Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Rok Drinovec

Razvoj mobilnega odjemalca za platformo Occapi

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Dejan Lavbič

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

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

Št. naloge: 00107 / 2013
Datum: 15.4.2013

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

Kandidat: **ROK DRINOVEC**

Naslov: **RAZVOJ MOBILNEGA ODJEMALCA ZA PLATFORMO OCCAPI**
DEVELOPMENT OF MOBILE CLIENT FOR OCCAPI PLATFORM

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

Platforma Occapi, ki se uporablja za kompleksno analizo dogodkov v realnem času ima na voljo številne funkcionalnosti, ki so na voljo preko različnih izhodnih kanalov. Med drugim platforma omogoča mobilnim napravam dostop preko API-ja in sicer storitve prijave uporabnika, pridobivanje podatkov o izjemnih dogodkih, pridobivanje podatkov o izjemnih dogodkih opremljene z geolokacijskimi podatki, pridobivanje podatkov za prikaz grafov itd. Razvoj medplatformske mobilne aplikacije je pogosto zahtevno opravilo, omejeno s številnimi vidiki. V okviru diplomske naloge naj študent razvije različne mobilne odjemalce z različnimi pristopi (domorodni in HTML5) za platformo Occapi. Študent naj razvoj po posameznih pristopih in platformah ovrednoti (tehnični in finančni vidik) ter poda kritično oceno uporabljenih pristopov.

Mentor:

doc. dr. Dejan Lavbič

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Rok Drinovec, z vpisno številko **63080089**, sem avtor diplomskega dela z naslovom:

Razvoj mobilnega odjemalca za platformo Occapi

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Dejana Lavbiča;
- 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, 15. marca 2014 Podpis avtorja:

Zahvaljujem se mentorju doc. dr. Dejanu Lavbiču za pomoč pri izdelavi diplomskega dela. Zahvaljujem se tudi Mihu Radeju in Milanu Pevcu, ki sta poskrbela za kar se da nemoteno delovanje storitev na strežniku. Najbolj pa se zahvaljujem družini in prijateljem, ki so me podpirali in spodbujali vsa ta leta študija.

Družini.

Seznam uporabljenih kratic

- **ADB** (angl. Android Debug Bridge) - orodje za razhroščevanje naprave Android,
- **ADT** (angl. Android Development Tools) - razvojna orodja za Android,
- **API** (angl. Application Programming Interface) - programski vmesnik, ki opisuje, kako naj posamezne programske komponente komunicirajo med seboj,
- **CSS** (angl. Cascading Style Sheets) - kaskadne stilske podloge za vizualno predstavitev spletnih strani,
- **DPI** (angl. Dots Per Inch) - število slikovnih pik na inčo, mera gostote pik,
- **DOM** (angl. Document Obect Model) - specificira prezentacijo in interakcijo objektov v dokumentih HTML,
- **HTML** (angl. Hyper Text Markup Language) - označevalni jezik za izdelavo spletnih strani,
- **IDE** (angl. Integrated Development Environment) - integrirano programsko okolje za razvoj aplikacij, ki omogoča pisanje kode, prevajanje in razhroščevanje,
- **JSON** (angl. JavaScript Object Notation) - človeku razumljiv tekstovni format za opis podatkov, alternativa XML-ju,

- **KPI** (angl. Key Performance Indicator) - ključni kazalnik uspešnosti, vrsta merjenja uspešnosti,
- **MVC** (angl. Model-View-Controller) - model-pogled-kontroler, način predstavitve uporabniškega vmesnika,
- **PHP** (angl. PHP Hypertext Preprocessor) - skriptni programski jezik za razvoj dinamičnih spletnih strani,
- **SDK** (angl. Software Development Kit) - razvojni programski paket, ki omogoča razvoj za določene programske pakete, operacijske sisteme, konzole ali druge platforme,
- **USB** (angl. Universal Serial Bus) - standard kablov, konektorjev in protokolov za prenos podatkov med računalnikom in elektronskimi napravami,
- **XML** (angl. Extensible Markup Language) - označevalni jezik za opis podatkov v obliki, ki je razumljiva človeku in računalniku.

Kazalo

Povzetek

Abstract

1 Uvod **1**

2 Uporabljene tehnologije **3**

2.1 Mobilne naprave 3

2.2 JavaScript 4

2.3 Sencha Touch 5

2.4 Sencha Architect 5

2.5 Android SDK 7

2.6 AChartEngine 8

2.7 VMware Player 9

2.8 Xcode 9

2.9 ECSlidingViewController 11

2.10 ios-linechart 11

2.11 Occapi 11

3 Razvoj mobilne aplikacije **13**

3.1 Mobilna aplikacija 13

3.2 Realizacija zahtev 15

3.2.1 API 18

3.2.2 Sencha Touch 19

3.2.3 Android 22
3.2.4 iOS 25
3.3 Primerjava kode 27
3.3.1 Sencha Touch 27
3.3.2 Android 28
3.3.3 iOS 29
3.4 Razhroščevanje 31
3.5 Posodabljanje aplikacije 31
3.6 Podpiranje dodatnih naprav 33
3.7 Grafično oblikovanje aplikacije 33
3.7.1 Simboli in ikone 33

4 Primerjava aplikacij **35**
4.1 Delovanje aplikacije 35
4.2 Primerjava razvoja in sredstev 37
4.2.1 Sencha Touch aplikacija 37
4.2.2 Domorodni aplikaciji 38
4.3 Končna ocena 39

5 Sklepne ugotovitve **41**

Literatura **43**

Kazalo slik in tabel

2.1 Diagram prehodov med zaslonskimi maskami v programu Xcode 10
2.2 Spletni vmesnik platforme Occapi 12

3.1 Mobilna naprava, posrednik in strežnik Occapi 14
3.2 Domorodni Android meni 16
3.3 Sencha Touch meni 16
3.4 Domorodni iOS meni 16
3.5 Graf izdelan z AChartEngine 17
3.6 Sencha Touch graf 17
3.7 Graf izdelan z ios-linechart 17
3.8 Android nastavitve 18
3.9 Sencha Touch nastavitve 18
3.10 iOS nastavitve 18
3.11 Google Maps v Sencha Touch aplikaciji 32
3.12 Google Street View v Sencha Touch aplikaciji 32

4.1 Primerjave hitrosti zagona aplikacije 36
4.2 Cene uporabljenih programov in knjižnic pri razvoju z uporabo standarda HTML5 38
4.3 Cene uporabljenih programov in knjižnic pri razvoju domorodnih aplikacij 38
4.4 Ocene bistvenih faktorjev pri aplikaciji in končna ocena 40

Kazalo programskih kod

2.1 Nepravilna primerjava spemenljivk v JavaScriptu 4
3.1 Pretvorba časovnega žiga v besedilo v Sencha Touchu 28
3.2 Zahteva za prijavo in preverjanje prisotnosti žetona v Sencha Touchu 28
3.3 Pretvorba časovnega žiga v besedilo v Javi 29
3.4 Zahteva za prijavo in preverjanje prisotnosti žetona v Javi . . 29
3.5 Pretvorba časovnega žiga v besedilo v objektnem C-ju 30
3.6 Zahteva za prijavo in preverjanje prisotnosti žetona v objektnem C-ju 30

Povzetek

Pred pričetkom izdelave mobilne aplikacije se moramo odločiti, kakšen pristop bomo uporabili. Vedno več je aplikacij, ki so izdelane z uporabo spletnih tehnologij. Za razliko od domorodnih aplikacij nimajo neposrednega dostopa do sistemskih funkcij in delujejo nekoliko počasneje, ampak delujejo na večini operacijskih sistemov današnjih mobilnih naprav. Skrajša se razvojni čas, prav tako pa nam ni potrebno poznati programskih jezikov, v katerih so napisane domorodne aplikacije.

V diplomskem delu smo izvedli primerjavo med pristopoma razvoja z izdelavo aplikacije za pregledovanje podatkov platforme Occapi, ki zbira podatke iz množice naprav in interneta ter jih predstavi na uporabniku prijazen način v obliki ključnih kazalcev uspešnosti. Podatki so prikazani s tekstovnimi seznami in črtnimi grafi.

Razvili smo tri aplikacije. Uporabili smo razvojno okolje Sencha Architect za razvoj HTML5 aplikacije. Za Android aplikacijo smo uporabili Eclipse IDE skupaj z Android ADT. Za iOS aplikacijo pa smo uporabili XCode v virtualiziranem okolju. Podrobno smo opisali razvoj v vseh razvojnih okoljih. Podali smo zahteve in njihove realizacije. Na koncu smo vse tri aplikacije tudi testirali in podali oceno obeh pristopov razvoja.

Ključne besede:
platforma Occapi, HTML5, Sencha Touch, Android, iOS, mobilna aplikacija.

Abstract

Before developing mobile application we have to decide which approach to use. More and more applications are made by using web technologies. Unlike native applications they do not have direct access to system functions and they work a bit slower but they work on most operating systems of today's mobile devices. Development time is shorter and we do not need to know programming languages in which native applications are written.

In thesis we made comparison between two approaches of developing application for viewing Occapi platform data, which collects data from many devices and Internet and presents them in a user-friendly manner in form of key performance indicators. Data is shown with text lists and line graphs.

We developed three applications. We used development enviroment Sencha Touch for developing HTML5 application. For Android application we used Eclipse IDE with Android ADT. For iOS application we used XCode in virtualized enviroment. We described in detail the development in all of development enviroments. We supplied the requirements and their implementation. At the end, we tested all three applications and gave an assessment of the two approaches of development.

Keywords:
Occapi platform, HTML5, Sencha Touch, Android, iOS, mobile aplication.

Poglavje 1

Uvod

Živimo v obdobju poplave mobilnih naprav. V zadnjih nekaj letih smo priča ogromnega porasta števila naprav, kot so pametni telefoni in tablice. Z večanjem števila le-teh in višanjem njihovih zmogljivosti, pa se povečuje tudi število aplikacij zanje.

Uporabnik si vedno želi, da bi obstajala aplikacija, ki bi ustregla njegovim zahtevam. Od nje želi čim bolj zanesljivo in čim hitrejše delovanje, poleg tega pa je pozoren tudi na ceno aplikacije. Želi dobiti najboljše za svoj denar, še raje pa vidi, da je aplikacija brezplačna.

Kot razvijalci pa se moramo vprašati, ali se nam splača izdelati aplikacijo za potrebe uporabnika. Razvijalci morajo upoštevati stroške razvoja, kamor spadajo razvojni čas in cene licenc, če uporablja orodja za razvoj, za katera je potrebno plačati licenčnino. Pri razvoju je potrebno biti tudi pozoren na zanesljivost, zmogljivost in seveda na prenosljivost aplikacije. Če razvijamo aplikacijo za širši trg, je dobro, da pokrijemo čim več različnih mobilnih naprav, le-te pa se zelo razlikujejo. Tudi če razvijamo aplikacijo za stranko, bo lahko ta želela, da deluje na različnih mobilnih napravah.

Zelo pogosta mobilna naprava v današnjih časih poganja operacijski sistem Android. Aplikacije, ki tečejo na njem, so napisane v Javi. Zelo priljubljen operacijski sistem za mobilne aplikacije je tudi iOS (iPhone, iPad ...), le da so aplikacije zanj napisane v objektnem C-ju (angl. Objective C).

Dandanes ne smemo spregledati tudi Windows mobilnih naprav, na trgu pa so še BlackBerry mobilniki.

Če bi hoteli izdelati aplikacijo, ki bi delovala na vseh teh napravah, bi imeli zelo veliko dela, saj bi morali za vsako napravo razviti aplikacijo v drugem jeziku.

Mobilne naprave so iz dneva v dan zmogljivejše. Skoraj vsak od nas s seboj nosi napravo, ki je skoraj toliko zmogljiva kot stacionarni računalniki ali prenosniki. Tako zmogljive naprave tudi odlično poganjajo spletne brskalnike, za katere pa več ali manj veljajo standardi, ne glede na operacijski sistem. Če pospošimo, lahko rečemo, da razvijemo eno aplikacijo in ta potem deluje na vseh (na večini) mobilnih napravah.

Kdaj se odločiti za razvoj aplikacije v HTML5 (angl. Hyper Text Markup Language) standardu in kdaj se poslužiti razvoja v domorodnem (angl. native) jeziku? Preden se lotimo razvoja aplikacije je potrebno močno premisliti. S podobno težavo so se na drugih problemskih domenah ukvarjali tudi sorodna dela [17, 18, 19], ki prav tako obravnavajo primerjavo klasičnega razvoja aplikacije z razvojem v standardu HTML5.

V diplomskem delu smo predstavili razvoj aplikacije v več možnih standardih in za različne mobilne naprave. Razvili smo tri aplikacije, eno z uporabo standarda HTML5, eno domorodno aplikacijo za Android in eno domorodno aplikacijo za iOS. V drugem poglavju so opisane tehnologije, ki smo jih uporabili v času razvoja. Tretje poglavje predstavlja opredelitev mobilne aplikacije in realizacijo njenih zahtev z uporabo različnih pristopov in razvojnih okolij, četrto poglavje pa opisuje končno primerjavo vsakega izmed pristopov, delovanje posamezne aplikacije in končno oceno.

Poglavje 2

Uporabljene tehnologije

V nadaljevanju so predstavljena orodja in tehnologije, ki smo jih uporabili pri izdelavi mobilnih aplikacij.

2.1 Mobilne naprave

Za čim bolj podrobno primerjavo načinov razvoja je zaželeno imeti različne naprave. Naprave pa naj se ne bi razlikovale samo po tehničnih specifikacijah ampak tudi po operacijskem sistemu, ki ga poganjajo.

Pri razvoju in testiranju aplikacij smo uporabili Samsung Galaxy Nexus, Asus Nexus 7 (2012) in iPod Touch 5. Prvi dve napravi imata nameščen operacijski sistem Android 4.4 KitKat, zadnji pa iOS 7. V času razvoja so nam bile naprave v pomoč pri testiranju, saj aplikacija v emuliranem okolju deluje nekoliko drugače, prav tako pa se klikanje z miško težko primerja s pritiskanjem in vlečenjem po zaslonu. Napravi z Androidovim operacijskim sistemom sta bili nepogrešljivi pri testiranju in razhroščevanju, saj aplikacija v Androidovemu emulatorju deluje bistveno počasneje. Bistveno vlogo pa so igrale pri končnem preizkušanju hitrosti in samem občutku delovanja.

2.2 JavaScript

Tako kot PHP (angl. PHP Hypertext Preprocessor) je JavaScript skriptni jezik za razvoj dinamičnih spletnih strani. Glavna razlika med njima je, da se PHP program izvaja na strežniku, JavaScript program pa se izvaja v uporabnikovemu brskalniku. V vseh sodobnih brskalnikih lahko odpremo okno za razvijalce, kamor že lahko pišemo kodo JavaScript.

JavaScript je netipiziran jezik, kar pomeni, da nam kot programerjem ni potrebno določiti tipa vrednosti, ki jo bo hranila spremenljivka. To je pozitivna stvar, ker nam ni potrebno pri vsakem deklariranju spremenljivke določati tipa, prav tako lahko eno spremenljivko uporabimo za shranjevanje več tipov. Ni potrebno ustvarjati več funkcij, ki se razlikujejo samo v tipu parametrov, ampak v funkciji preverimo tip parametra z uporabo `typeof`. Negativna stvar netipiziranega jezika pa je oteženo razhroščevanje, saj se nam lahko kar hitro zgodi, da na primer po nesreči primerjamo objekt z besedilnim nizom, kar je seveda napačno in bo vedno neresnično.

```
var string = "Besedilo";
var object = { value: "Besedilo" };
if(string == object) {
    alert("Nemogoče");
}
```

Programska koda 2.1: Nepravilna primerjava spemenljivk v JavaScriptu

JavaScript dopušča precej svobode, do vseh funkcij in vrednosti lahko brez težav dostopamo od kjerkoli, prav tako jih lahko spreminjamo, privatne spremenljivke ne obstajajo. To lahko privede do tega, da kakšno spremenljivko ali funkcijo po nesreči prepišemo z drugo, če nismo dovolj pozorni.

2.3 Sencha Touch

Sencha Touch [2], v nadaljevanju tudi ST, je knjižnica za JavaScript, ki omogoča enostavno izdelavo mobilnih spletnih aplikacij, ki izgledajo kot domorodne aplikacije. Uporablja standarde HTML5, CSS (angl. Cascading Style Sheets) in JavaScript. Poenostavi interakcijo z DOM (angl. Document Obect Model) objekti, vsebuje veliko pogosto uporabljenih funkcij in elementov. Podpira mobilne naprave z operacijskim sistemi:

- BlackBerry,
- Android,
- iOS,
- Windows.

Ažurni podatki o uradno podprtih operacijskih sistemih in napravah so objavljeni na spletni strani [3]. Knjižnica podpira večino sodobnih mobilnih brskalnikov, prav tako pa lahko aplikacije uporabljamo z brskalniki na namiznih računalnikih, kar omogoča hitrejši razvoj in razhroščevanje.

2.4 Sencha Architect

Sencha Architect [4] je IDE (angl. Integrated Development Environment) za izdelavo aplikacij Sencha Touch, ki so namenjene mobilnim napravam, in Ext JS, ki so namenjene namiznemu računalniku. Vsebuje orodja za izdelavo grafičnega vmesnika, urejevalnik besedila z dokončevanjem kode in pakiranje aplikacije za operacijska sistema Android in iOS. Sencha Architect podpira MVC (angl. Model-View-Controller) pristop razvoja aplikacije. Vsaka aplikacija je razdeljena na več skupin:

- pogledi,
- kontrolerji,

- shrambe,
- modeli,
- ostali viri (predloge CSS, datoteke s kodo, teme).

Pogledi vsebujejo vse vizualne komponente aplikacije, vsaj statične, dinamične pa lahko ustvarimo tudi s kodo. Urejanje pogledov je precej preprosto, uporablja se pristop "povleci in spusti". Začetni pogled aplikacije izberemo z nastavljanjem lastnosti `initialView`. Za lažji dostop do komponent lahko določimo njihovi lastnosti `itemId` in `id`. Pri uporabi `id` moramo biti pozorni, saj se v DOM-u ne sme podvojiti. Vsaki komponenti lahko določimo tudi kodo, ki naj se izvede ob določenih dogodkih, kot so na primer inicializacija, prikaz, dotik in sprememba vrednosti, vendar raje celotno kodo zapišemo v kontrolerje.

S kontrolerji dosežemo ločitev pogledov in programske kode. Kontrolerji vsebujejo akcije, funkcije in reference na komponente. Akcije se izvedejo ob dogodku neke komponente ali večih komponent. Če selektor akcije nastavimo na: `#formId button`, lahko izdelamo formo z več gumbi, ki vsi sprožijo eno akcijo.

Shrambe služijo za enostavnejše nalaganje in predstavitev niza podatkov. Vsaki shrambi je potrebno določiti model, v katerem so določena vsebovana polja skupaj z njihovimi tipi. Če želimo podatke v shrambi spreminjati in jih poslati nazaj na strežnik, moramo modelu nastaviti lastnost `idProperty`, ki mora biti unikatni identifikator. Shramba je množica `record`-ov, katerih vrednosti polj dobimo z `record.get("imePolja")` in nastavimo z `record.set("imePolja", vrednost)`. Shramba zazna spremembe v `record-u`, prav tako zazna dodajanje ali brisanje novega vnosa. Spremembe v shrambi sinhroniziramo s strežnikom z uporabo ukaza `store.sync()`. Shrambe so nadvse uporabne za sezname. Vsebujejo funkcije za filtriranje, iskanje in grupiranje. Vsaka od teh operacij pa se takoj odraža tudi v seznamu.

V novi verziji 3.0 je podprto tudi spreminjanje teme. Podprte teme:

- Sencha,

- Cupertino (za iOS 7),
- Cupertino Classic (za iOS 6 in nižje),
- Mountain View (za Android),
- BlackBerry 10,
- Windows 8.

Aplikacija ob menjavi takoj dobi obliko domorodne aplikacije izbranega operacijskega sistema. Barve in oblike komponent lahko v vsaki temi tudi ročno spreminjamo. Celoten izgled aplikacije tako lahko bistveno spremenimo že samo z določanjem dveh barv, to sta `$active-color` in `$base-color`.

Izdelava aplikacije je z uporabo Sencha Architecta enostavnejša kot izdelava z ročnim pisanjem kode. Program je dostopen za uporabnike operacijskih sistemov Windows, Mac in Linux.

Na voljo imamo 30-dnevno preizkusno različico, s katero smo tudi izdelali aplikacijo. Za nadaljnjo uporabo pa je potrebno za Sencha Architect odšteti 399.00 ameriških dolarjev. Aktualne cene in ponudbe so dostopne na trgovini Sencha Architect [5].

2.5 Android SDK

Android SDK [6] (angl. Software Development Kit) je skupek vseh potrebnih razvojnih orodij za izdelavo Android aplikacije. Paket vključuje:

- Eclipse + vtičnik ADT (angl. Android Development Tools),
- platformna orodja za Android,
- najnovejšo različico Androida,
- najnovejšo različico sistemske slike Androida za emulator.

Android SKD uporablja programski jezik Java. Pri razvoju ne potrebujemo fizične naprave, ampak lahko uporabimo priložen emulator. Kar hitro pa smo ugotovili, da deluje emulator zelo počasi in smo za testiranje in razhroščevanje uporabljali fizično napravo, priklopljeno preko kabla USB (angl. Universal Serial Bus).

Vsak Android projekt je razdeljen na mape:

- src - vsebuje programsko kodo Java;
- libs - kamor vstavimo dodatne knjižnice;
- res - kamor vstavimo različne vire, po večini so to datoteke `.xml`; vsebuje mape:
  - anim - vsebuje opis animacij;
  - drawable - vsebuje slike, stile seznamov, gumbov in ostalih komponent;
  - layout - kjer so določeni pogledi, prav tako pa izgled celic seznama;
  - menu - vsebujejo elemente spustnih seznamov;
  - values - tu se določijo statični napisi, ki se uporabijo v pogledih, barve, teme;
  - xml - mapa za splošne datoteke `.xml`, vanjo smo shranili definicijo nastavitev.

Android SDK deluje na operacijskih sistemih Windows, Mac in Linux.

2.6 AChartEngine

Knjižnica AChartEngine [7] omogoča risanje različnih grafov, kar ni podprto s strani Android SDK, razen če bi to implementirali sami z risanjem preprostih likov. Knjižnica omogoča preprosto predstavitev grafikonov v obliki točk, črt, stolpcev, tortnih diagramov in podobno. Omogoča tudi precej oblikovnih nastavitev.

2.7 VMware Player

Ker si ne lastimo strojne opreme, na kateri bi deloval OS X, smo morali ta operacijski sistem virtualizirati. V ta namen smo uporabili program VMware Player [8]. Program ustvari navidezno okolje, ki deluje kot ločena strojna oprema. Omejimo ji velikost diska, pomnilnika in procesorsko moč.

2.8 Xcode

Xcode je prav tako IDE, v katerem lahko razvijamo aplikacije za iOS in OS X. V operacijskih sistemih OS X se Xcode naloži preko programa Appstore.

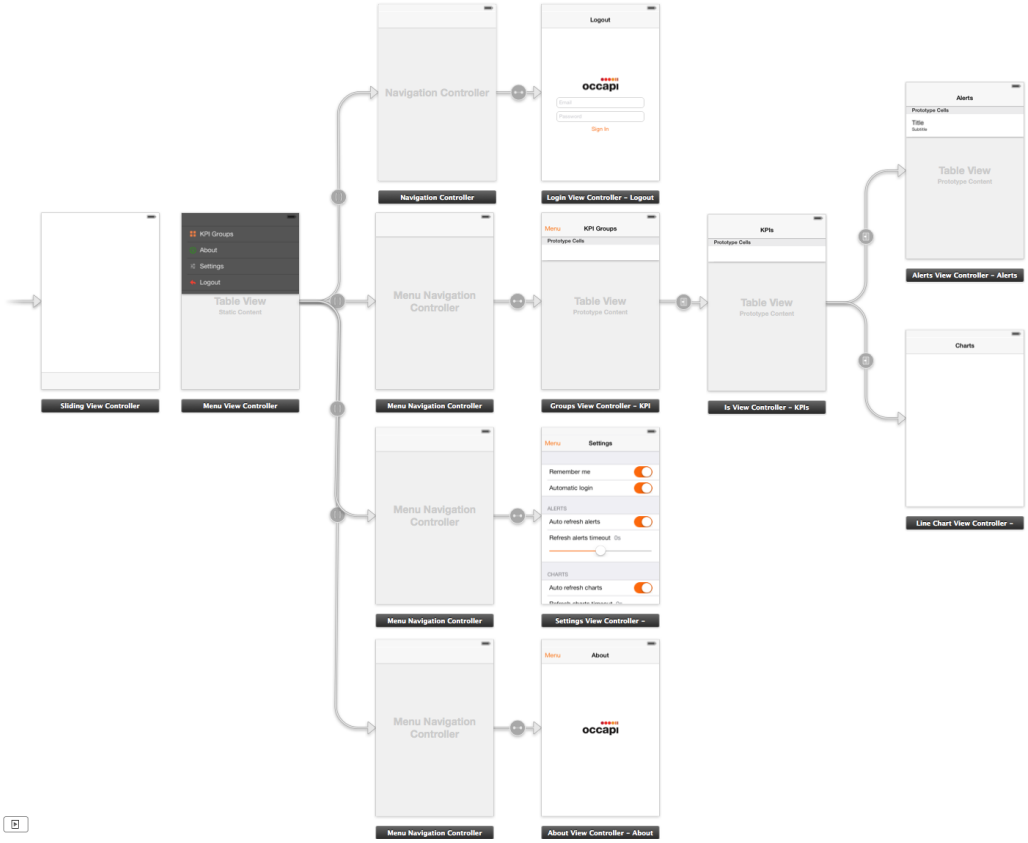
Aplikacije se razvija v objektnem C-ju. Za razliko od Sencha Architecta in Android SDK-ja, kjer je viden en pogled naenkrat, imamo tu eno veliko površino imenovano "Storyboard", na kateri so razvidni vsi pogledi. Le-ta je prikazana na sliki 2.1. To se nam zdi bolj uporabno, saj je takoj razvidno, kako se pogledi povezujejo med seboj. Za udoben pregled pa je potrebno imeti precej visoko resolucijo zaslona, saj se pri nizki ne vidi kaj dosti. Tudi samo logično povezovanje pogledov je zelo intuitivno. Če hočemo ob dotiku neke celice v seznamu odpreti nov pogled, samo z miško povlečemo iz celice na nov pogled. Če hočemo ob prehodu kaj postoriti, kodo zapišemo v funkcijo:

```
- (void) prepareForSegue:(UIStoryboardSegue *)segue
        sender:(id)sender {
    /* ... */
}
```

Uporabna zadeva, ki jo lahko naredimo pri urejanju pogledov, so tudi statični seznami, ki grafično niso podprti ne v Sencha Architectu ne v Android SDK-ju.

Vsakemu pogledu lahko določimo prirejen kontroler. Koda je zapisana v dveh datotekah, v eni zaglavni datoteki s končnico `.h`, ki vsebuje deklaracije in mikro definicije, ter v datoteki s končnico `.m`, ki vsebuje izvorno kodo.

Pri preizkušanju in razhroščevanju aplikacije smo uporabljali emulator, nad katerim smo bili zelo pozitivno presenečeni, saj je deloval zelo hitro, kot bi bila naprava fizična. Za razliko od Sencha Architecta in Android SKD-ja pa Xcode deluje samo na operacijskem sistemu OS X. Program je, kljub temu da smo OS X virtualizirali, deloval presenetljivo tekoče.



Slika 2.1: Diagram prehodov med zaslonskimi maskami v programu Xcode

2.9 ECSlidingViewController

Na iOS-u drsni meni ni podprt s strani sistema. V ta namen smo uporabili zunanjo odprtokodno knjižnico ECSlidingViewController [14], katerega smo malce dodelali. Meni se od ostalih dveh, pri Sencha Touch in aplikaciji za Android, razlikuje po tem, da se za prikaz odmakne glavni pogled, namesto da bi se meni pojavil čez glavni pogled.

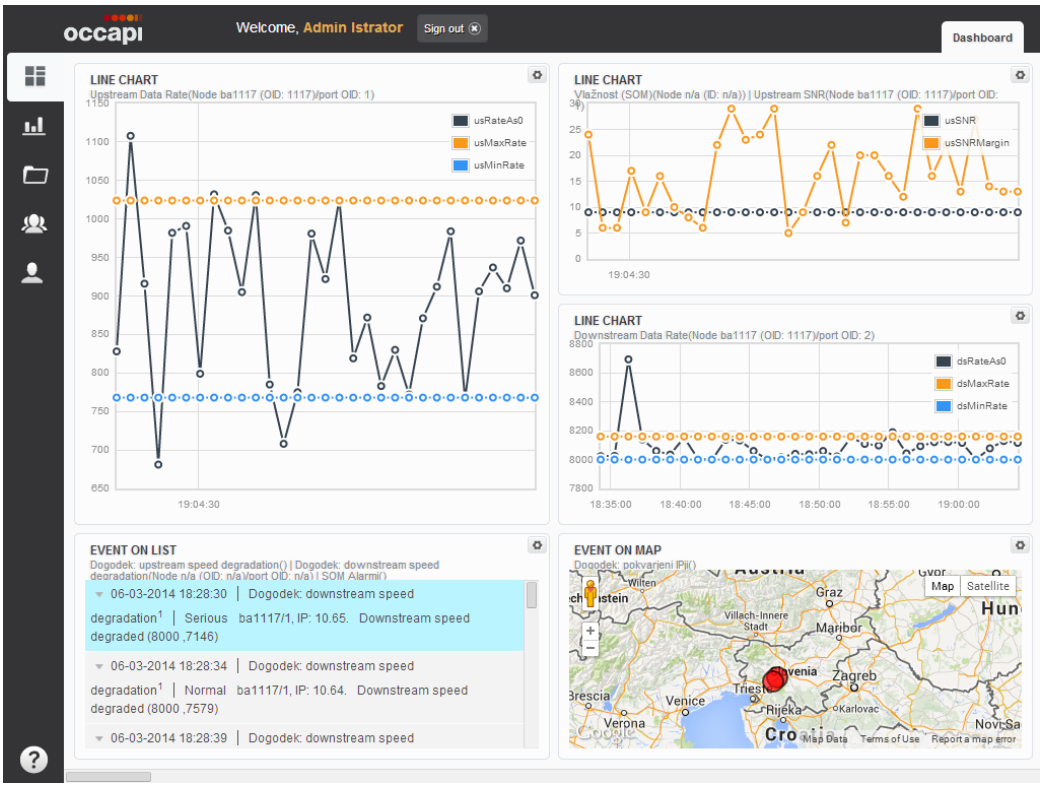
2.10 ios-linechart

Tako kot pri Androidu, smo tudi pri iOS-u uporabili knjižnico za izdelavo grafov, ker to sistemsko ni podprto. Za razliko od AChartEngine, ios-linechart podpira ustvarjanje samo črtnih grafov. Izgled grafa pa se skoraj popolnoma ujema s tistim pri spletnem vmesniku Occapi.

2.11 Occapi

Platforma Occapi [1] na najnižjem nivoju zbira podatke iz množice naprav kot so senzorji, omrežne naprave, merilniki, terminalna oprema in druge naprave. Te podatke obogati s podatki s spleta in drugih baz ter z uporabo naprednih analiz na podlagi strojnega učenja izboljša uporabno vrednost podatkov. Rezultate predstavi s KPI-ji (angl. Key Performance Indicator) na uporabniku razumljiv način v obliki vizualnih podatkov kot so črtni grafikoni in seznami.

Spletna aplikacija platforme Occapi je bila že razvita. Glavni pogled je viden na sliki 2.2. To smo vzeli za izhodiščni izgled naše aplikacije.



Slika 2.2: Spletni vmesnik platforme Occapi

Poglavje 3

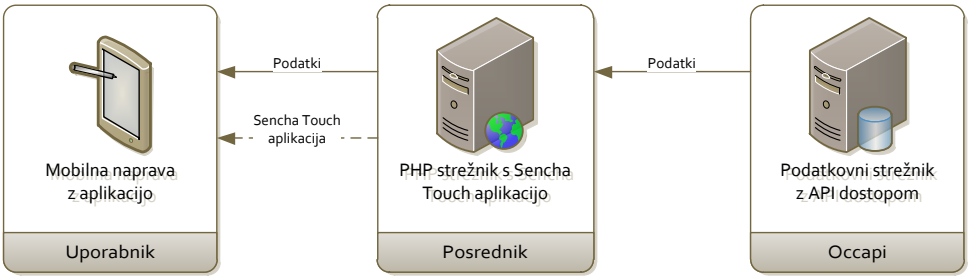
Razvoj mobilne aplikacije

To poglavje opisuje zastavitev funkcionalnosti aplikacije, podroben opis postopkov, težav in rešitev v posameznem IDE in primerjavo med njimi.

3.1 Mobilna aplikacija

Za primerjavo različnih pristopov razvoja mobilne aplikacije smo morali najprej določiti, kakšno aplikacijo bomo izdelali. Odločili smo se, da bomo izdelali mobilno aplikacijo za spremljanje podatkov platforme Occapi. Spletni vmesnik za pregledovanje v realnem času že obstaja, a ni uporaben na mobilnih napravah zaradi prevelike zahtevnosti in postavitve pogledov. S tem namenom smo izdelali aplikacijo, ki omogoča vse bistvene funkcionalnosti kot jih ima spletni vmesnik, le da so le-te dostopne na mobilnih napravah.

Podatki, ki jih potrebuje aplikacija za delovanje, so dostopni preko API (angl. Application Programming Interface). Ker smo izdelali tudi HTML5 aplikacijo, smo se odločili, da bodo datoteke zanjo na spletnemu strežniku PHP in ne na napravi sami. Tako je mogoče enostavnejše popravljanje aplikacije brez ponovnega nalaganja na napravo. Povezava med komponentami je vidna na sliki 3.1.



Slika 3.1: Mobilna naprava, posrednik in strežnik Occapi

Aplikacija je morala podpirati:

- prijavo v sistem,
- prikazovanje KPI-jev v obliki tekstovnih seznamov in črnih grafikonov,
- avtomatsko osveževanje KPI-jev,
- uporabniške nastavitve.

Zaradi manjšega zaslona in nižjih zmogljivosti mobilnih naprav v primerjavi z namiznimi računalniki smo takoj določili omejitev za prikaz enega KPI-ja naenkrat, to je en grafikon ali tekstovni seznam.

Danes poznamo več načinov razvoja:

- domorodni, kjer aplikacija deluje na operacijskem sistemu, za katerega je bila razvita;
- HTML5, kjer aplikacija deluje v večini današnjih brskalnikov;
- hibridni, v katerem domorodna aplikacija z uporabo komponente brskalnika prikaže spletno vsebino.

Ker smo primerjali različne načine razvoja, smo morali pravzaprav razviti več aplikacij. Izdelali smo tri aplikacije. Najprej smo razvili aplikacijo z uporabo

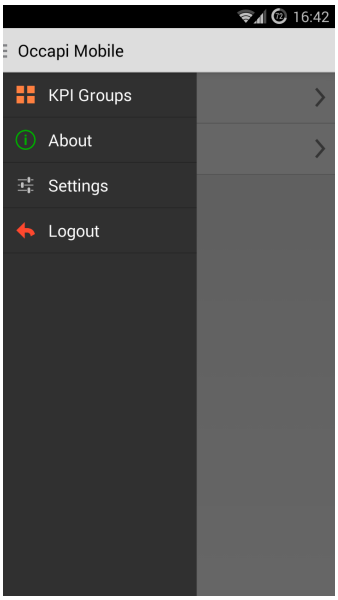
standarda HTML5. Pri tem smo uporabili IDE Sencha Architect. Kasneje smo izdelali domorodno aplikacijo za Android in nazadnje še domorodno aplikacijo za iOS. Sencha Architect nam ponuja pakiranje v domorodni ovoj in tako omogoča dostop do raznih funkcionalnosti operacijskega sistema, ki samemu brskalniku niso dostopne. Teh funkcionalnosti v našem primeru nismo uporabljali in zato tega načina nismo obravnavali kot hibridni. V iOS-ovemu Safariju pa je mogoče ustvariti bližnjico aplikacije na namizje, kjer se ob zagonu naloži spletna aplikacija brez Safarijevih orodnih vrstic, tako da pakiranje aplikacije ni bilo potrebno.

3.2 Realizacija zahtev

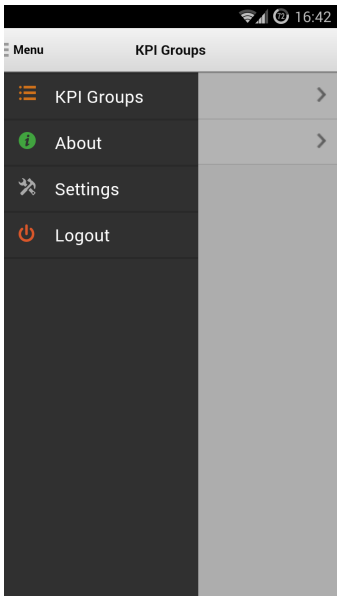
Zastavili smo principe delovanja in izgleda aplikacije, ki pa se v času razvoja niso bistveno spreminjali. Želeli smo, da bi bil izgled aplikacije podoben spletnemu vmesniku za platformo Occapi, prav tako pa bi aplikacija v vseh načinih razvoja delovala in izgledala podobno, pri tem pa bi uporabili čim več sistemskih komponent.

Za prvi pogled smo izdelali prijavno masko z logotipom, vnosnimi polji in gumbom za prijavo. Privzeto se shrani samo elektronska pošta prijavljenega uporabnika, v nastavitvah pa se lahko nastavi tudi shranjevanje gesla in avtomatsko prijavljanje.

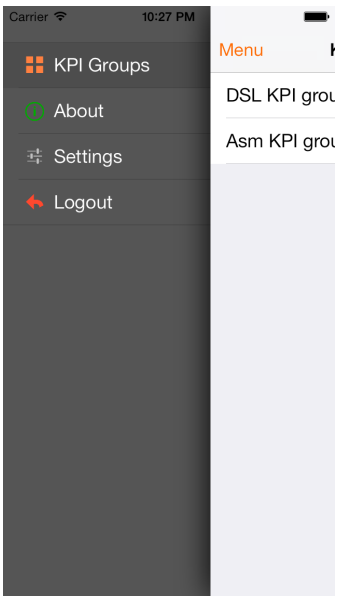
Pri uspešni prijavi se pojavi pogled s seznamom vseh skupin KPI. V tem pogledu smo na levi strani izdelali drsni meni, ki je poznan uporabnikom Androida in iOS-a. Izgled menija je viden na slikah 3.2, 3.3 in 3.4. Meni se lahko odpre z dotikom tipke na levi strani orodne vrstice ali s potegom z leve proti desni.



Slika 3.2: Domorodni Android meni



Slika 3.3: Sencha Touch meni



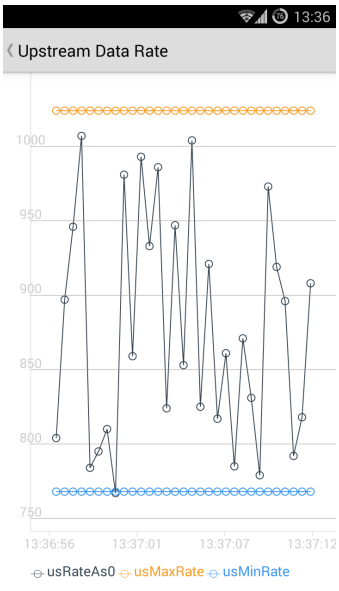
Slika 3.4: Domorodni iOS meni

Z izbiro skupine KPI preidemo v pogled s seznamom KPI-jev. Ti imajo na desni strani ikono, ki pove, ali je KPI predstavljen kot seznam ali črtni graf.

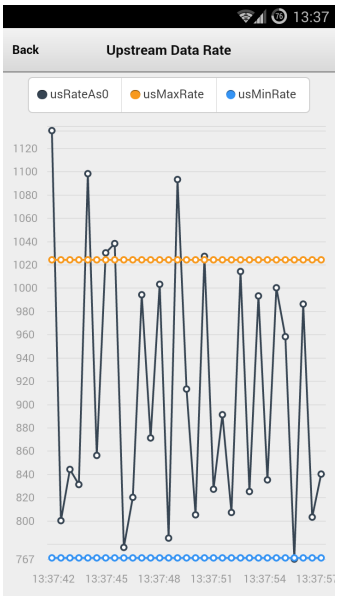
Z izbiro prvega se pomaknemo na tekstovni seznam, iz katerega so razvidni datum in čas vnosa, sporočilo in pomembnost, ki je določena z barvo. Rdeča barva pomeni, da je sporočilo bolj pomembno. Ti KPI-ji imajo med drugim lahko tudi podatke o zemljepisni dolžini in širini ter radiju, ampak takih KPI-jev v našem primeru nismo zasledili. Podprli smo avtomatsko osveževanje seznamov. Časovne premore med osveževanji je mogoče spremeniti v nastavitvah, prav tako pa lahko osveževanje popolnoma onemogočimo.

Če se vrnemo nazaj in izberemo drugi tip KPI-ja, se pomaknemo v pogled s črtnim grafom, ki lahko vsebuje več serij točk. Abscisna os predstavlja datum in čas, ordinatna pa vrednost posamezne serije KPI-ja. Graf vsebuje tudi legendo, iz katere so razvidne prisotne serije. Pri grafu smo se odločili uporabiti izgled, ki bo čim bolj podoben tistemu s spletnega vmesnika platforme Occapi. Za točke smo uporabili kroge brez polnila, ki so med seboj

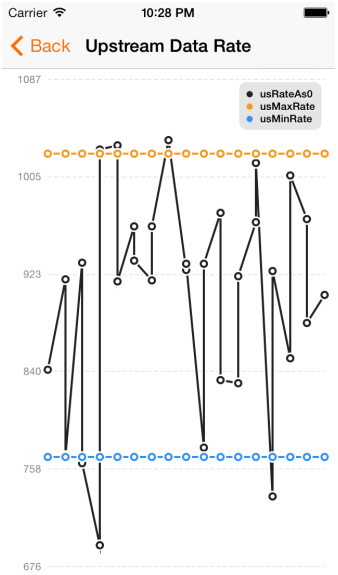
povezani s črto. Barve osi, ozadja in točk smo prav tako uporabili skladno s spletnim vmesnikom. Tako kot seznami se tudi grafi avtomatsko osvežujejo.



Slika 3.5: Graf izdelan z AChartEngine



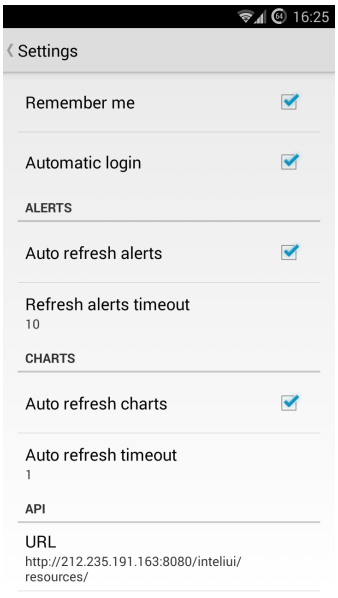
Slika 3.6: Sencha Touch graf



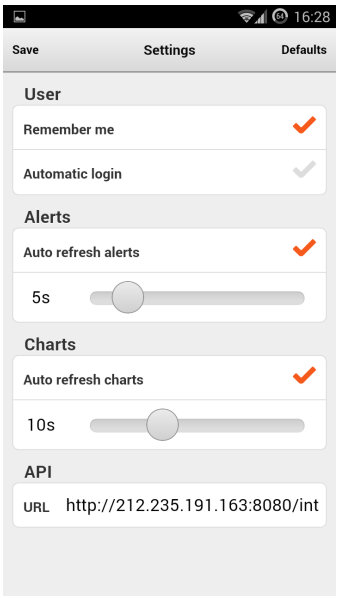
Slika 3.7: Graf izdelan z ios-linechart

Za razliko od spletnih strani je pri mobilnih aplikacijah ponavadi potrebna samo enkratna prijava, kar smo tudi realizirali. V drsnem meniju smo omogočili tudi opcijo za pregled informacij o aplikaciji, nastavitve in odjavo. Izgled pogleda z nastavitvami je viden na slikah 3.8, 3.9 in 3.10. V nastavitvah smo podprli:

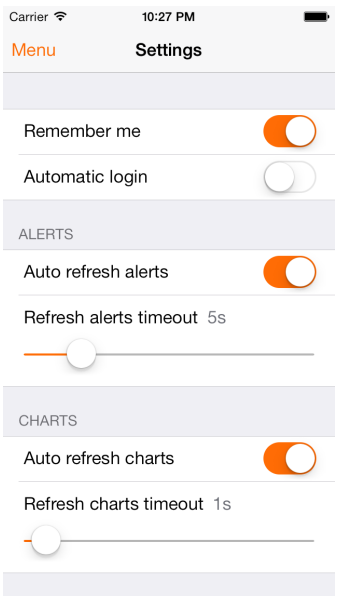
- shranjevanje uporabniških vstopnih podatkov,
- avtomatsko prijavo v sistem ob zagonu aplikacije,
- vklop in izklop avtomatskega osveževanja seznamov in grafov,
- časovni premor med posamezno osvežitvijo,
- spremembo spletnega naslova za API.



Slika 3.8: Android nastavitve



Slika 3.9: Sencha Touch nastavitve



Slika 3.10: iOS nastavitve

3.2.1 API

Platforma Occapi ima API, ki nam omogoča dostop do funkcij platforme iz drugih aplikacij. API vse podatke vrača v obliki JSON (angl. JavaScript Object Notation). Podpira naslednje funkcionalnosti:

- prijavo z elektronskim naslovom in geslom, odgovor vsebuje žeton, ki se uporablja pri vseh naslednjih poizvedbah;
- prenos skupin KPI-jev;
- prenos KPI-jev, ki spadajo pod podano skupino, katere ime podamo kot parameter;
- prenos seznama alarmov za vse KPI-je;
- prenos serij dvodimenzionalnih točk za posamezen KPI.

3.2.2 Sencha Touch

Razvoja smo se najprej lotili v Sencha Architectu, ker nam je od uporabljenih programskih okolij slednji še najbolj domač. Že takoj na začetku pa smo naleteli na problem pri dostopu do strežnika. Pri poskusu prijave v sistem smo naleteli na napako:

```
XMLHttpRequest cannot load http://212.235.191.163:8080...
Origin http://localhost:8888...
is not allowed by Access-Control-Allow-Origin.
```

To se zgodi, ker gre za zahtevo, ki ni v domeni trenutne strani. Brskalnik ne dovoli, da bi zahtevali stran, ki ni v isti domeni kot trenutna stran. To lahko v brskalniku Chrome zaobidemo tako, da ga zaženemo s parametrom `--disable-web-security`. To pa je bila samo začasna rešitev v samem začetku razvoja.

Nato smo namesto zahteve Ajax uporabili zahtevo JsonP, ki pa je omejena samo na metodo GET (branje). Takoj smo naleteli na nov problem, ker strežnik ne podpira formata zahteve JsonP. Čeprav se nam je uspelo prijaviti, brskalnik ni razumel odgovora. Pri zahtevi JsonP mora vsaka zahteva vsebovati parameter callback in če je ta enak `Ext.data.JsonP.callback123`, potem mora strežnik vrniti podatke v obliki:

```
Ext.data.JsonP.callback123({
    token : "9a479be6f0730e77e08b372633ac0baa",
    status : true,
    message : "Login successful"
})
```

Kasneje smo napisali preprosto skripto PHP, ki je služila kot posrednik. Skripta je sprejemala zahteve aplikacije, prenesla podatke s strežnika in jih posredovala aplikaciji. Ker smo pri skripti določili, da lahko sprejema zahteve iz vseh domen, ni bilo več težav pri prenosu podatkov, edino, kar se je spremenilo, je bila povečana latenca, ki jo je povzročil posrednik. Nazadnje

smo posegli v API in v glave odgovorov dodali navodila, da se lahko zahteve berejo iz poljubne domene.

Za glavni pogled aplikacije smo uporabili komponento `Ext.navigation.View`, v katero dodajamo poglede programsko s klicem metode `push`, ki ji podamo pogled in besedilo naslova. Pri pogledu z menijem smo uporabili komponento `Ext.Container`, ki ima parameter za razporeditev otrok nastavljen na `card`. Tako je viden samo eden od pogledov, poglede pa preklapljamo s klicem metode `setActiveItem`. Meni smo izdelali s komponento `Ext.Sheet`, ki je plovna in modalna. Vsebuje seznam, ki smo ga stilsko prilagodili, da je bolj podoben Androidovemu drsnemu meniju. Seznam uporablja shrambo, v kateri so statični vnosi. Vsak vnos vsebuje besedilo naslova, identifikator pogleda, barvo ikone in črko, ki se uporablja za prikaz ikone.

Prehodi med pogledi so animirani. Želeli smo uporabiti identičen prehod pri vseh treh aplikacijah. Navigacijski komponenti smo lastnost `animation` nastavili na `slide`. Pogledom, ki pa se ne uporabljajo v navigacijski komponenti, pa je bilo potrebno nastaviti animacijo za prikaz in skrivanje:

```
hideAnimation: {
    type: "slideOut",
    direction: "right"
},
showAnimation: {
    type: "slide",
    direction: "left"
}
```

Za prenos podatkov preko različnih pogledov smo uporabili globalno spremenljivko `App`, ki smo jo definirali ob zagonu aplikacije. Tako se žeton, ki ga prejmemo pri prijavi, shrani in ga lahko uporabljamo povsod v aplikaciji:

```
var t = App.user.token;
```

Ker API vrača podatke v obliki JSON, je to idealno za JavaScript. Prav tako ni bilo nobene potrebe po obdelavi podatkov, vse odgovore smo lahko

neposredno uporabili, le pri prijavi in grafih je bilo potrebno ročno izluščiti podatke. Tu smo uporabili zahtevo Ajax in odgovor v obliki niza pretvorili v objekt, kar lahko storimo s klicem metode `Ext.decode(odgovor)`. Pri seznamih smo uporabili pristop s shrambo, ki smo ji določili parameter naslova, s katerega prenese podatke, ter model s primernimi polji in njihovimi tipi. Seznami, ki uporabljajo shrambe, se sami osvežijo ob spremembi podatkov. Vse, kar je bilo potrebno določiti, je bila predloga celice seznama, s katero smo določili, kateri podatki naj se pokažejo, in stil, kjer je bilo to potrebno. Zahteva za prenos podatkov s strežnika se izvede ob klicu shrambine metode `load`, tako smo pri avtomatskem osveževanju samo ponavljali klic te metode.

Klic metode z zamikom je zelo preprost. Uporabili smo `Ext.defer`, podali metodo, ki naj se izvede, časovno zakasnitev v milisekundah, po potrebi pa še območje (angl. scope) in argumente. Pri podani metodi smo na koncu opravili ponovni klic le-te, s čimer smo dosegli ponavljanje osveževanja. Metoda `Ext.defer` vrne identifikator, ki smo ga uporabili za prekinitev osveževanja z uporabo brskalnikove metode `clearTimeout(identifikator)`.

Grafi za vir podatkov prav tako uporabljajo shrambe. Pri modelih ni podprt tip objekta, zato nismo mogli neposredno uporabiti shrambe, ker so podatki o točkah grafa globlje v objektovi hierarhiji. Podatke smo s strežnika prenesli z zahtevo Ajax, nato smo izluščili potrebne podatke o točkah in jih ročno dodali v shrambo. Ker abscisna os predstavlja čas, smo uporabili časovni tip osi, ki je že podprt, potrebno je bilo samo določiti format prikaza časa. S samim stilom ni bilo nobenih težav, hitro smo izgled približali grafu s spletnega vmesnika.

Sencha Touch sam po sebi ne ponuja uporabniških nastavitev. Izdelali smo pogled s potrebnimi polji za nastavitve. Njihove vrednosti smo kot JSON shranili v piškotek brskalnika. Alternativa bi bila uporaba brskalnikove lokalne shrambe ali baze SQLite.

Sencha Architect je vseboval vse komponente, ki so bile potrebne za realizacijo zahtev, tako da smo aplikacijo izdelali brez uporabe dodatnih knjižnic. Logotip, ikona za pomik naprej v seznamu ter ikoni za seznam in graf so edine

stvari, ki niso izdelane v Sencha Architectu. Za izdelavo naše aplikacije je bil ta način razvoja popoln.

3.2.3 Android

Za prenos podatkov med pogledi smo na začetku uporabili pristop, ki je značilen za Android, preko dodatkov (angl. extras). Pri začetni verziji smo najprej prenesli podatke in jih preko dodatkov posredovali naslednjemu pogledu. Pri seznamu alarmov, kjer gre za malce večjo količino podatkov, smo naleteli na napake, kot so sesutje aplikacije in prazni seznami, ki pa nam jih ni uspelo takoj odkriti. Ugotovili smo, da obstaja neka zgornja meja količine podatkov, ki se lahko prenaša preko dodatkov, ki smo jo očitno prekoračili. Nato smo se lotili podobnega pristopa kot pri Sencha Touch aplikaciji. Izdelali smo razred `App`, ki je razširjal razred `Application`, ki smo mu dodali spremenljivke za hranjenje podatkov. Tako smo lahko prav tako od povsod dostopali do žetona, ki ga prejmemo ob prijavi:

```
String t = ((App)getApplicationContext()).getUser().getToken();
```

Za razliko od Sencha Architecta je bilo potrebno vse podatke, ki smo jih prejeli s strežnika pretvoriti v obliko, ki jo lahko uporabimo v programu. Uporabili smo za to namenjeno knjižnico JSON-Simple [11]. Sprva je vse delovalo hitro, ko pa je bilo potrebno 1 MB velik niz s podatki pretvoriti v objekt, je operacija `new JSONObject(niz)` trajala 5 sekund. Na spletnih forumih smo zasledili, da naj bi bila Googlova knjižnica Gson [12] pri tem hitrejša. Izkazalo se je, da to drži, vendar je pretvorba še vedno trajala 4 sekunde. Nazadnje smo uporabili knjižnico Jackson [13], ki naj bi bila pri tem najhitrejša. Ta knjižnica pa sploh ni podpirala pretvarjanja niza v JSON. Namesto tega smo morali izdelati javanske razrede, skladne s tistimi v prejetih odgovorih. Pri grafih je bilo to precej nerodno, saj smo morali izdelati tri razrede, da smo prišli do podatkov o točkah. Knjižnica je pri pretvorbi za parameter zahtevala vhodni tok in vrnila podatke kot javanski objekt. S takim pristopom pa je bila razlika občutna, saj se je pretvorba izvedla v

trenutku. Prav tako smo lahko ta javanski objekt neposredno uporabili v adapterju za seznam.

Vse prenose podatkov s strežnika smo izvedli asinhrono, ker bi bila drugače aplikacija med prenosom neodzivna. Pri Sencha Touch aplikaciji se že vsi prenosi podatkov izvajajo asinhrono, zato za prenose le-teh nismo potrebovali veliko časa. Tu pa smo za vsak prenos podatkov ustvarili razred, ki razširja razred `AsyncTask`. Metodi `doInBackground` podamo seznam parametrov, ki so potrebni za prenos podatkov, po končanem prenosu pa s stavkom `return` posredujemo podatke metodi `onPostExecute`, kjer jih uporabimo za predstavitev v aplikaciji (posodobimo seznam ali graf). Avtomatsko osveževanje smo izvedli z uporabo razreda `Handler` s klicem metode `postDelayed`, kjer v argumentih podamo metodo, ki naj se izvede, in časovni zamik v milisekundah.

Android podpira izdelavo navigacijskega drsnega menija (angl. Navigation Drawer). Meni za prikazovanje pogledov ne uporablja aktivnosti, ampak delčke (angl. Fragments). Pri seznamih se je to izkazalo za zelo uporabno zadevo. Na začetku smo za vsak seznam ustvarili novo aktivnost, v pripadajoči datoteki `.xml` smo dodali komponento in sicer seznam in nato še datoteke za definiranje izgleda posamezne celice. Prav tako je potrebno za vsak seznam izdelati še adapter, ki napolni vsako celico z vsebino. Z uporabo delčka pa je bilo potrebno izdelati razred, ki je razširjal `ListFragment`, in ob njegovem nastanku določiti adapter s klicem metode `setListAdapter(adapter)`. S tem smo se znebili nepotrebne datoteke `.xml`.

Ostali pogledi, ki se ne uporabljajo v meniju, se povezujejo samo s pomočjo programskega klica. Na mestu, kjer želimo prikazati željen pogled, izvedemo:

```
Intent intent = new Intent(KpisActivity.this,
        ChartActivity.class);
startActivity(intent);
overridePendingTransition(R.anim.right_to_left,
        R.anim.right_to_left_exit);
```

Potrebno je bilo tudi definirati podrobnosti animacij v datotekah `.xml`. Primer definiranja animacije za prikaz naslednje aktivnosti (datoteka `right_to_left.xml`):

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">
    <translate
        android:fromXDelta="100%"
        android:toXDelta="0%"
        android:fromYDelta="0%"
        android:toYDelta="0%"
        android:duration="250" />
</set>
```

Android SDK sam po sebi ne podpira izdelave grafov, zato smo uporabili knjižnico AChartEngine. Abscisna os grafa predstavlja čas, kar nam ni povzročalo težav, ker knjižnica to podpira. Namesto navadnih serij točk smo ustvarili časovne serije (`new TimeSeries(nazivSerije)`). Seriji preprosto podamo točko s klicem metode `serija.add(cas, y)`. Kar knjižnica ne podpira, je barva polnila točk. Kljub temu pa smo dosegli izgled, ki je zelo podoben grafu platforme Occapi.

Nastavitve, kakršne podpira Android, so bile za naše potrebe uporabne. Nismo uporabili nastavitev, ki se jih ustvari preko čarovnika, ker bi nastale nepotrebne datoteke. Ustvarili smo aktivnost za nastavitve in `.xml` datoteko, v kateri so definirane skupine in njihove nastavitve. Na žalost v nastavitvah ni podprt drsnik, zato smo za nastavitev časovnih zamikov uporabili tekstovno polje, ki smo mu določili, da lahko sprejema samo številke in mu omejili dolžino na dve števki.

Pri razvoju v Android SDK smo uporabili še največ knjižnic izmed vseh treh načinov, to pa zaradi težav pri pretvarjanju JSON objektov. Uporabna funkcija razvojnega okolja Eclipse je bilo ustvarjanje ikon, tako jih ni bilo potrebno ustvariti v drugem programu ali prenašati s spleta. Kar nas je

motilo, je počasnost emulatorja, zato smo namesto tega uporabili fizično napravo, kar je olajšalo preizkušanje in razhroščevanje aplikacije.

3.2.4 iOS

Domorodno aplikacijo za iOS smo razvijali prvič, zato nam je vzela največ časa. Pri prenosu podatkov s strežnika smo uporabili delegate. Ustvarili smo datoteko `API.m`, v kateri smo napisali celotno kodo za prenos podatkov s strežnika. Na mestih, kjer potrebujemo podatke s strežnika, pokličemo metodo iz te datoteke. Pri prijavi klic iz kontrolerja izgleda takole:

```
API *api = [[API alloc] init];
api.delegate = self;
[api login:tfEmail.text :tfPassword.text];
```

Metoda `login` prenese podatke s strežnika, nato pa jih posreduje prijavnemu kontrolerju s klicem njegove metode `[self.delegate loginCompleted:YES : message]`. S tem pristopom smo dosegli bolj pregledno kodo, saj je koda za dostop do strežnika ločena od ostale kode.

Prav tako kot pri Androidu, smo morali tudi tu vse podatke prejete s strežnika pretvoriti v obliko, ki smo jo lahko uporabili v aplikaciji. Prejete podatke smo v slovar `NSDictionary` pretvorili s klicem metode `[NSJSONSerialization JSONObjectWithData:podatki options:kNilOptions error:&error]`. Pri uporabi nismo zasledili, da bi bilo pretvarjanje počasno, kot je bilo na Androidu, zato ni bilo potrebno izdelati razredov za vse objekte JSON. Vrednost dobimo iz slovarja z uporabo metode `[json objectForKey:-kljuc]`.

Prenašanja podatkov med pogledi smo se lotili na način, ki smo ga uporabili pri prejšnjih dveh aplikacijah. Ustvarili smo edinski (angs. singleton) globalni razred, ki smo ga uporabili v vseh kontrolerjih. Tako smo lahko žeton, ki ga prejmemo ob prijavi, uporabili na naslednji način:

```
DataClass *d = [DataClass instance];
NSString *t = d.token;
```

Za razliko od prejšnjih dveh aplikacij smo tu za izdelavo drsnega menija morali uporabiti dodatno knjižnico, ker tega iOS sam po sebi ne podpira. Uporabili smo knjižnico ECSlidingViewController.

Za risanje grafov smo prav tako uporabili dodatno knjižnico in sicer ios-linechart. Izmed vseh treh aplikacij je bilo tu potrebno največ dela, vendar je bilo pri tem tudi več nadzora in tako je graf najbolj podoben tistemu iz platforme Occapi. Knjižnica za abscisno os ne podpira časa, ampak samo decimalna števila z enojno natančnostjo (`float`), zato smo morali to implementirati sami. Problem se je pojavil, ker je dobljen čas zapisan v obliki časovnega žiga (število milisekund od leta 1970), kar se ne da shraniti kot `float`. Tip `float` je 32-bitno decimalno število, pri katerem se uporablja 1 bit za predznak, 8 za eksponent in 23 za mantiso, torej imamo 23 bitov za natančnost. Časovni žig do današnjega dne pa za zapis potrebuje najmanj 41 bitov, kar pomeni, da manjka 18 bitov za doseganje take natančnosti, oziroma 8 bitov, ker smo številko delili s 1000, saj v našem primeru milisekunde lahko zanemarimo. Težavo smo rešili tako, da smo za referenco vzeli najmanjši časovni žig, ki smo ga potem odšteli od vseh vrednosti. Na ta način se je število potrebnih bitov na primer zmanjšalo na 17 bitov, če so točke v razponu enega dneva, zato smo lahko te vrednosti uporabili v grafu. Pri prikazu dejanske vrednosti v grafu pa smo dodali vrednost najmanjšega žiga in nato pretvorili v uporabniku razumljivo obliko.

Avtomatsko osveževanje smo izvedli z zakasnjenim klicem metode za prenos podatkov o alarmih oziroma grafih. Pri tem smo uporabili `performSelector`, ki ji podamo metodo, ki naj se izvede, in časovni zamik v sekundah.

Pogled z nastavitvami smo izdelali sami, ker tako omogoča več kontrole, poleg tega pa je bila standardna lokacija nastavitev pri iOS-u, skozi sistemske nastavitve, nesprejemljiva, ker smo hoteli doseči čim večjo podobnost med vsemi tremi aplikacijami. Izdelava statičnih seznamov je bila tu še najlažja. Preprosto smo nastavili število skupin, skupaj z njihovim nazivom, dodali celice in vanje vstavili potrebne elemente kot so labele, stikala, be-

sedilna polja in drsniki. Ob odpiranju pogleda smo nastavili vsebine polj glede na pripadajočo vrednost v nastavitvah. Pri spremembi vsebine polj pa se spremenjene vrednosti zapišejo v nastavitve. Vrednost posamezne nastavitve preberemo z ukazom `[[NSUserDefaults standardUserDefaults] objectForKey: IME_NASTAVITVE]`, shranjevanje nastavitev pa je prav tako preprosto. Za privzete vrednosti nastavitev smo ustvarili datoteko `defaultPrefs.plist`, ki vsebuje ključe in vrednosti v formatu XML (angl. Extensible Markup Language). Privzete nastavitve je bilo potrebno pri vsakem zagonu aplikacije registrirati.

```
NSString *defaultPrefsFile = [[NSBundle mainBundle]
        pathForResource:@"defaultPrefs" ofType:@"plist"];
NSDictionary *defaultPreferences = [NSDictionary
        dictionaryWithContentsOfFile:defaultPrefsFile];
[[NSUserDefaults standardUserDefaults]
        registerDefaults:defaultPreferences];
```

3.3 Primerjava kode

Naredili smo kratko primerjavo programske kode iz vsakega razvojnega okolja. Pri primerjavi smo bili še posebej pozorni na obseg in razumljivost kode.

Celotne izvorne kode vseh treh aplikacij so dostopne na spletni strani GitHub [20, 21, 22].

3.3.1 Sencha Touch

Tu je koda še najmanj obsežna in tudi najbolj razumljiva. V programski kodi 3.1 je razvidna pretvorba časovnega žiga v datum, ki vsebuje leto, mesec in dan. Zahteva podatkov s strežnika, razvidna v programski kodi 3.2, se izvede asinhrono, ob uspešnem prenosu pa se izvede funkcija podana kot parameter `success`, v kateri se prejeti tekst pretvori v objekt.

```
var date = new Date(timestamp);
return Ext.Date.format(date, "Y-d-m");
```

Programska koda 3.1: Pretvorba časovnega žiga v besedilo v Sencha Touchu

```
Ext.Ajax.request({
    url: settings.get("apiUrl") + "login/" + user + "/" + pass,
    success: function(response, opts) {
        var obj = Ext.decode(response.responseText);
        if (obj.token) { /* ... */ }
    }
});
```

Programska koda 3.2: Zahteva za prijavo in preverjanje prisotnosti žetona v Sencha Touchu

Za večino klicev funkcij se uporablja poimenovane parametre (angl. named parameters), kar lahko vidimo v programski kodi 3.2. Funkciji pravzaprav podamo objekt, ki vsebuje parametre, v našem primeru `url` in `success`. Priporočljivo je vsak parameter zapisati v novo vrstico, da dosežemo pregledno programsko kodo.

3.3.2 Android

Koda se po obsežnosti in razumljivosti bistveno ne razlikuje od prejšnje, vendar moramo upoštevati, da smo morali pred tem ustvariti razred `User`, ki vsebuje vse potrebne spremenljivke ter njihove `set` in `get` metode. Prav tako je potrebno ustvariti asinhrono opravilo, drugače bi bila aplikacija med prenosom neodzivna.

```
Date date = new Date(timestamp);
return new SimpleDateFormat("yyyy-MM-dd", Locale.US)
        .format(dateTime);
```

Programska koda 3.3: Pretvorba časovnega žiga v besedilo v Javi

```
public class Login extends AsyncTask<Void, String, Void> {
    @Override
    protected Void doInBackground(Void... params) {
        String apiUrl = sharedPref.getString(
                SettingsActivity.KEY_PREF_API_URL,
                C.DEFAULT_API_URL);
        URL url = new URL(C.apiURL + "login/"
                + mEmail + "/" + mPassword);
        ObjectMapper mapper = new ObjectMapper();
        User user = mapper.readValue(new InputStreamReader(
                url.openStream()), User.class);
        if(user.getToken()) { /* ... */ }
    }
}
```

Programska koda 3.4: Zahteva za prijavo in preverjanje prisotnosti žetona v Javi

3.3.3 iOS

Koda je tu še najbolj obsežna, ni pa bistvene razlike v razumljivosti. Pretvarjanje časovnega žiga v datum (vrstica 3.5) tu zahteva dodatno deljenje s 1000, podamo ga metodi namesto konstruktorju datuma in potrebno je ustvariti `formatter`. Pri prenosu podatkov s strežnika (vrstica 3.6) so klici metod nekoliko daljši kot pri Androidu, ker zahtevajo več parametrov. Če primerjamo obseg kode pri Androidu in iOS-u z obsegom le-teh v Sencha Touchu, lahko rečemo, da je koda pri Androidu in iOS-u nekoliko daljša, razlog

za to pa je v tipiziranem programskem jeziku.

```
NSDate *date = [NSDate dateWithTimeIntervalSince1970:
               (timestamp / 1000)];
NSDateFormatter *formatter = [[NSDateFormatter alloc] init];
return [formatter setDateFormat:@"yyyy-MM-dd"];
```

Programska koda 3.5: Pretvorba časovnega žiga v besedilo v objektnem C-ju

```
- (void) login:(NSString*)email password:(NSString*)password {
    NSString *apiUrl = [[NSUserDefaults standardUserDefaults]
            objectForKey:API_URL];
    NSString *urlAsString = [NSString stringWithFormat:
            @"%@login/%@/%@", apiUrl, email, password];
    NSURL *url = [[NSURL alloc] initWithString:urlAsString];
    [NSURLConnection sendAsynchronousRequest:[[NSURLRequest
            alloc] initWithURL:url] queue:[NSOperationQueue
            mainQueue] completionHandler:^(NSURLResponse
            *response, NSData *data, NSError *connectionError){
        NSError *error;
        NSDictionary *json = [NSJSONSerialization
                JSONObjectWithData:data options:kNilOptions
                error:&error];
        NSString *token = [json objectForKey:@"token"];
        if((NSNull*)token == [NSNull null]) { /* ... */ }
    }
}
```

Programska koda 3.6: Zahteva za prijavo in preverjanje prisotnosti žetona v objektnem C-ju

3.4 Razhroščevanje

Najpriročnejše je razhroščevanje Sencha Touch aplikacije, saj večino dela lahko opravimo kar v enem izmed podprtih spletnih brskalnikov na osebnem računalniku. Prav tako ni večjih težav pri razhroščevanju aplikacije na Android in iOS napravah. Pri iOS napravi omogočimo razhroščevanje v nastavitvah brskalnika Safari. Prav tako moramo omogočiti razvijalski način v brskalniku Safari na osebnem računalniku. Napravo priklopimo na računalnik preko kabla USB in v meniju Safarija izberemo **Develop > iOS naprava > aplikacija**. Sedaj je razhroščevanje identično tistemu v Safariju na osebnem računalniku. Podobno kot v Safariju je enostavno razhroščevanje aplikacije na Androidu v brskalniku Chrome. Najprej moramo omogočiti razvijalski način v nastavitvah naprave. Nato priklopimo napravo preko kabla USB na osebni računalnik in v brskalnik Chrome vpišemo naslov **about:inspect**. Izberemo odprto spletno stran in kliknemo **inspect**.

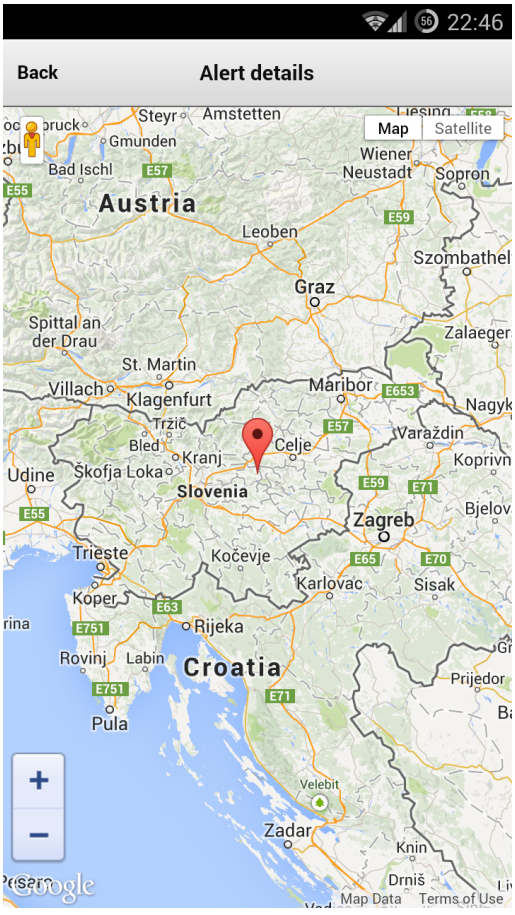
Razhroščevanje domorodnih aplikacij je preprosto, vendar moramo imeti pri razvoju za Android priklopljeno napravo preko kabla USB, saj je že samo poganjanje aplikacije v emulatorju počasno, kaj šele razhroščevanje. Eclipse vsebuje vsa orodja za kvalitetno razhroščevanje aplikacije. Za razhroščevanje domorodne aplikacije, kot tudi Sencha Touch aplikacije, na napravi Android moramo pred tem namestiti ADB (angl. Android Debug Bridge).

Pri iOS razvoju lahko uporabimo iOS simulator, ker deluje zelo hitro, še bolje pa je, če uporabimo napravo priklopljeno preko kabla USB.

3.5 Posodabljanje aplikacije

Predpostavimo, da želimo naši aplikaciji dodati novo funkcionalnost. Omenili smo že, da imajo lahko KPI alarmi podano lokacijo. To lokacijo bi lahko pokazali na zemljevidu. Pri Androidu je bilo potrebno uporabiti Google Maps Android API v2 [15], pri iOS-u pa Google Maps SDK for iOS [16]. Implementacija sicer ni zahtevna, ampak jo je potrebno narediti na dveh mestih. Če bi bila aplikacija na trgovini Google Play in Appstore, bi se tudi sama

posodobila na vseh napravah, kjer je nameščena, ker pa je aplikacija ročno naložena na napravo, je potrebno pri vsakem popravku posodobiti aplikacijo. Če pa uporabimo HTML5 pristop, je potrebno dodatno funkcionalnost izdelati samo enkrat in deluje na vseh podprtih napravah. Če uporabimo pristop, kjer se vse datoteke nalagajo s strežnika, potem dejansko ni potrebna posodobitev aplikacije, ker se ob zagonu prenesejo nove datoteke, kjer je funkcionalnost že podprta. V Sencha Touchu so Googlovi zemljevidi že podprti, kar nam še olajša izdelavo. Na sliki 3.11 vidite, da smo to tudi preizkusili. Po novem je v Sloveniji podprt Google Street View, prikazan na sliki 3.12, ki deluje tudi v aplikaciji.



Slika 3.11: Google Maps v Sencha Touch aplikaciji

Slika 3.12: Google Street View v Sencha Touch aplikaciji

3.6 Podpiranje dodatnih naprav

Zamislimo si scenarij, kjer bi radi našo aplikacijo podprli še na nekem dodatnem operacijskem sistemu, na primer Windows 8, ker želimo, da naša naprava deluje na čim več mobilnih napravah. Za razvoj potrebujemo Visual Studio. Aplikacijo lahko razvijamo v programskem jeziku VisualBasic, C++ ali C#, zadnji je še najbolj podoben programskemu jeziku Java, v katerem se programira aplikacije za Android. Potrebno bi se bilo naučiti vseh novih pristopov izdelave aplikacije za Windows 8. Poleg razvojnega časa, bi se povečal tudi čas, ki ga potrebujemo za vzdrževanje aplikacije. Potrebno bi bilo vzdrževanje že kar treh aplikacij. Potem je pa tu še BlackBerry.

Aplikacija razvita s Sencha Touchem pa je podprta na Windows 8 in BlackBerry, saj so standardi HTML5 več ali manj podprti na vseh mobilnih brskalnikih. Potrebno bi bilo izdelati preprosto domorodno lupino in izvesti dodatno testiranje. BlackBerry že sam ponuja razvoj aplikacij v HTML5, brskalnik IE 10 na Windows 8 pa prav tako podpira standarde HTML5. Podpiranje novih naprav na ta način ni tako zahtevno kot pri razvoju domorodne aplikacije.

3.7 Grafično oblikovanje aplikacije

3.7.1 Simboli in ikone

Pri Sencha Touchu so v verziji 2.2 uvedli nov način uporabe simbolov. Uporabili so pisavo, ki namesto črk vsebuje simbole. Priložena je pisava z 94-imi osnovnimi simboli. Tako na primer črka "H"predstavlja simbol "Domov". Prednost tega pristopa je, da so simboli v pisavi predstavljeni vektorsko, kar pomeni, da lahko simbol poljubno povečamo in ne bomo izgubili kvalitete, kot se to zgodi pri binarnih slikah. Prav tako lahko s pomočjo podloge CSS preprosto spreminjamo barvo simbola, enako, kot bi spreminjali barvo pisave.

Zaradi vseh prednosti smo ikone v stranskem meniju izdelali na ta način, barve pa smo lahko nastavili neposredno v brskalniku in takoj videli rezultat

ter dobljene vrednosti prekopirali v datoteko CSS.

Android SKD podpira izdelavo ikon. Lahko uvozimo lastne slike, sestavimo tekst ali pa uporabimo simbole, ki so priloženi. Ko končamo postopek se izdela več binarnih slik za podporo različnih DPI-jev (angl. Dots Per Inch). Ko je ikona, spreminjanje velikosti in barve ni več mogoče.

Xcode podpira samo uvoz ikon. Tako moramo vse ikone ročno izdelati v drugem programu ali pa poiskati primeren paket le-teh.

Da bi si pomagali pri izdelavi ikone aplikacije, smo uporabili predlogo App Icon Template [10].

Poglavje 4

Primerjava aplikacij

Po končanem razvoju smo opravili testiranje delovanja posamezne aplikacije. Primerjali smo tudi končno dolžino kode, ki smo jo napisali. Pri končni primerjavi smo upoštevali vse naslednje kriterije:

- hitrost zagona,
- odzivnost dotikov,
- gladkost prehodov,
- hitrost delovanja funkcij,
- obsežnost kode (število vrstic),
- potrebno tehnično znanje,
- cene potrebne programske opreme.

4.1 Delovanje aplikacije

V tabeli 4.1 so izmerjeni časi, ki so potrebni za zagon aplikacije. Za merjenje časa smo uporabili sistemsko štoparico na Androidu, ki je natančna na stotinko sekunde. Vsako verzijo aplikacije smo zagnali petkrat in vzeli povprečje petih časov zagona. V primerih, kjer se je aplikacija zaganjala eno sekundo

ali manj, so rezultati zaokroženi na desetinko sekunde natančno. Kjer pa je aplikacija za zagon potrebovala več kot eno sekundo, so rezultati zaokroženi na pol sekunde natančno.

Časi so bili izmerjeni od pritiska ikone aplikacije do pojava prijavnega zaslona. Prvi zagon je izmerjen takoj po namestitvi aplikacije, brez uporabniških nastavitev in predpomnenja (angl. caching). Aplikacija, ki je razvita s Sencha Touchem, pri prvem zagonu potrebuje več časa kot pri naslednjih zagonih, saj je aplikacija izdelana tako, da se vse datoteke nahajajo na strežniku in se ob prvem zagonu v celoti prenesejo. Pri nadaljnjih zagonih brskalnik ne naloži datotek, če se te niso spremenile, zato je zagon hitrejši. Prav tako pa je zagon še vedno bistveno počasnejši od domorodnih aplikacij. To je zaradi zahtev na strežnik in ustvarjanja vseh funkcij in objektov ob zagonu. Zaganjanje aplikacije si lahko predstavljamo kot zagon brskalnika in odpiranje nekoliko kompleksnejše strani, odvisno je tudi od kvalitete internetne povezave. Hitrejši zagon bi lahko dosegli tako, da ne bi vseh pogledov ustvarili na začetku, ampak šele, ko se potrebujejo, kar pa bi upočasnilo samo delovanje aplikacije. Opazili smo tudi, da se aplikacija na iOS-u zažene hitreje kot na Androidu, to pa zaradi brskalnika Safari.

Tip aplikacije in testna naprava	prvi zagon	nasl. zagon
Sencha Touch (Samsung Galaxy Nexus)	8.0 s	5.5 s
Sencha Touch (Asus Nexus 7 (2012))	7.5 s	4.5 s
Sencha Touch (iPod Touch 5)	5.5 s	5.0 s
domorodna (Samsung Galaxy Nexus)	1.0 s	1.0 s
domorodna (Asus Nexus 7 (2012))	1.0 s	1.0 s
domorodna (iPod Touch 5)	0.8 s	0.8 s

Tabela 4.1: Primerjave hitrosti zagona aplikacije

Zagon aplikacije bi bil hitrejši, če bi programsko kodo skrajšali, z uporabo `build` funkcije. To bi zmanjšalo velikost datotek s programsko kodo in tudi datoteko Sencha Touch knjižnice.

Odzivnost dotikov je pri Sencha Touch aplikaciji malce slabša, kar je bolj opazno na Androidu. Pri seznamih se prav tako opazi majhna razlika v odzivnosti dotikov in gladkosti pomikanja, spet se to bolj pozna na Androidu. Na iOS-u aplikacija deluje kot domorodna, če izvzamemo daljši čas zagona. Domorodni aplikaciji pa delujeta z odlično odzivnostjo, tako kot smo navajeni.

Prehodi so gladki tako v Sencha Touch aplikaciji kot v domorodnih aplikacijah in se izvedejo brez zatikanja in s približno enako hitrostjo. Nalaganje seznama alarmov deluje pri domorodnih aplikacijah hitreje, za prenos potrebuje povprečno **2.5 sekunde**. Izris grafov povsod deluje tekoče, pri Sencha Touch aplikaciji mogoče le prvo nalaganje potrebuje nekoliko več časa.

Domorodne aplikacije dajejo veliko prijetnejši občutek zaradi skoraj takojšnjega zagona, hitrih prenosov podatkov s strežnika, odzivnih dotikov in hitrega listanja seznamov. Razlike med domorodno in Sencha Touch aplikacijo niso velike, vendar se vsekakor občuti razlika, ki je večja na Androidu. Na iOS-u pa je bistvena razlika samo pri zagonu, drugače pa Sencha Touch aplikacija daje občutek domorodne aplikacije.

4.2 Primerjava razvoja in sredstev

Cene programov in knjižnic smo zapisali v ameriških dolarjih, ker so cene največkrat podane v tej valuti. Upoštevali smo, da programe uporabljamo več kot 30 dni. Upoštevali smo tudi, da si ne lastimo računalnika z operacijskim sistemom OS X.

4.2.1 Sencha Touch aplikacija

Pri razvoju smo uporabili preizkusno različico Sencha Architecta, ki jo lahko uporabljamo 30 dni. Po tem je potrebno kupiti licenco. Cene so razvidne v tabeli 4.2.

Program / knjižnica	cena
Sencha Touch	$0.00
Sencha Architect	$399.00
skupaj	**$399.00**

Tabela 4.2: Cene uporabljenih programov in knjižnic pri razvoju z uporabo standarda HTML5

Pri štetju vrstic kode smo upoštevali samo kodo, ki je v kontrolerjih in funkcijah aplikacije, nismo pa upoštevali kode, ki jo generira Architect. Torej smo upoštevali samo kodo, ki smo jo napisali ročno. Upoštevati pa moramo, da zaradi uporabe poimenovanih parametrov ena vrstica kode vsebuje manj znakov kot pri Javi in objektnem C-ju. Programska koda v celoti obsega **502 vrstici**. Stilska podloga CSS za izgled aplikacije pa obsega **146 vrstic**.

4.2.2 Domorodni aplikaciji

Pri razvoju domorodnih aplikacij smo uporabili samo programe in knjižnice, ki so popolnoma brezplačni, kot je razvidno v tabeli 4.3.

Program / knjižnica	cena
Android SDK	$0.00
AChartEngine	$0.00
VMWare Player	$0.00
XCode	$0.00
ECSlidingViewController	$0.00
ios-linechart	$0.00
skupaj	**$0.00**

Tabela 4.3: Cene uporabljenih programov in knjižnic pri razvoju domorodnih aplikacij

Tako kot pri Sencha Touch aplikaciji smo tudi tu upoštevali samo na ročno

napisano kodo. Pri aplikaciji za Android smo izvzeli kodo za razrede, ki definirajo strukturo objektov JSON, saj smo v Sencha Architectu prav tako izdelali modele za uporabo v shrambah, ki pa jih nismo upoštevali pri dolžini kode aplikacije. Pri uporabi razvojnega okolja Eclipse nam ni bilo potrebno ročno pisati ukazov za uvoz razredov (primer: `import android.app.Activity;`), ker za to poskrbi IDE sam, zato tudi teh nismo šteli. Programska koda obsega **1426 vrstic**. Datoteke z oblikami, barvami in stili pa obsegajo **163 vrstic**. Ker smo uporabili knjižnico AChartEngine, moramo upoštevati, da smo morali programsko določiti obliko grafa, kar poveča obsežnost kode.

Pri aplikaciji za iOS se pri ustvarjanju novega kontrolerja avtomatsko generirajo prazne funkcije, ki jih kontroler uporablja. Pri štetju smo izvzeli datoteke končnice `.h`, saj ne vsebujejo nobene dejanske kode. Programska koda obsega **1210 vrstic**. Spremembe izgleda aplikacije so se izdelale z uporabo grafičnega urejevalnika, zato obsega nismo mogli določiti. Prav tako kot pri aplikaciji za Android, smo morali tudi tu določiti obliko grafa programsko.

Programska koda obsega skupno **2636 vrstic**, kar je pet krat toliko kot pri Sencha Touch aplikaciji.

4.3 Končna ocena

Končno oceno smo podali na podlagi različnih faktorjev, ki so po našem mnenju pomembni pri razvoju in vzdrževanju aplikacije. Vsakega izmed faktorjev smo ocenili od 1 do 10, kjer je 1 najnižja ocena, 10 pa je najvišja ocena. Ocene so razvidne v tabeli 4.4.

Domorodne aplikacije definitivno delujejo malce hitreje kot Sencha Touch aplikacija, prav tako so malce bolj zanesljive. Sencha Touch aplikacija pa je hitrejša za razvoj, potrebnega je manj tehničnega znanja, saj je potrebno znanje samo enega programskega jezika, potrebno je manj zunanjih knjižnic, poleg tega pa so podprte tudi druge naprave, kot so Blackberry in Windows naprave.

	ST aplikacija	domorodna aplikacija
hitrost zagona	4	10
gladkost prehodov	8	10
hitrost delovanja funkcij	8	10
obsežnost kode	9	3
preglednost kode	9	5
potrebne zunanje knjižnice	10	6
potrebno tehnično znanje	8	5
čas razvoja	9	3
zanesljivost	7	9
pokritost mobilnih naprav	8	6
končna ocena	**8.0**	**6.5**

Tabela 4.4: Ocene bistvenih faktorjev pri aplikaciji in končna ocena

Poglavje 5

Sklepne ugotovitve

Razvili smo tri aplikacije, dve z uporabo klasičnega domorodnega pristopa, tretjo pa z uporabo standarda HTML5. Razvoj aplikacij z uporabo standarda HTML5 je vse bolj uporaben, ker so mobilne naprave vedno zmogljivejše. Tudi razlika v hitrosti delovanja teh aplikacij je v primerjavi z domorodnimi aplikacijami vedno manjša. Prav tako pravzaprav razvijamo in vzdržujemo le eno aplikacijo. Vse več je podjetjih, ki ponujajo svoje rešitve za poenostavljen razvoj HTML5 aplikacije. Tako podjetje je tudi Sencha, katerih rešitev Sencha Touch smo uporabili za izdelavo naše aplikacije. Razvoj je bil veliko hitrejši, kot razvijanje dveh domorodnih aplikacij za Android in iOS. Sencha Touch ima podprte vse funkcionalnosti, ki smo jih potrebovali v naši aplikaciji. Prav tako je koda krajša in bolj pregledna, vzdrževanje pa je preprostejše. Razlika v hitrosti delovanja je še opazna, a se bo s časom zmanjšala, ker vse stremi k izdelovanju HTML5 aplikacij.

V našem primeru bi se definitivno odločili za ta način razvoja. Če pa bi izdelovali aplikacijo, ki bi uporabljala veliko sistemskih funkcij, kompleksnih izračunov ali grafičnih operacij, bi bil razvoj domorodne aplikacije primernejši.

Literatura

[1] Platforma Occapi. Dostopno na:
http://www.opcomm.eu/sl/resitve/platforma-occapi

[2] Knjižnica Sencha Touch. Dostopno na:
http://www.sencha.com/products/touch

[3] Uradno podprte naprave knjižnice Sencha Touch. Dostopno na:
http://www.sencha.com/products/touch/features

[4] Sencha Architect IDE. Dostopno na:
http://www.sencha.com/products/architect

[5] Trgovina Sencha Architect. Dostopno na:
https://www.sencha.com/store/architect

[6] Android SDK. Dostopno na:
http://developer.android.com/sdk

[7] Knjižnica AChartEngine za izdelavo grafov na Androidu. Dostopno na:
https://code.google.com/p/achartengine

[8] VMWare Player, program za virtualizacijo. Dostopno na:
https://my.vmware.com/web/vmware/downloads

[9] Knjižnica za izdelavo grafov ios-linechart. Dostopno na:
https://github.com/mruegenberg/ios-linechart

[10] Predloga App Icon Template za izdelavo iOS ikone. Dostopno na: http://appicontemplate.com

[11] Knjižnica JSON-Simple. Dostopno na: https://code.google.com/p/json-simple

[12] Knjižnica Google-Gson. Dostopno na: https://code.google.com/p/google-gson

[13] Knjižnica Jackson. Dostopno na: http://jackson.codehaus.org

[14] ECSlidingViewController 2. Dostopno na: https://github.com/ECSlidingViewController/ECSlidingViewController

[15] Google Maps Android API v2. Dostopno na: https://developers.google.com/maps/documentation/android

[16] Google Maps SDK for iOS. Dostopno na: https://developers.google.com/maps/documentation/ios

[17] Primož Bečan, Razvoj medplatformne mobilne aplikacije v ogrodju MoSync, 2013. Dostopno na: http://eprints.fri.uni-lj.si/2134

[18] Aleksander Gregorka, Platforma Trafika v HTML5, 2013. Dostopno na: http://eprints.fri.uni-lj.si/1763

[19] Saša Nebojša Potežica, Razvoj aplikacij za pametne telefone, 2009. Dostopno na: http://eprints.fri.uni-lj.si/916

[20] Izvorna koda Sencha Touch aplikacije, objavljena na GitHubu. Dostopno na: https://github.com/drinovc/occapiMobile-html5

[21] Izvorna koda aplikacije za Android, objavljena na GitHubu. Dostopno na:
https://github.com/drinovc/occapiMobile-android

[22] Izvorna koda aplikacije za iOS, objavljena na GitHubu. Dostopno na:
https://github.com/drinovc/occapiMobile-ios