. Se ne spreminja z orientacijo zaslona.
Širina, ki je na voljo	w<Ndp>	Upošteva orientacijo zaslona, določa pa najmanjšo širino, pri kateri se lahko uporabijo viri. Če definiramo več direktorijev, se uporabi tisti, ki je najbližji, a ne preseže predpisane širine.
Višina, ki je na voljo	h<N>dp	Velja enako kot za širino.
Velikost ekrana	small normal large xlarge	Velikost ekrana glede na gostoto
Vidik zaslona	long notlong	Zaslon, ki spada pod long, je širši. Ni povezano z orientacijo zaslona.
Orientacija zaslona	port land	Port - naprava je v pokončnem položaju. Land - naprava je v ležečem položaju. Orientacija se spreminja, ko uporabnik obrača napravo.
Priklopna postaja	car desk	Ali je naprava priključena na namizni polnilec ali na avtomobilski polnilec.
Nočni način	night notnight	Nočni način je odvisen od ure.
Gostota slikovnih pik	Ldpi Mdpi Hdpi Xhdpi Nodpi tvdpi	Ekrani z nizko gostoto. (c. 120dpi) Srednja gostota (c. 160dpi) Visoka gostota (240dpi) Zelo visoka gostota (320 dpi) Za vire, ki niso odvisni od gostote. Zasloni, ki padejo med mdpi in hdpi.

Tip zaslona na dotik	notouch stylus finger	Naprava nima zaslona na dotik. Zaslon na dotik, ki uporablja pisalo. Naprava ima zaslon na dotik
Razpoložljivost tipkovnice	keysexposed keyshidden keyssoft	Na voljo je tipkovnica, programska ali strojna. Strojna tipkovnica, ki pa je skrita (zaprta). Omogočena programska tipkovnica, a ni nujno, da je vidna.
Glavni način za vnost podatkov	nokeys qwerty 12key	Naprava nima fizičnih tipk za vnos podatkov. Fizična QWERTY tikpovnica. Klasičen vnos pri telefonih je na voljo.
Razpoložljivost navigacijskega gumba	navexposed navhidden	Navigacijski gumbi so na voljo. Navigacijski gumbi niso na voljo.
Glavni način za navigacijo, ki ni zaslon na dotik	nonav dpad trackball wheel	Na voljo je samo navigacija z zaslono na dotik. Naprava ima smerne tipke. Naprava ima sledilno kroglico. Naprava ima koleščk.

Tabela 6.3: Pravila za podajanje kvalifikatorjev

Če bi torej poimenovali direktorij ***drawable-hdpi-port***, bi bilo to narobe, ker je v tabeli orientacija zaslona pred gostoto slikovnih pik. Pravilni zapis bi bil ***drawable-port-hdpi***. V imenu direktorija imamo lahko samo en zapis za vsak tip kvalifikatorja. Če želimo uporabiti isti vir za dve različni regiji, ne smemo poimenovati direktorija *drawable-rSI-rEN*. Ustvariti moramo dva direktorija, in sicer *drawable-rSI* in *drawable-rEN*. V tem primeru bi lahko prišlo do nepotrebnega podvajanja virov, kar samo poveča velikost aplikacije. Slednjemu se izognemo tako, da vir shranimo v direktorij *drawable/ime_vira*, nato pa v direktorijih za različni regiji ustvarimo xml datoteko, v kateri shranimo pot do potrebnega vira. Primer xml datoteke, ki se imenuje icon.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<bitmap xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@drawable/ime_vira" />
```

V programski kodi se nato sklicujemo na to xml datoteko, ki deluje kot kazalec do vira. Na tak način prihranimo pri prostoru. Do virov lahko dostopamo s pomočjo programske kode ali znotraj xml datotek, v katerih smo definirali uporabniški vmesnik. V slednjem primeru uporabimo naslavljanje na sledeči način: @ime_mape/ime_vira.

Naslavljanje virov

Vsi viri, ki jih ali dodamo kot slike, xml datoteke, kontrole, ki prihajajo z uporabniškega vmesnika, se shranijo v avto-generirani datoteki R.java. Te datoteke sami ne smemo spreminjati, saj bo prišlo pri prevajanju do napake. Ko želimo znotraj kode uporabiti nek vir, ga naložimo po naslednjem ključu:

- ¡ime_paketa¿.R.¡vrsta_vira¿.¡ime_vira¿

Ime paketa ni nujno potrebno, kadar naslavljamo vire, ki izhajajo iz našega paketa. Vrsta vira predstavlja podrazred v razredu R. Primere smo si že ogledali v tabeli 5, saj imena poddirektorijev ustrezajo imenom podrazredov. Nekaj primerov dostopa do virov:

- setContentView(R.layout.activity_column);

S tem ukazom naložimo uporabniški vmesnik, ki smo ga naredili s pomočjo urejevalnika za uporabniški vmesnik.

- TextView columnTitle = (TextView)findViewById(R.id.ColumnTitle);

Zgoraj smo naložili pogled za besedilo, ki se nahaja v eni izmed definicij uporabniškega vmesnika.

- MenuInflater(getApplication()).inflate(R.menu.menu_activity_law, menu);

S tem klicem smo naložili elemente opcijskega menu, ki so definirani v xml datoteki. Primer take xml datoteke se nahaja v naslednjem poglavju.

Kadar želimo naslavljati vire znotraj xml datoteke, pa uporabimo naslednjo sintakso: @[¡ime_paketa¿:].¡vrsta_vira¿/¡ime_vira¿

Primer takega izpisa:

```
<Button
android:layout_height="100dip"
android:layout_width="100dip"
android:background="@drawable/main_btnnovice_selector" />
```

Kot lahko vidimo zgoraj, gremo po vir v direktorij drawable, ki se imenuje main_btnnovice_selector.

Poleg lastno izdelanih virov imamo na voljo nekaj že narejenih, ponje pa gremo v paket android. (npr. : android.R.layout.simple_list_item_1).

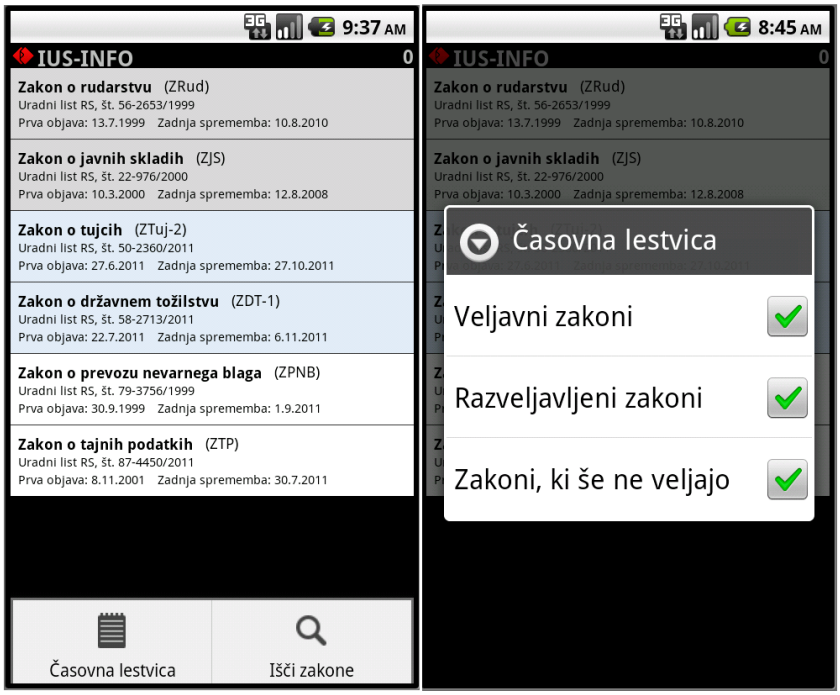
Menu

Meniji so pomemben del uporabniške izkušnje, zato ima Android možnost kreiranja preprostih menujev, ki jih delimo v tri skupine:

- Opcijski menu (Options Menu) - menu se pojavi, ko pritisnemo na gumb MENU na mobilni napravi
- Kontekstni menu (Context Menu) - ko nekaj časa držimo neko kontrolo (uporabnik izvede t.i. long press na elementu s seznama), se pojavi plavajoči menu z raznimi možnostmi (npr. izbris zapisa)
- Podmenu (Submenu) - Plavajoči menu, ki se pojavi, ko kliknemo možnost na opcijskem menuju

Možnosti za menu lahko ustvarimo v programski kodi, vendar Android priporoča, da uporabimo xml datoteke in jih definiramo kot vir. Vire smo opisali že v prejšnjem poglavju, zato naj samo povem, da so datoteke z definicijami menujev shranjene v poddirektoriju *menu*. Spodaj imamo primer xml datoteke za definiranje menujev. Slika 6.12 pa prikazuje rezultat na zaslonu:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item
android:id="@+id/filterMenu"
        android:title="Časovna lestvica"
android:orderInCategory="1"
android:icon="@drawable/cas_lestvica_notepad">
            <menu>
                        <group android:checkableBehavior="all">
                        <item android:id="@+id/valid_law"
                              android:title="@string/valid"
                              android:checked="true"/>
                        <item android:id="@+id/not_valid_law"
                              android:title="@string/not_valid"
                              android:checked="true"/>
                        <item android:id="@+id/not_yet_valid_law"
                              android:title="@string/not_yet_valid"
                              android:checked="true" />
                    </group>
            </menu>
   </item>
  <item
android:id="@+id/searchMenu"
android:orderInCategory="2"
android:title="Išči zakone"
```



Slika 6.12: Leva stran slike prikazuje opcijski meni, desna pa podmeni za opcijo Časovna lestvica

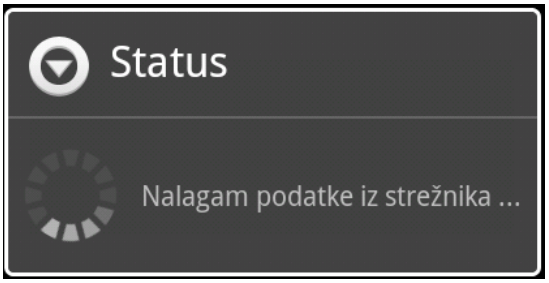
```
android:icon="@drawable/search" />
</menu>
```

Nastaviti moramo atribut *id*, da vemo na katero možnost smo pritisnili, *orderInCategory* določa vrstni red izrisa, *title* vsebuje besedilo, ki se prikaže. Nastavimo lahko tudi atribut *icon*, če želimo pri prikazu imeti tudi sliko. Iz primera xml datoteke in slike 6.12 je razvidno, da je podmenu podrejeni element v xml zapisu.

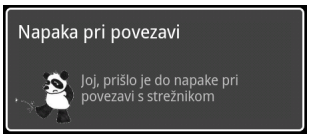
Pogovorna okna

Pogovorno okno (ang. Dialog) se pojavi pred trenutno aktivnostjo. Fokus aplikacije se preseli na okno in čaka bodisi na uporabnikov odgovor bodisi le obvesti uporabnika o trenutnem stanju aplikacije ter čaka na konec trenutnega opravila. Poznamo več vrst pogovornih oken:

- AlertDialog - najbolj priporočljiv za uporabo. Vsebuje lahko do tri gumbe in seznam predmetov, ki so lahko v obliki radijskih gumbov ali potrditvenih polj.



Slika 6.13: Primer pogovornega okna ProgressDialog

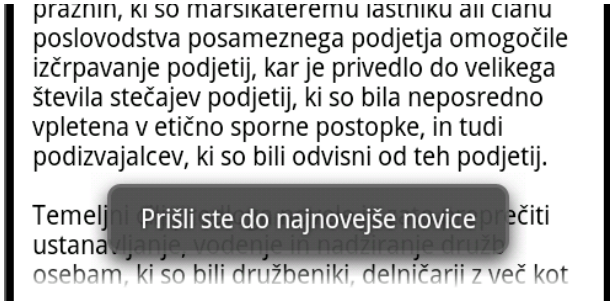


Slika 6.14: Primer lastno izdelanega pogovornega okna

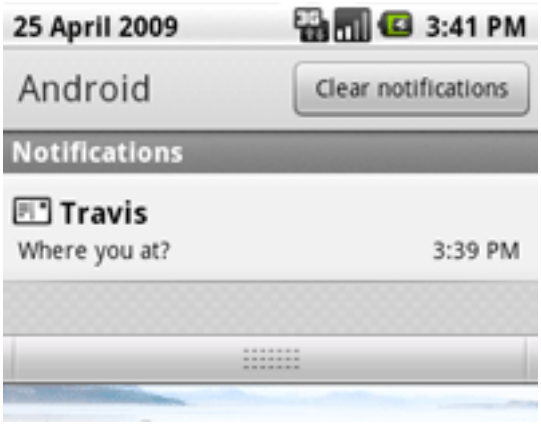
- ProgressDialog - okno, ki prikaže vrstico ali kolo napredka. Primer na sliki 6.13.
- DatePickerDialog - okno za izbiro datuma
- TimePickerDialog - okno za izbiro časa
- CustomDialog - lahko ustvarimo lastno pogovorno okno, razporeditev lahko definiramo v xml datoteki ali v programski kodi. Primer lastnega pogovornega okna imamo na sliki 6.14.

Poleg pogovornih oken imamo na voljo tudi t.i. Toast sporočila. Toast je pojavno okno, ki se prikaže v ospredju aplikacije. Zasede samo toliko prostora, kakor je dolgo sporočilo. Primeren je za krajša sporočila, kot lahko vidimo na sliki 6.15. Uporabimo ga, kadar smo prepričani, da uporabnik spremlja dogajanje na zaslonu. Določimo mu lahko dolžino prikaza, na voljo sta dve vrednosti (LENGTH_SHORT in LENGTH_LONG).

Naj na koncu omenim še sporočila, ki jih uporabnik prejme v statusni vrstici. Uporaba je primerna, kadar uporabljamo storitev v ozadju. Za primer lahko vzamemo aplikacijo za elektronsko pošto, ki v ozadju preverja prihod nove pošte in o tem obvesti uporabnika. Opomni ga lahko s pomočjo besedila, zvoka, vibracije in utripajoče lučke na mobilni napravi. Ko pritisnemo na obvestilo, ponavadi sprožimo novo aktivnost, ki nam prikaže podrobnosti o obvestilu. Na sliki 6.16 lahko vidimo primer takega sporočila.



Slika 6.15: Primer sporočila Toast



Slika 6.16: Primer sporočila v statusni vrstici

Graditelj gest

Ko uporabljamo naprave, ki imajo zaslon na dotik, pričakujemo, da bomo imeli možnost upravljanja tudi z gestami. Prav v ta namen imamo pri razvoju aplikacij na voljo orodje graditelj gest (ang. Gestures Builder). Orodje je vgrajeno v emulator za Androida. Zaženemo ga kot navadno aplikacijo. Na zaslonu se nam prikaže črn ekran, na spodnjem delu imamo gumb za dodajanje gest (ang. Add gesture). Kliknemo na gumb, naredimo gesto in jo poimenujemo ter nato shranimo. Vse narejene geste se shranijo v datoteko gestures na simulirani kartici na emulatorju. Datoteko prenesemo na trdi disk s pomočjo ukazne vrstice (ang. Command prompt) z ukazom pull. V programu naložimo datoteko kot knjižnico gest, nastavimo poslušalca, ki spremlja geste in ob zadetku sproži zahtevano akcijo. Primer naložitve in poslušalca za geste, ki smo ju implementirali v naši aplikaciji:

```
GestureOverlayView gestures = findViewById(R.id.gestures);
gestures.addOnGesturePerformedListener(this);
GestureLibrary libraries = GestureLibraries.fromRawResource


public void onGesturePerformed(GestureOverlayView overlay,
Gesture gesture) {
  ArrayList<Prediction> predictions =

  if (predictions.size() > 0) {
     Prediction prediction = predictions.get(0);
     String action = predictions.get(0).name;
     if (prediction.score > 1.0) {
       Toast.makeText(this, prediction.name,
        Toast.LENGTH_SHORT).show();
         if("swipe_l2r".equals(action)) {
           btnNewsPrevious();
         }
         else if("swipe_r2l".equals(action)) {
           btnNewsNext();
         }
     }
  }
}
```

6.6.2 Razvoj uporabniškega vmesnika za iOS napravo

Medtem ko smo lahko pri Androidu rekli, da vsaka aktivnost predstavlja svoje okno, to za iOS aplikacije ne velja. Tukaj imamo na voljo le eno okno, ka-

Slika 6.17: Graditelj gest

teremu pa lahko menjamo različne poglede. Prav tako ni priporočljivo, da bi aplikacija tekla v ozadju, čeprav je to postalo možno z izdajo četrte verzije operacijskega sistema iOS. Naslednja pomembna razlika je pomanjkanje avtomatskega sproščanja pomnilnika. Medtem ko se pri Androidu avtomatsko sprosti pomnilnik, ko ne uporabljamo več nekega objekta, moramo za to pri iOS aplikacijah poskrbeti sami.

Kot sem že omenil v poglavju o platformi iOS, bomo za razvoj uporabniškega vmesnika uporabljali ogrodje UIKit, ki nam omogoča dostop do kontrol za uporabniški vmesnik, gumbov in pogledov. Vsaka aplikacija v iOS ima en UIApplication, ki predstavlja začetno točko in je odgovorna za inicializacijo in prikaz okna UIWindow. Prav tako poskrbi za nalaganje pogleda UIView v okno UIWindow in skrbi za življenjski cikel aplikacije. UIApplication izpolni vse te zahteve s pomočjo zastopnika (UIApplication Delegate). Medtem ko UIApplication prejema zahteve, zastopnik odloči, kako se bo aplikacija odzvala na te klice. Ko ustvarimo nov projekt se generirajo metode, ki se sprožijo v različnih trenutkih življenjskega cikla aplikacije. Te metode so naslednje:

- (bool)didFinishLaunchingWithOptions - tukaj se inicializirajo začetni pogledi in vse kar želimo sami inicializirati
- (void)applicationWillResignActive - sproži se, kadar gre aplikacija iz ak-

tivnega v neaktivno stanje. Pri telefonih je to v primeru prejema klica ali SMS sporočila. Najpogosteje se to zgodi, ko uporabnik ugasne aplikacijo in se le ta premakne v ozadje. V tej metodi moramo poskrbeti za začasno zaustavitev opravil, ustaviti merilnike časa, če jih imamo. Pri igrah moramo poskrbeti za premor.

- (void)applicationDidEnterBackground - tukaj shranimo podatke, sprostimo vire in shranimo stanje aplikacije, v primeru, da se bo zaključila, preden jo ponovno zaženemo. Metoda se pokliče samo v primeru, če aplikacija podpira izvajanje v ozadju.
- (void)applicationWillEnterForeground - aplikacija se pomakne iz ozadja v neaktivno stanje. V tej metodi lahko razveljavimo vse spremembe, ki smo jih naredili ob klicu metode applicationDidEnterBackground
- (void)applicationDidBecomeActive - ponovno zaženemo vsa zaustavljena opravila in osvežimo uporabniški vmesnik, če je le to potrebno
- (void)applicationWillTerminate - metodo pokličemo, ko zaustavljamo aplikacijo. Če je potrebno, shranimo podatke.
- (vodi)dealloc - metoda, kjer sprostimo vse objekte.

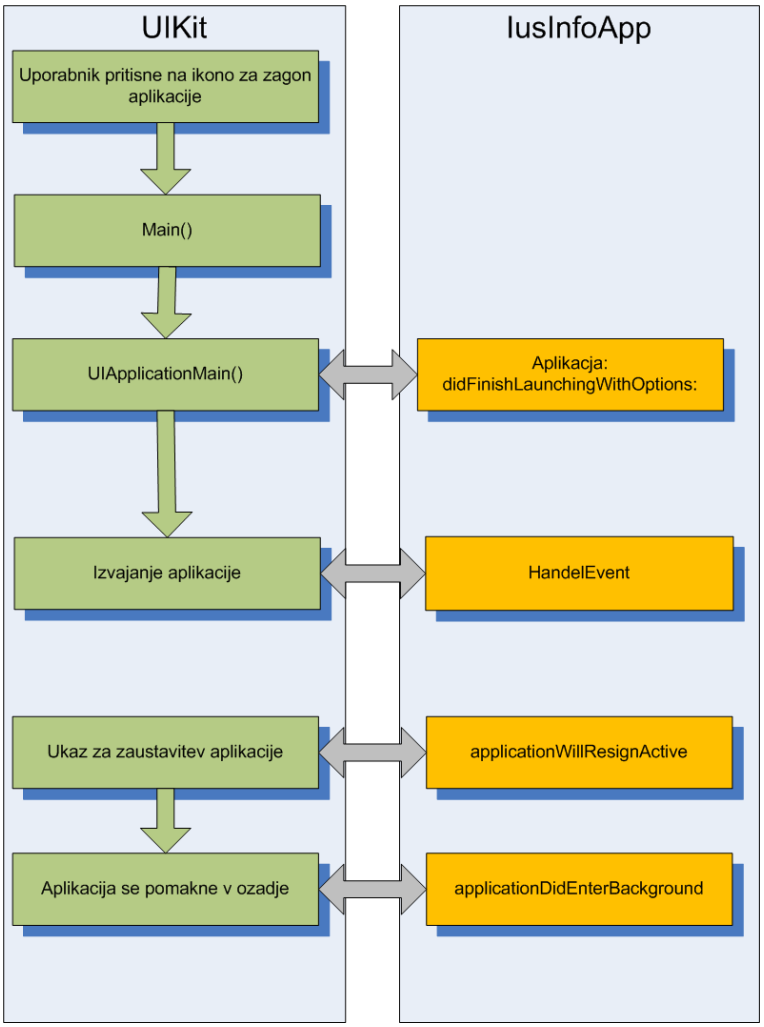
Primer življenjskega cikla aplikacije lahko vidimo na spodnji sliki 6.18.

Naj še enkrat omenimo, da je tak cikel postal možen šele s četrto verzijo operacijskega sistema iOS. Pred njo se je namesto metod *applicationDidEnterBackground* in *applicationWillResignActive* poklicala metoda *applicationWillTerminate*.

Izdelava uporabniškega vmesnika

Ko začnemo nov projekt, nam Xcode ponudi že nekaj vnaprej pripravljenih razporeditev za uporabniški vmesnik. Med naslednjimi možnostmi se odločimo za tisto, ki se nam zdi najbolj primerna za našo aplikacijo:

- Aplikacija z navigacijskim krmilnikom, kjer imamo navigacijsko vrstico tik pod statusno vrstico na vrhu zaslona (ang. Navigation-based application).
- Aplikacija z razdeljenim pogledom (ang. Split View-base Application). Pri tej aplikaciji dobimo uporabniški vmesnik, kjer imamo glavni pogled (izbira elementa na seznamu) in podrobni pogled (ki je prikazan glede na izbiro v glavnem pogledu).



Slika 6.18: Primer življenjskega cikla aplikacije

- Aplikacija z zavihki (ang. Tab Bar Application). Na voljo imamo zavihke, enako možnost imamo pri Androidu z uporabo gradnika TabHost.
- Aplikacija z dvema pogledoma (ang. Utility Application). Na voljo imamo glavni pogled in zrcalni pogled (ang. flipside) ter dva gumba. S pritiskom na prvega dobimo zrcalni pogled, kjer je na voljo drugi za vrnitev na glavni pogled.
- Aplikacija z enim pripravljenim pogledom (ang. View-based Application).
- Aplikacija z enim oknom (ang. Window-based Application). Osnovna aplikacija na katero sami dodajati elemente.

Ko začnemo delati na novem projektu, se moramo odločiti ali bomo delali aplikacijo za mobilni telefon iPhone ali tablični računalnik iPad. Obstaja še tretja možnost, in sicer univerzalna. To pomeni, da bo aplikacija tekla na obeh napravah. Pri slednji možnosti moramo seveda poskrbeti za vse možne poglede in jih tudi ustvariti. Nato določimo, katere orientacije bomo podpirali. Medtem ko smo imeli pri Androidu tri možnost (normalna pokončna, telefon obrnemo za 90 stopinj na levo ali na desno), imamo pri iOS na voljo štiri orientacije. Poleg že omenjenih imamo na voljo še obratno pokončno, kjer napravo obrnemo za 180 stopinj. Sledi možnost za izbiro ikone, ki bo predstavljala aplikacijo. Na koncu lahko še določimo sliko, ki se prikaže ob zagonu programa. Naložimo lahko sliki za pokončno in ležečo verzijo programa. Glede na izbrano vrsto aplikacije na začetku projekta dobimo različne generirane datoteke. Pri aplikaciji z razdeljenim pogledom dobimo že ustvarjene datoteke za zastopnika (AppDelegate), datoteke za glavni pogled in datoteke za podrobni pogled.

Same izdelave uporabniškega vmesnika se lahko lotimo s pomočjo orodja Interface Builder. Datoteke, ki shranijo vse narejene vmesnike, se imenujejo nib datoteke. Naj tukaj omenimo, da ima uporabniški vmesnik tudi xml predstavitev, vendar na žalost ni namenjen ročnemu urejanju, ker je težje berljiv od Androidove xml datoteke. XML datoteke nimajo na voljo edinstevnih označb, kot je na primer oznaka TextView pri Androidu. Pri iOS ima vsaka kontrola za oznako ključ (ang. Key), kateri lahko nato spreminjamo standardne lastnosti (v primeru TextView so to velikost, vrsta pisave, stil pisave ...). Iskanje in spreminjanje te datoteke zato ni priporočljivo. Primer izpisa za kontrolo TextView v iOS:

```
<key>71</key>
<dict>
```

```
                <key>autoresizingMask</key>
                <integer>36</integer>
                <key>frameOrigin</key>
                <string>{4, 14}</string>
                <key>frameSize</key>
                <string>{721, 71}</string>
                <key>text</key>
                <string>V senci boja proti terorizmu</string>
</dict>
```

Interface Builder je zelo zmogljivo orodje, kjer lahko ustvarimo povezave s kodo le s potegom miške. Primer nastanka take povezave lahko vidimo na sliki xx. Pri tem se bodisi povežemo z že deklariranim elementom v kodi bodisi program sam ustvari vse potrebne spremenljivke v vseh razredih. Poskrbi tako za inicializacijo, kakor tudi za uničenje elementa v metodi dealloc(). Na tak način lahko povežemo tudi metode. To nam omogočata direktivi IBOutlet in IBAction. IBOutlet pove orodju Interface Builder, da je bila spremenljivka ustvarjena za njegovo uporabo. Za povezavo moramo še vedno poskrbeti sami. Akcije so sporočila, ki jih pošiljajo objekti iz nib-a metodam zunaj nib datoteke. Metodo definiramo v kodi s pomočjo ukaza IBAction. Nobena metoda, ki je na tak način povezana z nib datoteko, ne sme vračati nikakršnih vrednosti, ker prevajalnik zapis IBAction prevede v void metodo oz. v metodo, ki nič ne vrača. Taki metodi lahko podamo samo en parameter, ki se mu reče pošiljatelj (ang. sender). Pošiljatelj predstavlja identifikaciajo (id) kontrole, ki kliče akcijo. Če bi imeli gumb UIButton, ki kliče IBAction metodo spremeniIme, bi bil pošiljatelj kazalec na ta gumb. Vsaka spremenljivka, ki mora komunicirati s kontrolo znotraj orodja Interface Builder, mora biti definirana kot IBOutlet in vsaka metoda, ki jo pokliče neka kontrola, mora biti definirana kot IBAction. Primer deklaracije za IBOutlet in IBAction:

IBOutlet:

```
@property (nonatomic, retain) IBOutlet UITextView *TitleText;
```

IBAction:

```
- (IBAction)spremeniIme:(id)sender;
```

Pri prikazu uporabniškega razreda sta pomembna razreda UIView in UIViewController. UIView je pogled, ki skrbi za prikaz vmesnika na zaslonu. UIViewController pa predstavlja krmilnik, ki upravlja s pogledom. Najboljše bi ga opisali kot vez med pogledom, ki ga predstavlja UIView in podatkovnimi razredi. UIView skrbi za izris na zaslonu in dogodke, ki so posledica uporabnikove interakcije z aplikacijo. Krmilnik UIViewController pa skrbi za nalaganje in sprostitev pogledov, upravlja z njimi in skrbi za njihov življenjski cikel.

Podrazredi pogleda UIView	**Podrazredi**
UIWindow	
UILabel	
UIPickerView	
UIProgressView	
UIActivityIndicatorView	
UIImageView	
UITabBar	
UIToolBar	
UINavigationBar	
UITableViewCell	
UIActionSheet	
UIAlertView	
UIScrollView	UITableView
	UITextView
UIWebView	
UIControl	
UIButton	
	UIDatePicker
	UIPageControl
	UISegmentedControl
	UITextField
	UISlider
	UISwitch
UISearchBar	

Tabela 6.4: Podrazredi razreda UIView

Vsaka grafična kontrola, ki je na voljo v orodju Interface Builder, je podrazred pogleda UIView. V tabeli TABELA so našeteti vsi podrazredi. Iz tabele je tudi razvidno, da imata nadaljnje podrazrede še pogleda UIScrollView in UIControl.

Kot sem že omenil zgoraj, krmilnik UIViewController skrbi za življenjski cikel pogleda. Tega ne smemo zamešati z življenjskim ciklom aplikacije. Metode, ki skrbijo za ta cikel, so naslednje:

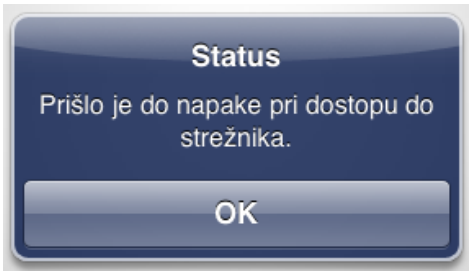
- *didReceiveMemoryWarning* - sproži se, ko krmilnik prejme opozorilo glede spomina
- *didRotateFromInterfaceOrientation* - sproži se, ko obrnemo zaslon

- *viewDidAppear* - sproži se, ko se prikaže pogled
- *viewDidDisappear* - pogled je izginil
- *viewDidLoad* - pogled se je naložil v spomin
- *viewDidUnload* - pogled je bil uničen
- *viewWillAppear* - pokliče se tik, preden se pogled prikaže
- *viewWillDisappear* - pokliče se tik, preden pogled izgine
- *willRotateToInterfaceOrientation:duration:* - pokliče se, ko začnemo obračati zaslon
- *shouldAutoRotateToInterfaceOrientation* - pri tej metodi določimo ali dovolimo, da se spremeni orientacija zaslona, ko obračamo napravo

Ko ustvarjamo aplikacijo, moramo poskrbeti, da bo vsak pogled, ki predstavlja neko vsebino (npr. Kolumna in novice), imel definirano svojo datoteko nib. Lahko bi sicer združili vse vmesnike v eno datoteko, vendar bi v tem primeru vse naenkrat naložili v spomin in ne le takrat, ko bi vmesnik potrebovali. Pri Androidu je enakovredna možnost definirati grafični vmesnik in ga naložiti v lastno aktivnost. Razvoj preprostega uporabniškega vmesnika lahko strnemo v naslednje korake. Najprej ustvarimo podrazred krmilnika UIViewController v Xcode, nato naredimo pogled UIView v datoteki nib. Pogled nato povežemo iz nib datoteke v krmilnik.

Sedaj si bomo pogledali še nekaj kompleksnejših pogledov s krmilniki. Začeli bomo s pogledom z zavihki UITabBar in s krmilnikom UITabBarController. Pogled z zavihki vsebuje dva ali več zavihkov, kjer ima vsak zavihek svoj krmilnik. Ko uporabnik izbere zavihek, se najprej naloži krmilnik, ta pa nato poskrbi za nalaganje pogleda. Podoben princip uporablja Android, kjer začnemo graditi vmesnik s TabHost razporeditvijo in ob pritisku na zavihek naložimo izbrano aktivnost. Naslednji pogled je navigacijski pogled UINavigationBar, ki se nahaja tik pod statusno vrstico na vrhu zaslona. Običajno ima na voljo dva gumba, na levi strani je gumb, ki nas vrne na prejšnji pogled, na desni pa gumb, ki nas vodi do naslednjega pogleda. Pogled je primeren za sezname, kjer vrtamo globje v podatke. Dober primer je brskanje po programih v AppStore aplikaciji za mobitel iPhone, kjer izberemo IGRE, nato se nam pokaže izbira žanra, v navigaciji pa je na voljo gumb, ki nas bo popeljal nazaj na vrste programov. Vrtamo pa lahko tudi naprej do željene aplikacije. Na voljo imamo tudi pogled s tabelami UITableView. Tabele imajo lahko le en

Slika 6.19: Primer uporabe pogleda UIProgressView



Slika 6.20: Primer opozorilnega okna za iOS

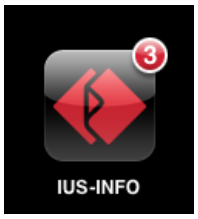
stolpec in več vrstic. Zaradi teh omejitev ga lahko primerjamo z ListView razporeditvijo na Androidu. Obadva pogleda predstavljata nek seznam elementov, na katerega lahko kliknemo in nas vodi v naslednji pogled ali aktivnost. Vsaka vrstica vsebuje celico, katero lahko preuredimo po svoje. Tako kot pri Androidu lahko tudi tukaj ustvarimo področja (npr. urejanje po abecedi, kjer se vsako področje začne s črko). Pri iOS imamo na voljo še združevalni stil (UITableViewStyleGrouped).

Pogovorna okna

Včasih mora uporabnik počakati, da se zaključi neka aktivnost ali pa želi aplikacija uporabnika le opozoriti oz. obvestiti o svojem delovanju. Pri iOS imamo za to na voljo razreda UIActivityIndicatorView in UIProgressView. Prvi prikazuje krog, ki se vrti in nam s tem pove, da aplikacija izvaja aktivnost v ozadju. Drugi pa nam sporoča napredek aktivnosti v ozadju. Pri Androidu je za tako sporočanje na voljo pogovorno okno ProgressView, ki mu obliko (vrteči krog ali okno s prikazom napredka) določimo kar v atributu.

Za opozarjanje uporabnika uporabljamo pogled UIAlertView in njegovega zastopnika UIAlertViewDelegate. Na zaslonu se pojavi pred pogledom in čaka na uporabnikovo rekacijo. Zastopnik poskrbi za pritiske različnih gumbov.

Slika 6.21: Primer okna UIActionSheet



Slika 6.22: Primer uporabe značk (Application Badges)

Za opozorila imamo na voljo tudi pogled UIActionSheet, ki je zelo podoben pogovornemu oknu. Za razliko od pogleda UIAlertView se ne prikaže na zaslonu, ampak prileze na zaslon s spodnje ali zgornje stranice zaslona.

Namesto opozorila v statusni vrstici pri Androidu iOS uporablja t.i. Application Badges (aplikacijske značke). Uporaba je preprosta. Vse, kar moramo narediti, je, da določimo vrednost značke:

```
[UIApplication sharedApplication].applicationIconBadgeNumber = 3;
```

Značke lahko uporabimo, če želimo narediti funkcijo, ki v ozadju preverja, ali so na voljo nove novice. Funkcija bi nove novice naložila in s pomočjo značke obvestila, da ga čaka določeno število novih novic. Primer obvestila z značkami si lahko ogledamo na spodnji sliki (6.22).

Med ostala pogovorna okna spadata še pogleda UIDatePicker, ki se uporablja za izbiro datuma in časa, in UIPickerView, kjer izberemo neko vrednost (npr. državo).

Viri

Vire shranjujemo v projekt, kjer pa ni potrebna stroga hierarhija, kot je bilo to potrebno pri Androidu. Za hierarhijo virov poskrbimo sami, da se ne izgubimo, ko upravljamo z njimi. Sliko naložimo, tako da povemo ime datoteke, nato pa

Slika 6.23: Pogled za izbiranje elementov UIPickerView

aplikacija sam pogleda pobrska po svoji datotečni strukturi, da najde pravilno datoteko. Primer nalaganja slike:

```
UIImage *image = [UIImage imageNamed:@"ime_slike"]
```

Shranjujemo lahko tudi razne sezname v datoteke plist (property list), in se nato sklicujemo na to datoteko kot vir podatkov (ang. Data source). Podobno smo naredili pri aplikaciji za Androida.

Razred za prepoznavo gest

Za prepoznavo gest imamo na voljo razred UIGestureRecognizer. Sestavljajo ga podrazredi za šest različnih gest, in sicer dotik (ang. tap), stisk (pinch), povlek (ang pan ali drag), poteg (ang. swipe), vrtenje, (ang. rotate) in dolge pritiske (ang. long press):

- *UITapGestureRecognizer* - diskretna metoda, parametra sta numberOfTapsRequired in numberOfTouchesRequired (touch predstavlja število prstov, ki se istočasno dotikajo zaslona)
- *UIPinchGestureRecognizer* - zvezna metoda, parametra sta lestvica (ang. scale) in hitrost (ang. velocity)
- *UIRotationGestureRecognizer* - zvezna metoda, parametra sta rotacija in hitrost.
- *UISwiperGestureRecognizer* - diskretna metoda, parametra sta smer (ang. diretion) in število dotikov.
- *UIPanGestureRecognizer* - zvezna metoda, parametra sta najmanjše in največje število dotikov (ang. minimumNumberOfTouches).

- *UILongPressGestureRecognizer* - zvezna metoda, parametri so število potrebnih kratkih dotikov (ang. numberOfTapsRequired), število potrebnih dotikov (ang. numberOfTouchesRequired), najmanjši čas dotika (ang. minimalPressDuration) in dovoljeno gibanje (ang. allowableMovement). Slednji parameter določa, za koliko se lahko premakne prst, da bomo še vedno zaznali dolgi dotik. Vključen je, ker prst redkokdaj stoji popolnoma na miru na isti lokaciji.

Primer kode, ki prikazuje odziv na dvojni dotik (ang. double tap). Najprej inicializiramo gesto in ji določimo metodo, ki se bo odzvala na dvojni dotik. Določimo še potrebno število dotikov:

```
UITabGestureRecognizer *t = [[UITapGestureRecognizer alloc]
        initWithTarget: self
        action: @selector (doubleTap) ];
t.numberOfTapsRequired = 2;
[view addGestureRecognizer:t];
[t release];
```

Metoda, ki se odzove na dvojni dotik (v konzolo izpišemo besedno zvezo "dvojni dotik"):

```
-(void) doubleTap{ NSLog(@"dvojni dotik");}
```

Po naših izkušnjah je orodje pri Androidu lažje za uporabo, ker si lahko izmislimo kakršno koli gesto s potegom miške. Ta možnost je odlična za začetnike pri programiranju mobilnih aplikacij, ker lahko implementirajo geste hitro in učinkovito. Pri iOS napravah nam implementacija gest vzame več časa.

Slika 6.24: Primer uporabniškega vmesnika na iPadu

Poglavje 7

Primerjava razvoja za Androida in iOS napravo

Prva razlika, ki jo opazimo, je, da potrebujemo za razvoj aplikacij za iOS naprave računalnik Mac, na katerem teče operacijski sistem MacOS. Le ta lahko poganja razvojno orodje Xcode, katerega potrebujemo za oblikovanje uporabniškega vmesnika in pisanje programske kode. Razvoj za Androida pa poteka v razvojnem okolju Eclipse, ki je na voljo za različne operacijske sisteme (Windows, Mac OS X, Linux). Naslednja razlika je v uporabi programskega jezika. iOS aplikacije se programira v jeziku objective-C, medtem ko androidne aplikacije pišemo v programskem jeziku Java. Pri iOS ni omogočeno avtomatično sproščanje pomnilnika, kar nam nekoliko oteži razvoj, saj moramo sami poskrbeti za sprostitev objekta, ko ga ne potrebujemo več. Obe razvojni okolji ponujata urejevalnik uporabniškega vmesnika (ADT vtičnik za Eclipse za Androida in Interface Builder za iOS). Pri obeh imamo na voljo vpogled v xml datoteko, ki predstavlja uporabniški vmesnik. Androidova xml predstavitev je preglednejša, saj značke ponujajo boljši opis kontrole. Kontrola za tekst bi pri Androidu izgledala takole:

```
<TextView android:id=''txt'' android:text=''primer značke''/>
```

Značka pove, da gre za kontrolo, v kateri prikažemo besedilo, lastnosti pa določimo s pomočjo atributov. Pri iOS bi struktura xml datoteke izgledala takole:

```
<key>12</key>
<dict>
        <key>text</key>
        <string>primer značke </key>
</dic>
```

Značka ključ (key) predstavlja identifikacijo kontrole, nato pa značko dict, čigar otroci predstavljajo lastnosti kontrole.

Apple je šele s četrto verzijo iOS dovolil razvijalcem, da se lahko njihove aplikacije izvajajo v ozadju oz. ni omogočal več-opravilnosti(ang. multi-tasking). Pred tem se je aplikacija ustavila in sprostila spomin. Kljub temu, da so to omogočili,pa še vedno predlagajo ustavitev celotne aplikacije ob izhodu iz nje. Nasprotno aplikacija za Androida že od začetka ponuja več-opravilnost. Pri obeh moramo poskrbeti za shranitev podatkov, ko se aplikacija umakne v ozadje.

Android ponuja orodje za izdelavo gest. Orodje je primerno za začetnike, ki želijo hitro narediti aplikacije in uporabiti kakršne koli geste. Apple pa je šele v tretji verziji operacijskega sistema iOS dodal metode za nekaj standardnih gest. Pred tem je moral vsak razvijalec sam napisati razred za prepoznavanje gest.

Uporabnika obveščamo o spremembah s pomočjo pojavnih oz. pogovornih oken. Pri obeh operacijskih sistemih poteka prikaz na podoben način. Kadar se aplikacija izvaja v ozadju, imamo pri iOS možnost prikazati spremembe s pomočjo značk (ang. badges), pri Androidu pa lahko uporabnika obvestimo s pomočjo zapisa v statusni vrstici. Pri prenosnih telefonih lahko sprožimo še vibracijo in/ali zvočni signal.

Manjše število fizičnih gumbov pri iOS pomeni, da moramo več funkcij, ki so pri Androidu na voljo preko gumbov (opcijski menu, iskanje in vračanje na prejšnji pogled), narediti s pomočjo uporabniškega vmesnika. Na tega moramo postaviti gumbe in sprogramirati njihove funkcije.

Pomembno je tudi testiranje aplikacije na dejanskih mobilnih napravah. Število naprav, ki jih poganja iOS, lahko preštejemo na prste(iPad 1 in 2, štiri verzije iPhone-a, iPodTouch), tako da ni težko preveriti, če naša aplikacija deluje povsod. Naprav, ki jih poganja Android, pa je mnogo več, zato je testiranje težje in posledično ne moremo jamčiti, da bo aplikacija delala na vseh.

Slika 7.1 prikazuje prvi zaslon na telefonu Nexus One in na tabličnem računalniku iPad. Zaradi večjega ekrana lahko pri iPadu prikažemo več podrobnosti. Pri novicah in kolumnah tako prikažemo naslov in uvod najnovejše vsebine, pri ostalih pa kratek opis.

Na sliki 7.2 lahko vidimo razlike pri prikazu vsebine. Na manjšem ekranu prikažemo vsebino, podrobnosti o avtorju pa se pokažejo kot pojavno okno. iPad pa lahko prikaže avtorja in kratko predstavitev v istem pogledu.

Slika 7.1: Primer glavnega zaslona na telefonu Nexus One in iPadu

Slika 7.2: Primer kolumne pri telefonu Nexus One in iPadu

Poglavje 8

Zaključek

Aplikacije postajajo mobilne. Danes ni več dovolj, da podjetje ponuja svoje storitve na namiznem računalniku, pa naj bo namizna aplikacija ali spletna stran. Ljudje hočejo ali celo potrebujejo dostop do informacij na poti. Tak dostop do informacij omogočajo mobilne naprave, kot so pametni mobilni telefoni in tablični računalniki. Vse te mobilne naprave poganjajo različni operacijski sistemi, kar pomeni, da mora podjetje za čim večjo uporabo svojih storitev podpirati vse te različne naprave. V okviru diplomske naloge smo razvili mobilno aplikacijo za naprave, katere poganja operacijski sistem Android in si pogledali, kako bi potekal razvoj na tabličnem računalniku iPad.

Vsaka platforma ima za razvoj aplikacij na voljo lastna orodja. To pomeni, da mora biti razvijalec podkovan v več programskih jezikih in načinih dela, ki jih ponujajo različna delovna okolja. V podjetju za razvoj aplikacij uporabljamo .NET ogrodje in programski jezik C#. Preskok na okolje Eclipse in programski jezik Java ni bil pretirano težak, saj se sintaksi jezikov ne razlikujeta preveč. Večji težave smo imeli pri uporabi delovnega okolja Xcode 4 in programskega jezika objective-C, pri katerem je sintaksa ravno toliko drugačna, da smo se morali navaditi na nov način razmišljanja.

Obe okolji imata svoje prednosti in slabosti. Xcode ima boljši urejevalnik uporabniškega vmesnika (Interface Builder), dodatek za Eclipse pa omogoča boljše upravljanje raznih virov. Dodajanje gest v aplikacije je pri Androidu lažje za začetnike, ker lahko z vgrajenim orodjem (Gesture Builder) naredijo kakršno koli gesto, ki bo naredila točno to, kar želijo. Knjižnica za iOS naprave pa ponuja samo razred, ki vsebuje nekaj predpripravljenih standardnih gest. Poganjanje emulatorjev poteka hitreje v okolju Xcode, kar je posledica tega, da je Xcode namensko orodoje za razvoj aplikacij za iOS naprave, medtem ko moramo za Androida namestiti dodatek za okolje Eclipse. Pri iOS razvoju

moramo paziti še na ročno sproščanje pomnilnika, ker objective-c za iOS (še) ne podpira avtomatičnega sproščanja pomnilnika, kot je to mogoče pri Javi (garbage collector)

Razlika se pojavi tudi pri fizični sestavi naprav. Androidove naprave imajo na voljo več namenskih gumbov. Tako imamo na voljo gumb BACK, s katerim se lahko vračamo na pretekle aktivnosti. Pri iOS moramo za tako preklapljanje poskrbeti sami. Androidne naprave vsebujejo še gumb MENU, kjer z lahkoto določimo razne menujske možnosti, medtem ko moramo za iPada ponovno sami poskrbeti za razvoj podobnega uporabniškega vmesnika.

Dostop do podatkov, ki jih prikazujemo na mobilni napravi, smo realizirali s spletno storitvijo RESTful. Spletna storitev, ki temelji na REST pravilih zahteva manjšo količino podatkov. Manjša količina podatkov pa je zelo pomembna za mobilne naprave, ker prenos podatkov ni poceni.

Končni rezultat diplomske naloge je delujoča mobilna aplikacija IUS-INFO za pametne telefone, katere poganja operacijski sistem Android. Uporabnikom omogočamo dostop do informacij, ki so bile doslej dostopne le preko brskalnika. Zraven smo spoznali še način dela za naprave iOS. Znanje za razvoj aplikacij za iOS naprave nam bo prišel prav v prihodnosti, ko bomo želeli razviti še popolnoma delujočo aplikacijo za iPada.

Literatura

[1] "Nexus One Owner's Guide.", Dostopno na: http://static.googleusercontent.com/external_content/untrusted_dlcp /www.google.com/sl//googlephone/nexusone-userguide.pdf, 15.3.2010 [27.9.2011].

[2] M. Neuberg, *Programming iOS 4*. Sebastopol: O'Reilly Media, Inc., 2011.

[3] R. Meier, *Android 2 Application Development*. Indianapolis: Wiley Publishing, Inc., 2010.

[4] W. Lee, *Beginning Android Application Developmnet*. Indianapolis: Wiley Publishing, Inc., 2011.

[5] "iPad Technical specification". Dostopno na: http://www.apple.com/ipad/specs, 2011 [27.9.2011].

[6] "iOS Developer Library." Dostopno na: http://developer.apple.com/library/ios/navigation, 2011 [27.9.2011].

[7] *iOS Application Programming Guide*. Cupertion: Apple Inc, 2010.

[8] "iOS." Dostopno na: http://en.wikipedia.org/wiki/Iphone_os. 23.7.2011 [24.7.2011].

[9] 'Hardware Compatibility" v *Android 2.3 Compatibility Program*. Google, Inc., 2011, str. 14-20.

[10] K. Finkenzeller. *RFID Handbook*. Chichester: John Wiley & Sons, Ltd., 2010, str. 57-59.

[11] P. Cohen. "MacWorld Expo Keynote Live Update." Dostopno na: http://www.macworld.com/article/54764/2007/01/liveupdate.html, 9.1.2007 [27.9.2011]

[12] J.A. Brannan, B. Ward. *iOS SDK Programming: A Beginner's Guide*. New York: McGraw-Hill, 2011.

[13] "iPad user experience guideline." Dostopno na: http://uxmag.com/design/ipad-user-experience-guidelines, 2.2.2010 [27.9.2011].

[14] "Android TabHost Tutorial - Part 1." Dostopno na: http://www.androidpeople.com/android-tabhost-tutorial-part-1, 5.7.2010 [27.9.2011].

[15] "Android Patterns." Dostopno na: http://www.androidpatterns.com/, 2011 [27.9.2011].

[16] "Android Developers." Dostopno na: http://developer.android.com/index.html, 2011 [27.9.2011].

[17] C. Blunt. "Get Started Developing For Android With Eclipse, Reloaded." Dostopno na: http://coding.smashingmagazine.com/2011/03/28/get-started-developing-for-android-with-eclipse-reloaded/, 28.3.2011 [27.9.2011].

[18] "Windows Phone 7 hardware requirements confirmed." Dostopno na: http://www.gsmarena.com/windows_phone_7_hardware_requirements_confirmed_two_wp7_phones_and_an_htc_hd3_rumor-news-1513.php, 16.3.2010 [27.9.2011].

[19] T. Steffes, "Android REST XML implementation." Dostopno na: http://www.smnirven.com/?p=15, 25.8.2009 [27.9.2011].

[20] L.Richardson, S.Ruby *RESTful web services*. Sebastopol: O'Reilly Media, Inc., 2007.

[21] Noya. "Write a simple XML file in the SD card using XmlSerializer." Dostopno na: http://www.anddev.org/write_a_simple_xml_file_in_the_sd_card_using_xmlserializer-t8350.html, 9.10.2010 [27.9.2011].

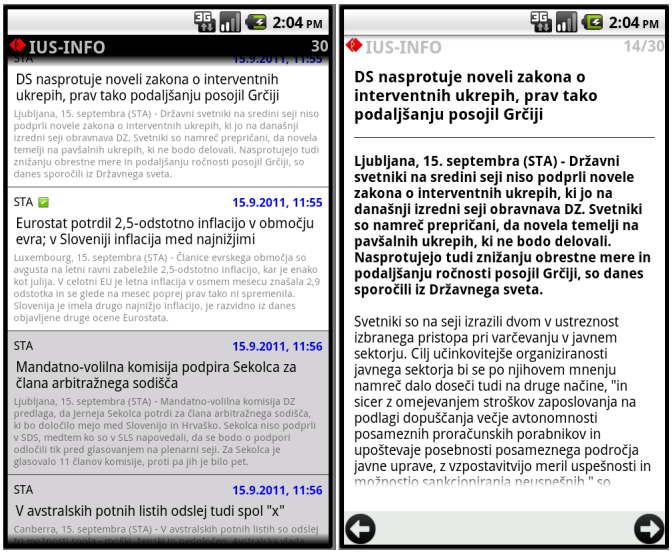
Dodatek A

Navodila za uporabo aplikacije - primer za Androida

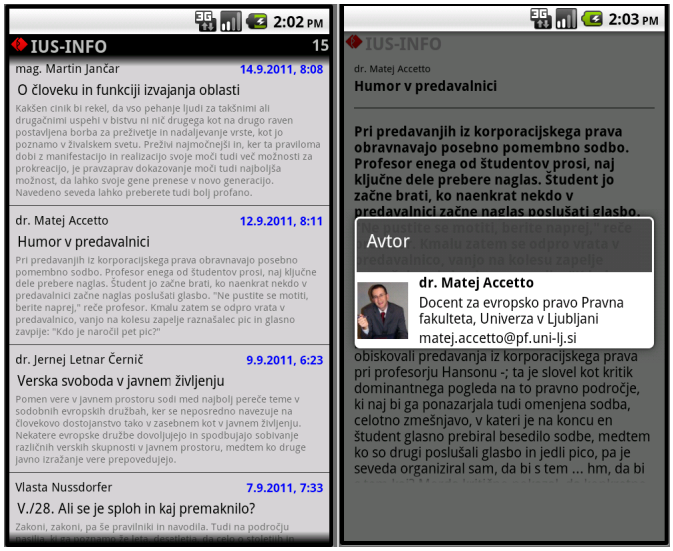
Aplikacijo zaženemo s pritiskom na ikono (slika A.1). Naloži se glavni zaslon, od koder lahko dostopamo do ostalih aktivnosti (slika A.2). Prva možnost je, da kliknemo na ikono Novice, kjer se naložijo novice (slika A.3). Zelena ikona ob naslovu označuje, da je novica že prebrana. Sivo obarvana novica označuje, da novice še nismo prebrali, in da je bila na novo naložena s strežnika. Ob kliku na novico se nam odpre njena vsebina. S pritiskom na puščice na spodnji strani ekrana se lahko premikamo med novicami. Premikamo se lahko tudi s potegom prsta z desne proti levi za naslednjo ali z leve proti desni za prejšnjo novico. Kolumne potekajo po enakem postopku kot novice. Pri kolumni si lahko ogledamo kratke informacije o avtorju (slika A.4). Pri rubriki Koristno imamo na voljo tri zavihke. Prvi zavihek nam omogoča izračun potnih stroškov, drugi zavihek nam posreduje informacije o prometnih prekrških (slika A.5), tretji pa nam naloži najnovejšo kolumno. Rubrika Priljubljeno hrani bližnjice do členov, ki smo jih shranili. Rubrika Zakoni nam omogoča brskanje po zakonih (slika A.6), ob kliku na zakon pa lahko preberemo izbrani člen. Rubrika Izračuni vsebuje tri zavihke. Prvi je izračun o otroškem dodatku, drugi je izračun avtorskega honorarja in zadnji je izračun potnih stroškov (slika A.7).

Slika A.1: Ikona aplikacije

Slika A.2: Glavni zaslon aplikacije za Androida

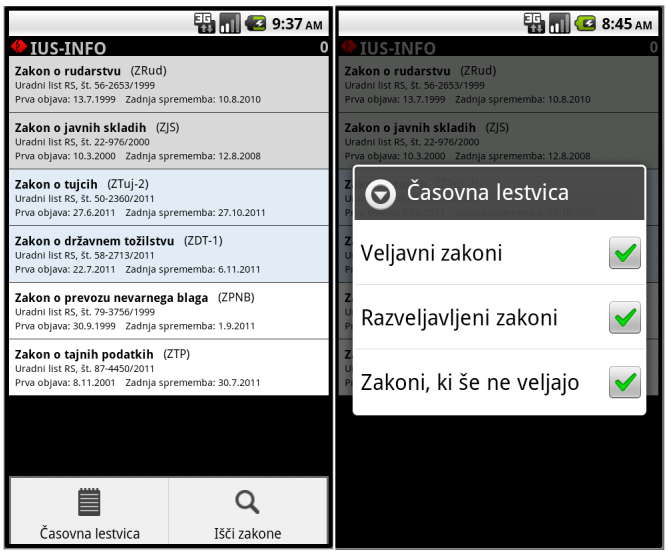


Slika A.3: Zaslona za seznam novic in branje novice

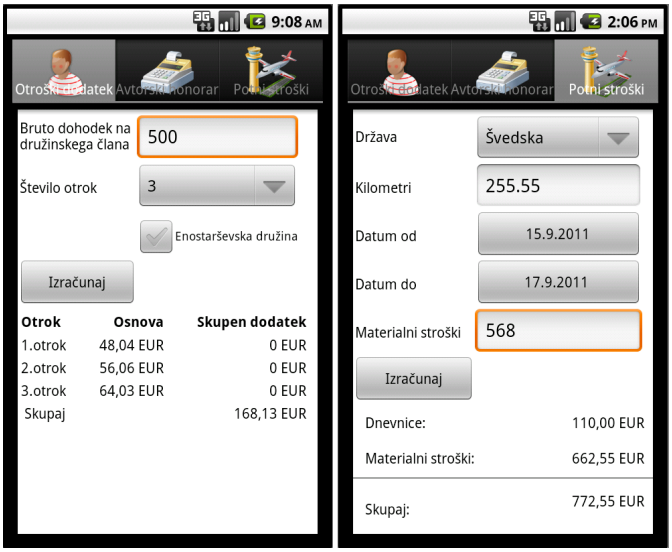


Slika A.4: Seznam kolumn in informacije o avtorju

Slika A.5: Seznam prometnih prekrškov in podrobnosti o izbranem prekršku



Slika A.6: Seznam zakonov in filtriranje zakonov



Slika A.7: Zaslona za izračuna otroškega dodatka in potnih stroškov

Slike

2.1 Mobitel Nexus One 7
2.2 Tablični računalnik iPad. Prva generacija. 10

6.1 Arhitektura IUS-TIME 24
6.2 Podatkovna baza IUS-TIME 30
6.3 Podatkovna baza PortalCMS 32
6.4 Diagram primera uporabe 34
6.5 Življenjski cikel aktivnosti pri aplikaciji za Androida 37
6.6 Sprožitev nove aktivnosti 38
6.7 Primer programskega dodajanja elementov 39
6.8 Urejevalnik uporabniškega vmesnika 40
6.9 Prikaz hiearhije pogledov in odvisnosti parametrov za razporeditev od nadrejenega pogleda 41
6.10 Prikaz različnih vrednosti za lastnost weight, na levi strani ima spodnje okno za tekst vrednost 0, na desni pa 1 43
6.11 Slika začetnega okna IUS-INFO aplikacije 44
6.12 Leva stran slike prikazuje opcijski meni, desna pa podmeni za opcijo Časovna lestvica 51
6.13 Primer pogovornega okna ProgressDialog 52
6.14 Primer lastno izdelanega pogovornega okna 52
6.15 Primer sporočila Toast 53
6.16 Primer sporočila v statusni vrstici 53
6.17 Graditelj gest 55
6.18 Primer življenjskega cikla aplikacije 57
6.19 Primer uporabe pogleda UIProgressView 62
6.20 Primer opozorilnega okna za iOS 62
6.21 Primer okna UIActionSheet 63
6.22 Primer uporabe značk (Application Badges) 63
6.23 Pogled za izbiranje elementov UIPickerView 64

6.24 Primer uporabniškega vmesnika na iPadu 66

7.1 Primer glavnega zaslona na telefonu Nexus One in iPadu 69
7.2 Primer kolumne pri telefonu Nexus One in iPadu 69

A.1 Ikona aplikacije 75
A.2 Glavni zaslon aplikacije za Androida 76
A.3 Zaslona za seznam novic in branje novice 76
A.4 Seznam kolumn in informacije o avtorju 77
A.5 Seznam prometnih prekrškov in podrobnosti o izbranem prekršku 77
A.6 Seznam zakonov in filtriranje zakonov 78
A.7 Zaslona za izračuna otroškega dodatka in potnih stroškov 78

Tabele

2.1 Tehnične značilnosti mobitela Nexus One 9
2.2 Tehnične značilnosti tabličnega računalnika iPad. 11

3.1 Številke in imena verzij Androida 14
3.2 Verzije operacijskega sistema iOS 16

6.1 HTTP metode za spletno storitev RESTful 25
6.2 Tipi virov 45
6.3 Pravila za podajanje kvalifikatorjev 48
6.4 Podrazredi razreda UIView 60