Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Primož Bečan

Razvoj medplatformne mobilne aplikacije v ogrodju MoSync

diplomsko delo
univerzitetni študijski program

Mentor: doc. dr. Dejan Lavbič

Ljubljana 2013

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: 01956/2013
Datum: 02.09.2013

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

Kandidat: **PRIMOŽ BEČAN**

Naslov: **RAZVOJ MEDPLATFORMNE MOBILNE APLIKACIJE V OGRODJU MOSYNC**

DEVELOPMENT OF CROSS-PLATFORM MOBILE APPLICATION IN MOSYNC

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Podpora mobilnim odjemalcem je pri razvoju informacijskih sistemov vedno bolj pomembna. Dostop do podatkov poslovnega informacijskega sistema preko mobilne naprave je zato pogosto del obveznih funkcionalnosti sistema. Težava pa se pojavi pri podpori širokega spektra mobilnih naprav z različnimi karakteristikami. Razvoj takšnih mobilnih aplikacij, ki podpirajo različne operacijske sisteme, velikosti zaslonov in hitrosti procesorjev, je zelo težaven in zahteva veliko prilagajanja. Ravno zaradi te težave pa so se pojavila različna ogrodja, ki takšen razvoj olajšajo. V okviru diplomske naloge naj študent s pomočjo ogrodja MoSync, ki podpira razvoj različnih mobilnih aplikacij, razvije informacijsko podporo za izbrano problemsko domeno obvladovanja sončnih elektrarn. Rezultat naj bo predstavljen v obliki delujočega prototipa z identificiranimi prednostmi in slabostmi uporabe ogrodja MoSync za razvoj mobilne aplikacije, primerne za več različnih naprav.

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 Primož Bečan, z vpisno številko **63070030**, sem avtor diplomskega dela z naslovom:

Razvoj medplatformne mobilne aplikacije v ogrodju MoSync

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, dne 9. septembra 2013 Podpis avtorja:

Kazalo

Povzetek

Pri razvoju programske opreme se pogosto pojavlja želja po podpori s strani mobilnih naprav. Mobilne aplikacije lahko razvijemo v okolju za domorodni razvoj, s pomočjo spletnih tehnologij in v hibridnem načinu. Okolje za razvoj domorodnih aplikacij je primerno za aplikacije, ki potrebujejo veliko sistemskih virov za svoje delovanje. Omogoča hitrejše izvajanje programske kode in dostop do vseh storitev, ki jih nudi naprava. Njegova slabost pa je počasnejši razvoj, različni programski jeziki glede na proizvajalca naprave in omejitev delovanja aplikacije na posamezni sklop naprav. Te pomanjkljivosti lahko do neke mere odpravimo s pomočjo spletnih tehnologij.

V diplomskem delu smo preučili, kako obstoječo aplikacijo podpreti tudi na mobilnih napravah. Za ta namen smo razvili mobilno aplikacijo za pregled nad proizvodnjo sončnih elektrarn. Cilj je bil razvoj aplikacije, ki bo delovala na mobilnih telefonih in tabličnih računalnikih z operacijskimi sistemi *Android*, *iOS* in *Windows Phone*. Prototip smo izdelali s pomočjo orodja *Codiqa*, ki omogoča hitro izdelavo zaslonskih mask brez dodatnega pisanja programske kode. Za izdelavo same aplikacije smo uporabili ogrodje *MoSync*. Zanj smo se odločili, ker poleg razvoja v spletnih tehnologijah ponuja tudi njihovo prevedbo v domorodni uporabniški vmesnik. Prikaz podatkov je realiziran z *jQuery mobile*, za komunikacijo z vgrajenimi senzorji naprave pa smo uporabili knjižnico *Wormhole*, ki je del ogrodja *MoSync*.

Ključne besede: medplatformni razvoj mobilnih aplikacij, MoSync, jQuery mobile, domorodni uporabniški vmesnik, Codiqa

Abstract

When developing software a desire for a mobile devices support is often expressed. Mobile applications can be developed in native development platform with the help of web technologies or in hybrid way. Native applications development platform is suitable for applications which need a great number of system sources for its operation. It enables fast performance of programme code and the access to all services offered by the device. Its weakness is slow development, various programme languages according to a device's producer and the limitation of application's performance. These limitations can be eliminated to a certain extent by using web technologies.

In my dissertation we study the support of the current application on mobile devices. To this end we have developed a mobile application for surveying the production of solar power plants. Our aim was to develop the application that would run on mobile phones and tablets with operating systems *Android*, *iOS* and *Windows Phone*. We developed a prototype using mobile development tool *Codiqa* which enables an easy development of forms without additional writing of programme code. For the development of the application itself we used the framework *MoSync*. We chose it as besides the development in web technologies it offers also their transformation in native user interface. Data display was realized by *jQuery mobile* while for the communication with integrated mobile device sensors we used library *Wormhole* that is a part of framework *MoSync*.

Keywords: cross-platform mobile application development, MoSync, jQuery mobile, native user interface, Codiqa

Poglavje 1

Uvod

Z razvojem mobilnih aplikacij se pojavlja želja po njihovi podpori na mobilnih napravah. Lahko se lotimo razvoja mobilne aplikacije za vsako platformo posebej, v primeru enostavne aplikacije pa imamo možnost izdelave aplikacije v medplatformnih tehnologijah, ki je nato podprta na več mobilnih napravah. Potrebno se je vprašati, ali je pomembna hitrost izvajanja aplikacije ter ali je aplikacija enostavnejša in v njen razvoj ne želimo vložiti veliko časa. Aplikacija, ki je napisana v domorodni kodi, se izvaja hitreje, vendar jo je potrebno razviti za vsako platformo posebej. Velikokrat pa želimo podpreti že realizirano aplikacijo še na mobilnih napravah. Za prikazovanje informacij in upravljanje z nekaj osnovnimi funkcionalnostmi aplikacije hitrost izvajanja ni tako pomembna, zato lahko aplikacijo razvijemo v medplatformnem ogrodju in s tem prihranimo na razvojnem času. V diplomskem delu smo že obstoječo spletno aplikacijo za pregled proizvodnje sončnih elektrarn podprli tudi na mobilnih napravah. Aplikacijo smo razvili v medplatformnem ogrodju za platforme *Android*, *iOS* in *Windows phone*.

V poglavju 2 so predstavljeni obstoječi pristopi za dani problem, v poglavju 3 pa tehnologije, ki smo jih uporabili pri izdelavi aplikacije. Shema delovanja celotnega sistema, od branja podatka pri sončni elektrarni do prikaza na mobilni napravi, je prikazana v poglavju 4. Odločili smo se za uporabo ogrodja *MoSync*, ki omogoča tako razvoj aplikacije v spletnih tehnologijah

kot tudi uporabo domorodnega uporabniškega vmesnika. Posledično smo aplikacijo razdelili na dve samostojni aplikaciji. V poglavju 5 je predstavljena prva aplikacija, ki je namenjena spremljanju proizvodnje sončnih elektrarn in je zgrajena v uporabniškem vmesniku, ki ga nudi *jQuery mobile*. Aplikacija za upravljanje s sončnimi elektrarnami, ki je namenjena vzdrževalcem, je predstavljena v poglavju 6. Razvili smo jo v domorodnem uporabniškem vmesniku orodja *MoSync*. Na koncu, v poglavju 7, podajamo sklepne ugotovitve in smernice za nadaljnje delo.

Poglavje 2

Pregled obstoječih pristopov

Pri izdelavi medplatformne mobilne aplikacije lahko izbiramo med različnim orodji [1] in tehnologijami. Pri izboru smo se osredotočili na naslednje zahteve:

- podpora platform *Android*, *iOS* in *Windows Phone*;
- uporaba senzorjev na mobilni napravi;
- razvoj v spletnih tehnologijah *HTML5* in *JavaScript*;
- možnost razvoja v domorodnih tehnologijah.

Zahteva za možnost razvoja v domorodnih tehnologijah ni bila obvezna pri izdelavi aplikacije za nadzor sončnih elektrarn, vendar jo je smiselno upoštevati, ker lahko pride do nadgradenj aplikacije, katerih ni mogoče realizirati v spletnih tehnologijah.

RhoMobile Suite

RhoMobile Suite [7] je odprtokodno ogrodje, ki ga je razvilo podjetje Motorola. Omogoča razvoj domorodnih aplikacij za platforme: *Windows Embedded*, *Windows CE*, *Windows Phone*, *Apple iOS*, *Android* in *BlackBerry*. Razvoj aplikacij poteka po vzorcu model-pogled-kontroler, kjer model in kontroler napišemo v jeziku *Ruby*, pogled pa v jeziku *HTML5*, *CSS3* in *Java-*

Script. Ogrodje ne podpira uporabo domorodnih gradnikov za gradnjo uporabniškega vmesnika, omogoča pa podobni učinek z uporabo *CSS3* in *JavaScript*, s katerima dosežemo različen izgled aplikacije na različnih mobilnih napravah.

Za ogrodje se nismo odločili, ker učenje novega programskega jezika (*Ruby*) predstavlja težavo pri želji po hitrem razvoju aplikacij.

Appcelerator Titanium

Appcelerator Titanium [6] je ogrodje za razvoj medplatformnih aplikacij, ki jih lahko namestimo na: *Android*, *iOS*, *BlackBerry* in *Tizen*. Aplikacije lahko razvijemo v jezikih *HTML5*, *CSS3* in *JavaScript* ter tudi v *PHP*, *Ruby* in *Python*. Za komunikacijo s senzorji naprave lahko uporabimo API in s tem pridobimo upravljanje senzorjev kar iz spletnih tehnologij. Prav tako imamo z uporabo API možnost gradnje domorodnega uporabniškega vmesnika.

Ker ogrodje ne podpira aplikacij na platformi *Windows Phone*, ni bilo primerno za razvoj naše aplikacije.

Phonegap

Phonegap [12] je ogrodje, ki omogoča gradnjo aplikacij izključno v spletnih tehnologijah *HTML5*, *CSS3* in *JavaScript*. Aplikacija teče znotraj domorodnega okna brskalnika, ogrodje pa ne omogoča njene prevedbe v domorodno kodo. Do senzorjev mobilne naprave dostopamo preko domorodnega vmesnika API, ki predstavlja most med jezikom *JavaScript* in domorodno kodo za komunikacijo s senzorji. Aplikacijo lahko razvijemo za platforme: *Android*, *iOS*, *Windows phone*, *Symbian*, *Bada*, *WebOS* in *Blackberry*.

Phonegap ne omogoča prevedbe kode v domorodni način, zato ni primerno za zahtevnejše aplikacije in aplikacije, kjer se morajo določeni deli aplikacije izvajati hitreje.

MoSync

MoSync [13] omogoča razvoj mobilnih aplikacij v domorodnem načinu (*C/C++*), spletnih tehnologijah (*HTML5*, *CSS3* in *JavaScript*) in hibridnem načinu, ki je kombinacija obeh. Za komunikacijo s senzorji naprave iz jezika *JavaScript* uporablja lastno knjižnico *Wormhole*, ki jo lahko uporabimo tudi za gradnjo domorodnega uporabniškega vmesnika. Tako lahko s spletnimi tehnologijami zgradimo domorodni uporabniški vmesnik in s tem dosežemo različni izgled aplikacije na posameznih platformah. Ogrodje podpira razvoj aplikacij za platforme: *Android*, *iOS*, *Windows Phone*, *Windows mobile*, *BlackBerry*, *Symbian*, *Java ME* in *Moblin*.

Za *MoSync* smo se odločili, ker podpira veliko platform in omogoča razvoj aplikacij v tako spletnih tehnologijah kot tudi v domorodnem načinu. Ogledali smo si tudi izvedbo domorodnega uporabniškega vmesnika, ki ga lahko zgradimo kar v jeziku *HTML5* ali *JavaScript*.

Poglavje 3

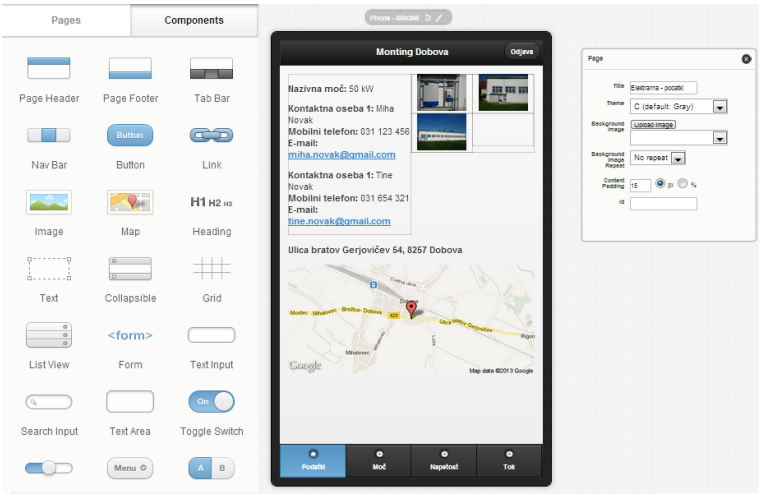
Uporabljene tehnologije

3.1 Codiqa - izdelava prototipov

Pred začetkom razvoja aplikacije se pogosto lotimo načrtovanja uporabniškega vmesnika. Z njim lažje ponazorimo videz aplikacije in njene funkcionalnosti naročniku in vsem udeležencem pri samem razvoju. Zaželeno je, da je izdelava hitra in enostavna. Ni potrebno, da vsebuje vse zahtevane funkcionalnosti, ampak je dovolj le njihova ponazoritev.

Za izdelavo načrta smo uporabili orodje *Codiqa* [3] [8]. Uporabniški vmesnik prikaže kot več zaslonskih mask, povezanih med seboj v delujoči aplikaciji. Končni izdelek je prototip mobilne aplikacije s prikazanimi funkcionalnostmi, a brez njihove realizacije. Orodje temelji na spletnih tehnologijah, saj ga upravljamo kar iz brskalnika. Njegova prednost je enostaven in uporabniku prijazen vmesnik (Slika 3.1). Za gradnike prototipa uporablja elemente knjižnice *jQuery Mobile* [5] [11]. Z njimi lahko kreiramo več zaslonskih mask, ki jih med seboj povežemo. Tako ustvarimo več dogodkovnih tokov, katerim v prototipu sledimo s pomikanjem po aplikaciji. Za uporabo orodja ne potrebujemo znanja o programiranju, temveč le osnovno poznavanje gradnikov in strukture mobilnih aplikacij. Izgradnje dodatnih funkcionalnosti ne ponuja, saj je njegov namen le njihov prikaz in ne realizacija. Izdelan prototip si lahko ogledamo preko spletne strani. Pri tem imamo možnost izbora med

različnimi velikosti ekranov, ki se ločijo na telefone in tablice. Omogoča tudi izvoz preko ogrodja *PhoneGap* [12] za razvoj medplatformnih mobilnih aplikacij, s pomočjo katerega lahko prototip kot aplikacijo namestimo na telefon.



Slika 3.1: Gradnja prototipa z orodjem Codiqa

3.2 MoSync

Za izdelavo medplatformne mobilne aplikacije lahko izbiramo med številnimi orodji. Ponujajo nam gradnjo aplikacije v spletnih tehnologijah, domorodni kodi in hibridnem načinu.

Odločili smo se za uporabo orodja *MoSync* [13]. Zaradi številnih prednosti v primerjavi z drugimi orodji nam omogoča hiter in fleksibilen razvoj. Izpostavimo lahko:

- podpora za 9 platform: *Android, Blackberry, iOS, Java ME MIDP, MeeGo, Moblin, Symbian, Windows Mobile, Windows Phone*;
- razvoj v spletnih tehnologijah, jeziku *C/C++* in hibridnem načinu;
- uporaba domorodnega uporabniškega vmesnika s spletnimi tehnologijami;

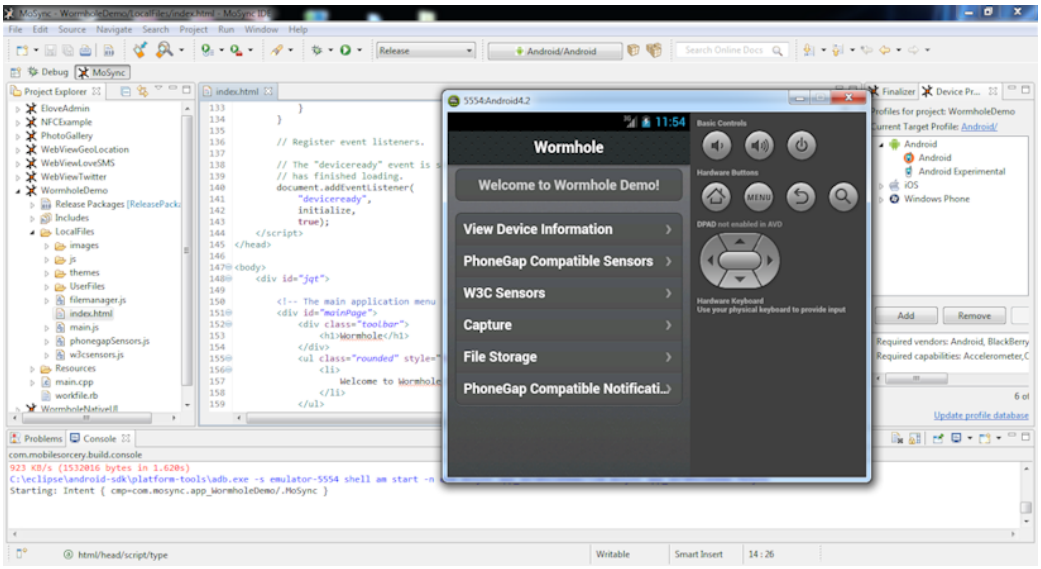
- razhroščevalnik kode *JavaScript*.

Slika 3.2: MoSync

Pri izdelavi aplikacije za pregled in administracijo sončnih elektrarn smo uporabili spletne tehnologije in domorodnega uporabniški vmesnik, kar nam je omogočilo hiter razvoj. V primeru težav zaradi omejitve funkcionalnosti pri spletnih tehnologijah so nam vedno ostale odprte možnosti za uporabo hibridnega načina, kjer poteka komunikacija med *C/C++* kodo in spletnimi tehnologijami. Na ta način nam orodje omogoča fleksibilnost, če bi v prihodnje prišlo po dodatnih zahtevah, ki jih v okviru spletnih tehnologij ne bi mogli realizirati.

Pričetek izdelave mobilne aplikacije je enostaven. Prvi korak je namestitev orodja *MoSync SDK*, ki ga najdemo na spletni strani MoSync [13]. V času pisanja aplikacije smo uporabljali verzijo *MoSync SDK 3.3*.

Orodje temelji na odprtokodni programski opremi *Eclipse* [16]. Ob zagonu programa imamo možnost uvoziti primere aplikacij, ki jih lahko zaženemo ter si ogledamo kodo (Slika 3.3).



Slika 3.3: Primer aplikacije v orodju MoSync

3.2.1 Testiranje aplikacije

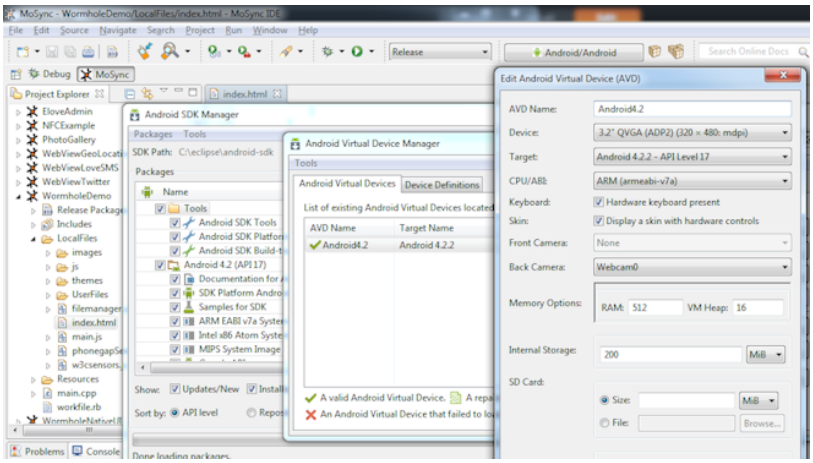
Ob razvoju nam orodje ponuja več načinov testiranja aplikacije:

- neposredno na mobilni napravi,
- v emulatorju posamezne mobilne platforme,
- v emulatorju *MoRE*, ki je del orodja *MoSync*.

Priporočeno je testiranje neposredno na mobilni napravi, ker so tam razmere najbolj podobne končnemu produkcijskemu okolju. Izvajanje aplikacije pri uporabi emulatorja je nekoliko počasnejše, vendar imamo možnost testiranja več platform hkrati, čeprav za njih nimamo mobilnih naprav. Pri razvoju aplikacije za pregled in administracijo sončnih elektrarn smo uporabili platforme *Android*, *Windows Phone* in *iOS*.

Android

Za zagon aplikacije v emulatorju platforme *Android* moramo namestiti *Android SDK* [17]. Po namestitvi izberemo v orodju *SDK Manager* verzijo platforme *Android*, ki jo želimo uporabljati. Priporočeno je izbrati tudi dodatek *Google USB Driver* za namestitev gonilnikov v primeru priklopa mobilne naprave preko vmesnika USB. V orodju *AVD Manager* lahko nato kreiramo navidezno napravo. Pozorni moramo biti na tip naprave, saj nam le-ta definira velikost ekrana. V primeru uporabe zunanje shrambe podatkov nastavimo velikost kartice za shranjevanje podatkov. Emulator omogoča tudi uporabo kamere računalnika za emulacijo kamere, ki se nahaja na mobilni napravi. Po kreiranju lahko napravo zaženemo znotraj orodja *MoSync* s klikom na gumb za zagon aplikacije. V primeru, da orodje ni našlo poti do namestitve *Android SDK*, se nam odpre okno, kamor jo vnesemo.

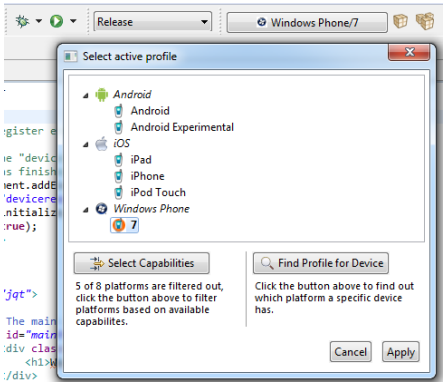


Slika 3.4: Kreiranje emulatorja za platformo Android

Windows Phone

Za emuliranje okolja *Windows Phone* moramo namestiti *Windows Phone SDK* [18]. Dodatne nastavitve emulatorja niso potrebne. Pred zagonom aplikacije le izberemo ciljno platformo *Windows Phone* iz seznama platform

(Slika 3.5). S klikom na zagon aplikacije se bo namestila in prikazala v emulatorju.



Slika 3.5: Izbor emulatorja za platformo Windows Phone

iOS

Podobno kot pri emulatorju za *Windows Phone* moramo tudi pri *iOS* izbrati ciljno platformo. Sam postopek namestitve emulatorja je nekoliko otežen, ker potrebujemo računalnik z nameščenim operacijskim sistemom *OS X*, kamor namestimo razvojno orodje *Xcode* [19]. Dodatna konfiguracija emulatorja ni potrebna.

MoRE

Pri razvoju imamo možnost uporabe notranjega emulatorja *MoRE* orodja *MoSync*. Vsebuje zmožnost interpretacije strojne kode navideznega stroja znotraj *MoSync*. Vezan ni na nobeno mobilno napravo, vendar ima omejitev pri prikazovanju domorodnega uporabniškega vmesnika. Ker se ta razlikuje pri vsaki platformi, ga *MoRE* ne omogoča. Dodatno mu lahko nastavljamo le velikost ekrana.

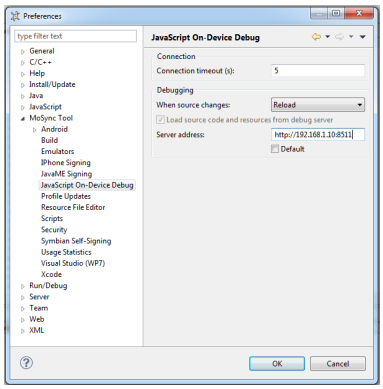
Slika 3.6: Emulator MoRE

3.2.2 Razhroščevanje aplikacije

Ob razvoju mobilnih aplikacij se pojavi težava pri odpravljanju napak. Ker se aplikacija ne izvaja v istem okolju, kot se je razvila, je težje slediti poteku izvajanja. Orodje *MoSync* ponuja vgrajeno rešitev, ki nam olajša delo. Vzpostavi most med izvajajočo se aplikacijo in razvojnim orodjem. Tako lahko spremljamo dogajanje znotraj aplikacije, tudi če se ta izvaja na drugi napravi ali pa znotraj emulatorja. Če želimo testirati aplikacijo na mobilni napravi, moramo paziti, da:

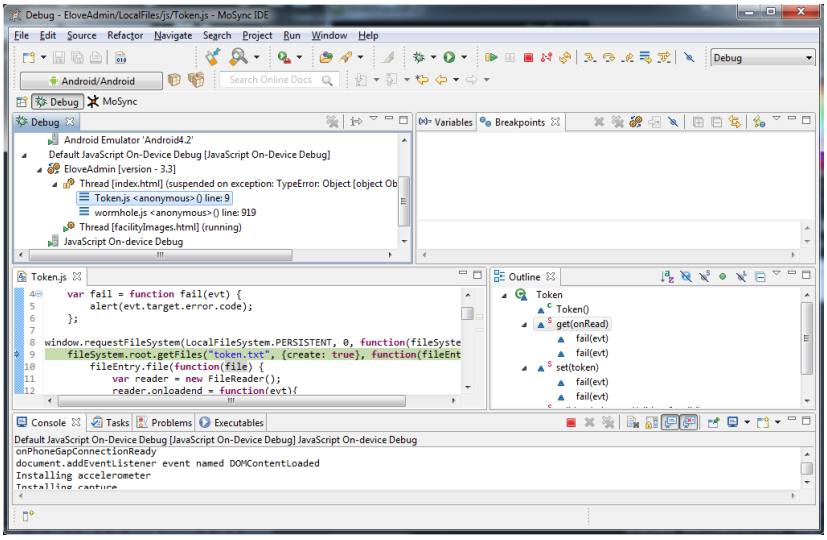
- je mobilna naprava povezana na isto omrežje kakor računalnik z razvojnim okoljem, kar lahko enostavno dosežemo z brezžično povezljivostjo na napravi;

- v orodju *MoSync* nastavimo naslov strežnika za razhroščevanje, ki ga najdemo znotraj orodja (Slika 3.7).



Slika 3.7: Nastavitev naslova razhroščevalnega strežnika

Aplikacijo moramo zagnati v razhroščevalni perspektivi (Slika 3.8), kjer imamo tudi možnost spremljanja izvajanja *C/C++* in *JavaScript* kode, pregled nad vrednostmi spremenljivk in postavljenimi prekinitvenimi točkami.

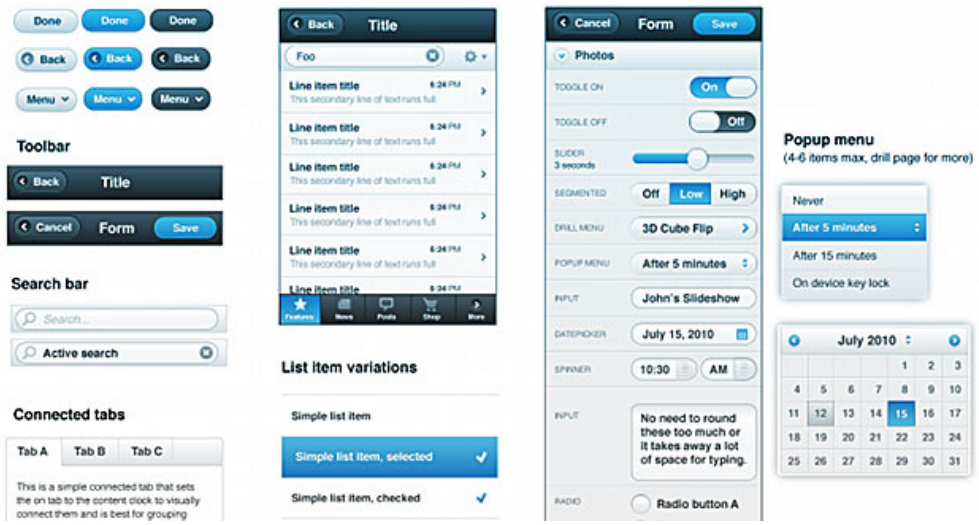


Slika 3.8: Razhroščevalna perspektiva

3.3 JavaScript in jQuery mobile

Pri spletnih tehnologijah lahko za izvajanje kode na strani klienta uporabimo različne tehnologije, kot so *JavaScript, Adobe Flash, Microsoft Silverlight ... JavaScript* je med klienti najbolj podprt in je zato tudi najbolj priljubljen. Uporabili smo ga za dinamično upravljanje aplikacije na strani klienta. Služi kot most med aplikacijo in mobilno napravo.

Ogrodje *jQuery mobile* [5] [11] temelji na jeziku *JavaScript*. Vsebuje vnaprej deklarirane gradnike (Slika 3.9), ki so optimizirani za prikaz na mobilnih napravah. Na ta način se olajša delo razvijalca, saj se lahko osredotoči na funkcionalnosti aplikacije. Ogrodje mu zagotavlja pravilen prikaz in delovanje uporabljenih gradnikov na različnih mobilnih napravah. Tako nam aplikacije ni potrebno prilagoditi za vsako napravo, ampak lahko uporabimo za vse enako kodo.



Slika 3.9: Gradniki ogrodja jQuery mobile

3.4 HTML5 in CSS3

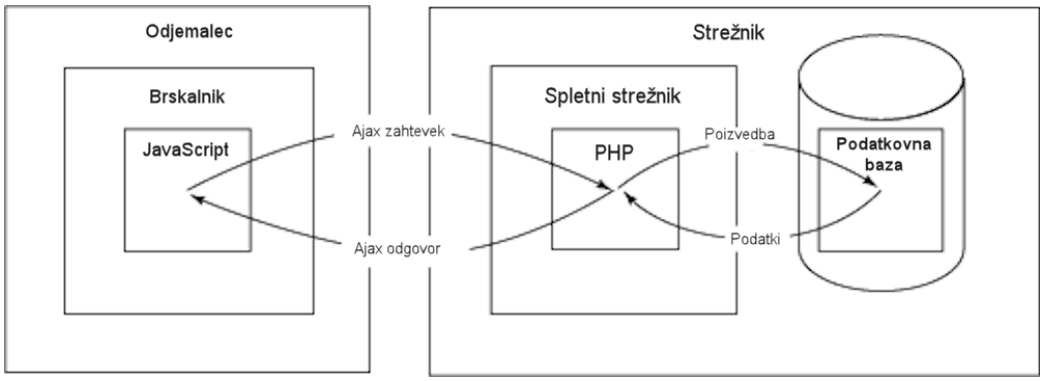
Za gradnjo uporabniške vmesnika smo uporabili *HTML5*. Z njim lahko enostavno določimo, katere elemente želimo prikazati in kakšno je njihovo zaporedje. Vnaprej definiramo posamezne zaslonske maske, ki jih nato le skrivamo in prikazujemo na zaslonu s pomočjo jezika *JavaScript*.

Sama postavitev elementov ni dovolj, saj jih želimo tudi oblikovati. To storimo z jezikom *CSS3*. Vsakemu elementu lahko določimo številne lastnosti, kot so dimenzije, barve, umestitev med ostale elemente, obrobe ... V mobilni aplikaciji ni bilo potrebno veliko oblikovanja, ker zanj v večini primerov poskrbi že ogrodje *jQuery mobile*.

3.5 Komunikacija s spletno aplikacijo

Mobilna aplikacija za svoje delovanje potrebuje internetno povezavo do spletne aplikacije. Podatki, ki jih prikazuje, se ves čas spreminjajo, zato lokalno hranjenje le-teh ni možno. Možnost, da bi ob zagonu prenesli vse podatke, ki jih potrebujemo, ni primerna. Na ta način ob uporabi aplikacije ne bi bila več potrebna internetna povezava, vendar je količina podatkov prevelika, da bi bilo to izvedljivo. Zato se prenesejo le tisti podatki, ki so potrebni, in le takrat, ko jih želimo prikazati.

Branje podatkov iz spletnega vira poteka preko asinhronskega klica v jeziku *JavaScript* (*AJAX*). Klic se izvede ločeno od niti za prikaz uporabniškega vmesnika in ne blokira same aplikacije. Na ta način lahko uporabniku prikažemo sporočilo o pridobivanju podatkov in mu ponudimo možnost za preklic. Preko vtičnika na spletni aplikaciji pošljemo zahtevek za želene podatke. Spletna aplikacija preveri, ali imamo do njih dostop, in jih pridobi iz baze podatkov. Kot odgovor jih vrne mobilni aplikaciji v obliki notacije *JavaScript* objekta *JSON* [22] (Slika 3.10).



Slika 3.10: Pridobivanje podatkov iz spletnega vira

Za komunikacijo nam spletna aplikacija preko svojega vtičnika ponuja več klicev. Vsi so dostopni preko protokola *HTTP* in metode *GET*. Ločimo jih na klice za: upravljanje z varnostnim žetonom, pridobivanje podatkov in spreminjanje podatkov.

Klica za upravljanje z varnostnim žetonom (Poglavje 5.4.1):

1. **Klic:** */user/token/request*
   Opis: pridobivanje izziva za varnostni žeton in validacija odziva
   Posredovani parametri: uporabniško ime v primeru pridobivanja izziva ali izziv in odziv v primeru validacije odziva
   Vrača: izziv ali varnostni žeton v primeru uspešne validacije. Ob napaki vrne sporočilo o napaki.

2. **Klic:** */user/token/validate*
   Opis: validacija že pridobljenega varnostnega žetona
   Posredovani parametri: varnostni žeton
   Vrača: sporočilo s potrditvijo (*OK*) ali zavrnitvijo (*Invalid token*) posredovanega žetona

Klici za pridobivanje podatkov sončne elektrarne:

1. **Klic:** */api/getFacility*
   Opis: pridobivanje seznama sončnih elektrarn in podatkov posamezne

sončne elektrarne
Posredovani parametri: varnostni žeton in identifikator elektrarne v primeru pridobivanja podatkov sončne elektrarne
Vrača: seznam sončnih elektrarn ali podatki o posamezni sončni elektrarni

2. **Klic:** */api/getImages*
Opis: pridobivanje slik posamezne sončne elektrarne
Posredovani parametri: varnostni žeton in identifikator elektrarne
Vrača: seznam slik z naslovi do njihovega spletnega vira

3. **Klic:** */api/getPower, /api/getVoltage, /api/getCurrent*
Opis: pridobivanje zgodovine meritev posamezne elektrarne
Posredovani parametri: varnostni žeton, identifikator elektrarne in časovni interval želene zgodovine
Vrača: seznam časovno urejenih točk z meritvami

Klica za spreminjanje podatkov sončne elektrarne:

1. **Klic:** */api/setContacts*
Opis: spreminjanje podatkov o kontaktnih osebah posamezne sončne elektrarne
Posredovani parametri: varnostni žeton, identifikator elektrarne in podatki o kontaktnih osebah
Vrača: napako ali sporočilo *OK* v primeru uspešne spremembe

2. **Klic:** */api/setImages*
Opis: dodajanje slike k sončni elektrarni
Posredovani parametri: varnostni žeton, identifikator elektrarne in slika
Vrača: napako ali sporočilo *OK* v primeru uspešnega prenosa

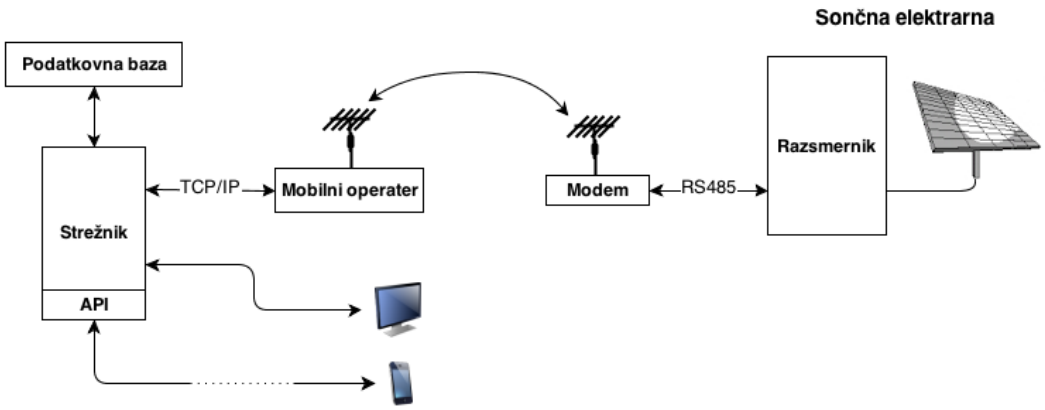
Poglavje 4

Sistem za nadzor sončnih elektrarn

Ena izmed glavnih prednosti mobilne aplikacije je dosegljivost nekaterih funkcionalnosti storitve, kjerkoli se nahajamo. V našem primeru so to funkcionalnosti spletne aplikacije za nadzor sončnih elektrarn.

Za lažje razumevanje si oglejmo grobo shemo povezovanja nadzornega sistema (Slika 4.1). Razsmernik je ključni element sončne elektrarne, na katerega so priklopljene sončne celice. Preko serijskega vmesnika *RS485* periodično odčitavamo različne vrednosti o samem delovanju. Podatke prenašamo preko mobilnega omrežja in jih beležimo v podatkovno bazo. Na strežniku jih obdelamo in prikažemo na spletnem vmesniku aplikacije. Uporabnik ima možnost spremljanja proizvodnje in obveščanje o izpadu posameznih enot elektrarne.

Z razširitvijo storitve na mobilne naprave želimo uporabniku ponuditi možnost pregleda nad proizvodnjo sončne elektrarne. Tako za ogled ne potrebuje več prisotnost računalnika, prav tako pa mu prihranimo čas v primerjavi z ogledom spletne aplikacije na mobilnem telefonu. Dodatno želimo realizirati upravljanje osnovnih podatkov o sončni elektrarni za upravljavce elektrarn.



Slika 4.1: Shema povezovanja sončne elektrarne

Poglavje 5

Pregledovalnik sončnih elektrarn

5.1 Namen aplikacije

Glavni cilj aplikacije je poenostaviti pregled nad proizvodnjo sončnih elektrarn. Upravljanje s sončnimi elektrarnami je možno preko spletne aplikacije, vendar želimo del funkcionalnosti zaradi hitrejše in enostavnejše uporabe podpreti na tudi na mobilni aplikaciji. Najbolj pogosto uporabljena funkcionalnost je pregled trenutne proizvodnje in njene krajše zgodovine, na kar se bomo tudi osredotočili pri razvoju mobilne aplikacije.

5.2 Analiza zahtev

Mobilna aplikacija mora vsebovati naslednje funkcionalnosti:

- prikaz osnovnih podatkov o elektrarni in kontaktnih osebah ter prikaz zemljevida z lokacijo elektrarne,
- možnost pregledovanja trenutne proizvodnje elektrarne in njene krajše zgodovine.

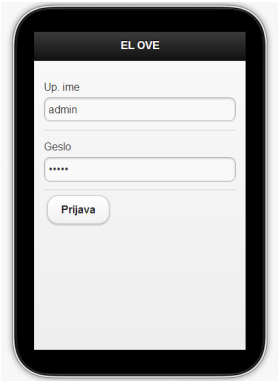
Uporabnik aplikacije se mora pred uporabo prijaviti v sistem, ki mu dodeli dostop do pregledovanja le tistih elektrarn, do katerih ima pravice vpogleda.

5.3 Izdelava prototipa

Pri razvoju prototipa smo uporabili orodje *Codiqa* [3] [8]. Prototip mobilne aplikacije smo razdelili na naslednje sklope:

- prijavno okno,
- seznam sončnih elektrarn,
- podroben pregled nad elektrarno.

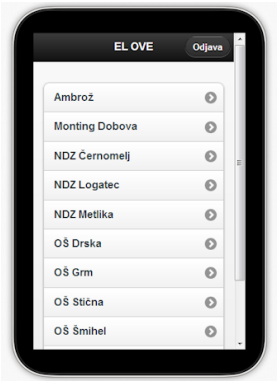
Prijavno okno (Slika 5.1) vsebuje polje za vnos uporabniškega imena in gesla ter gumb za prijavo v aplikacijo. Samo preverjanje veljavnosti prijavnih podatkov pri prototipu ni potrebno, zato aplikacija sprejme vsak vnos uporabniškega imena in gesla. V nadaljevanju imamo na vsakem koraku možnost odjave iz aplikacije s klikom na gumb "odjava". V tem primeru se vrnemo nazaj na prijavno okno.



Slika 5.1: Prijavno okno

Po uspešni prijavi se nam prikaže seznam sončnih elektrarn (Slika 5.2). Seznam je vnaprej pripravljen in se ne prilagaja glede na naš nivo dostopa

pri posameznih elektrarnah. S klikom na naziv elektrarne se odpre okno s podrobnostmi elektrarne.



Slika 5.2: Seznam sončnih elektrarn

Pri podrobnem pregledu nad elektrarno (Slika 5.3) izbiramo med različnimi grafi in informativnimi podatki o elektrarni. Ker gre za večjo količino informacij, jih je smiselno dodatno razdeliti. Želeni učinek smo dosegli z razporeditvijo po zavihkih. Zavihke z grafi si je priporočljivo ogledati v horizontalnem pogledu, saj je njihova os x veliko daljša od osi y.

Slika 5.3: Podroben pregled nad sončno elektrarno

Z orodjem *Codiqa* si olajšamo delo pri izdelavi prototipa, vendar ima tudi svoje pomanjkljivosti. V nekaterih primerih si želimo, da bi lahko elemente

oblikovno nekoliko dodelali z določitvijo dimenzij in poravnave. Na ta način bi poleg prikazanih funkcionalnosti dosegli tudi privlačnejši videz prototipa.

5.4 Izdelava aplikacije

Aplikacijo smo v celoti razvili v spletnih tehnologijah. Pri izdelavi smo uporabili knjižnice z namenom:

- *jQuery* [10]: komunikacija s spletnim vmesnikom,
- *jQuery mobile* [5] [11]: ogrodje aplikacije,
- *Highstock* [2] [9]: prikazovanje grafov pri pregledu proizvodnje elektrarne,
- *Hammer.js* [20]: upravljanje grafov s kretnjami,
- *MoSync wormhole* [13]: hranjene varnostnega žetona.

5.4.1 Varnost

Aplikacija omogoča nivojski dostop do pregleda sončnih elektrarn. Vsak uporabnik se mora pred uporabo aplikacije prijaviti s svojim uporabniškim imenom in geslom. Glede na njegov nivo dostopa se mu prikažejo podatki le tistih elektrarn, do katerih ima pravico vpogleda.

Vse podatke za prikazovanje aplikacija pridobi iz spletnega vmesnika. Ob vsakem zahtevku mora zato aplikacija posredovati tudi identifikacijo uporabnika. Le na ta način lahko spletni vmesnik vrne podatke, do katerih ima uporabnik dostop.

Ena izmed možnosti je posredovanje uporabniškega imena in gesla ob vsakem zahtevku. Uporabnik bi ju vnesel le ob začetku uporabe aplikacije, ta pa bi jih nato hranila za nadaljnjo uporabo. Opazimo lahko dve težavi. Prva nastane pri hranjenju uporabniških podatkov na mobilni napravi. Namreč iz varnostnih razlogov želimo, da je uporabniško ime in geslo prisotno le ob

prijavi brez nadaljnjega hranjenja. Vnosa prijavnih podatkov ob vsakem zahtevku za pridobivanje podatkov pa ne moremo smatrati za ustrezno rešitev, ker želimo le enkratno prijavo v aplikacijo ob začetku uporabe. Druga težava se pojavi pri pošiljanju zahtevka skupaj z uporabniškim imenom in geslom za pridobivanje podatkov iz spletnega vmesnika. Lahko se zgodi, da zahtevo prestreže tretja oseba in si s tem pridobi uporabnikove podatke.

Težavo s hranjenjem uporabniškega imena in gesla lahko rešimo z uvedbo varnostnega žetona. Ob prijavi iz spletnega vmesnika prejmemo žeton, s katerim se kasneje identificiramo spletnemu vmesniku. Na ta način ob vsakem zahtevku namesto uporabniškega imena in gesla pošljemo varnostni žeton. Ker žetone dodeljuje spletni vmesnik, jih lahko kadarkoli prekličemo ali izdamo zahtevek za novega. Če kdo prestreže našo komunikacijo, je njegov dostop omejen na veljavnost žetona in ne na veljavnost naših prijavnih podatkov. Žeton hranimo v mobilni aplikaciji, kar nam omogoča samodejno prijavo uporabnika tudi pri ponovni uporabi aplikacije. Uporabnik se mora tako prijaviti le ob prvi uporabi, kar poenostavi samo uporabo aplikacije.

Omenjena rešitev z varnostnim žetonom ima pomanjkljivost. Namreč za pridobitev žetona se moramo identificirati spletnemu vmesniku, kar pa storimo s svojim uporabniškim imenom in geslom. Če ju pošljemo spletnemu vmesniku, še vedno tvegamo, da ju bo nekdo prestregel. Z vpeljavo prijave z izzivom in odzivom [4] se lahko temu izognemo. Končna identifikacija uporabnika poteka na naslednji način (Slika 5.4):

1. Ob prvem zagonu aplikacije uporabnik vpiše svoje uporabniško ime in geslo.

2. Aplikacija pošlje spletnemu vmesniku zahtevek za izziv skupaj z uporabniškim imenom.

3. Spletni vmesnik vrne izziv, ki je sestavljen iz izvlečka zgoščevalne funkcije *SHA-1* [21] konkatenacije skritega niza in naključnega števila.

4. Aplikacija zgradi odziv s pomočjo izvlečka *SHA-1* konkatenacije izziva

in izvlečkom *SHA-1* gesla. Odziv skupaj z izzivom pošlje spletnemu vmesniku.

5. Spletni vmesnik lahko prav tako zgradi identičen odziv, saj ima izvleček uporabniškega gesla shranjenega v bazi podatkov. Prejeti in generirani odziv primerja in v primeru ujemanja vrne varnostni žeton. Žeton kreira iz izvlečka *SHA-1* konkatenacije skritega niza in naključnega števila.

6. Aplikacija se pri vsej nadaljnji komunikaciji s spletnim vmesnikom identificira z varnostnim žetonom.

Slika 5.4: Pridobivanje varnostnega žetona

Po končani prijavi uporabniških podatkov ne hranimo v aplikaciji, prav tako gesla v osnovni obliki ne pošiljamo po omrežju. S tem smo zagotovili osnovno varnost glede zasebnosti gesel. Za dodatno varnost podatkov bomo v prihodnosti vključili uporabo varne povezave preko protokola *HTTPS*. Za ta namen je potrebno pridobiti še podpisan *SSL certifikat* za spletni naslov vmesnika.

5.4.2 Struktura

Pri prvem zagonu aplikacije se uporabniku prikaže prijavno okno (Slika 5.5a). Prijava je potrebna samo prvič, ker si po pridobitvi varnostnega žetona le-tega shranimo. Ob vsakem ponovnem zagonu aplikacija preveri veljavnost obstoječega žetona in v primeru razveljavitve ponovno prikaže prijavno okno. Menjavo okna sprožimo s klicem funkcije znotraj knjižnice *jQuery mobile* [5] [11].

```
1  Token.get(function(token) {
2    if(token == null) {
3      $.mobile.changePage("login.html");
4    } else {
5      $.getJSON(API_URL+'/user/oken/validate/token='+token,
           function(data) {
6        if(data.status == 'OK')
7          $.mobile.changePage("facilities.html");
8        else
9          $.mobile.changePage("login.html");
10     });
11   }
12 });
```

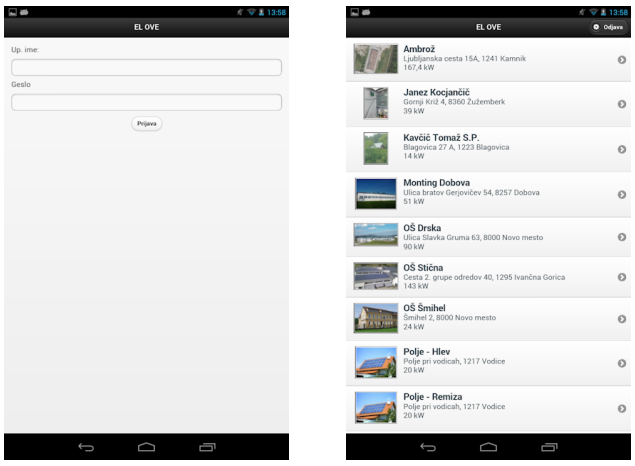
Primer kode 5.1: Preverjanje veljavnosti žetona in menjava okna v uporabniškem vmesniku

Za prikaz seznama elektrarn (Slika 5.2) smo uporabili gradnik *listview*, ki smo ga po pridobitvi preko zahteve *JSON* dinamično napolnili. Kreirali smo element seznama, katerega dvojnik smo nato za vsako elektrarno vstavili v seznam. Glede na podatke elektrarne smo mu spremenili ime, naslov, sliko in nazivno moč elektrarne. Pomembno je, da ustvarimo dvojnik, ker bi v nasprotnem primeru spreminjali podatke tudi za vse predhodne elemente.

```
1  var listItemHtml = $("#listItem").html();
2
3  Token.get(function(token) {
4    $.getJSON(API_URL+"/api/getFacility/token="+token,
       function(data) {
```

```
5     $("#list").empty();
6
7     $.each(data, function(i, item) {
8       var listItem = $("<li></li>");
9       listItem.html(listItemHtml);
10
11      $(listItem).find("a").attr("href", "facility.html?
            facilityID="+item.ID);
12      $(listItem).find("img").attr("src", item.image);
13      $(listItem).find(".title").html(item.title);
14      $(listItem).find(".address").html(item.address);
15      $(listItem).find(".power").html(item.power);
16
17      $("#list").append(listItem);
18      $("#list").listview("refresh");
19    });
20  });
21 });
```

Primer kode 5.2: Kreiranje seznama sončnih elektrarn

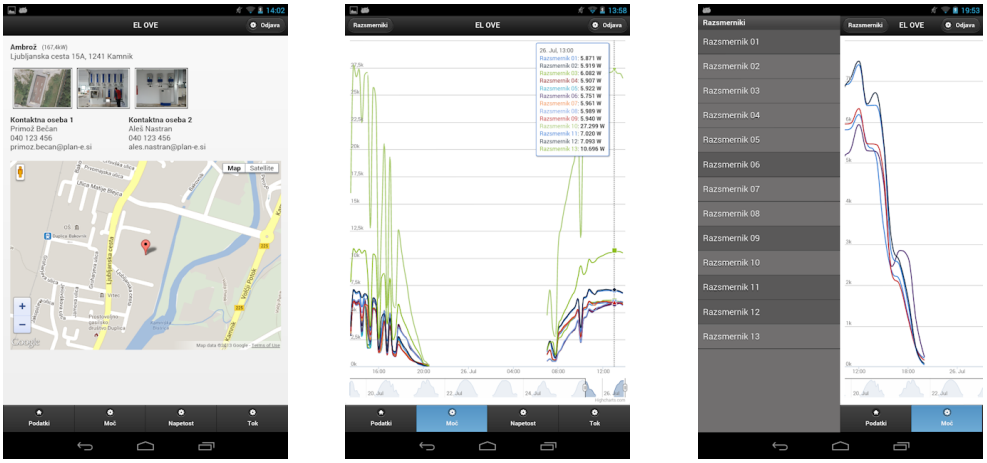


(a) Prijavno okno (b) Seznam elektrarn

Slika 5.5: Prijava in izbor elektrarne

Po izboru elektrarne se nam prikažejo informacije (Slika 5.6a). Poleg osnovnih podatkov si je mogoče ogledati slike elektrarne in pregledovati inte-

raktivni zemljevid, za katerega smo uporabili storitev *Google maps*. Zavihki nam ponujajo možnost ogleda grafov proizvodnje elektrarne (Slika 5.6b). Posamezni graf lahko pregledujemo s pomočjo navigacije na dnu prikaza in s pomočjo kretenj, ki jih vsakodnevno uporabljamo na mobilnih napravah. Ob kliku na izbrano točko grafa se nam prikaže podrobnejši opis o izmerjenih vrednostih. V primeru prevelikega števila razsmernikov v sončni elektrarni lahko postanejo grafi zaradi preveč informacij nepregledni. V tem primeru imamo možnost filtriranja posameznih razsmernikov (Slika 5.6c), kar nam omogoča prikaz samo tistih grafov, ki nas zanimajo.



(a) Informacije (b) Graf proizvodnje (c) Filtriranje grafa

Slika 5.6: Prikaz sončne elektrarne

5.4.3 Prikazovanje grafov

Izris

Pri izrisu grafov so se pojavile težave zaradi prevelike količine podatkov. Ker se meritve pri sončni elektrarni beležijo na intervalu nekaj minut, je bil izris grafa dovolj hiter samo ob ogledu nekaj urnega intervala. Če smo izbrali interval, ki se razteza čez več dni, pa je graf postal neodziven zaradi prevelike količine točk.

Rešitev je ponudila knjižnica *Highstock*, ki je različica knjižnice *Highcharts* [2] [9]. Za dani problem ponuja dve rešitvi:

- *grupiranje podatkov*, kjer knjižnica poskrbi, da združi večjo količino točk v eno, katero tudi izriše na grafu;
- *dinamično pridobivanje podatkov*, ki za vsako spremembo prikaza intervala na grafu pridobi sveže podatke iz strežnika.

Za svojo implementacijo smo izbrali drugo metodo. Ker ob vsaki spremembi prikaza pridobi nove podatke iz strežnika, jih ni potrebno v celoti posredovati že na začetku. Prikazovanje podatkov poteka na sledeči način:

1. Uporabnik v navigaciji grafa izbere želeni interval. Ob odprtju grafa je ta vnaprej določen.
2. Knjižnica *Highstock* pošlje zahtevek za pridobivanje podatkov na spletni vmesnik z informacijo o začetku in koncu intervala.
3. Spletni vmesnik pridobi iz baze podatkov vse točke na želenem intervalu. Grupira jih do te mere, da so še primerne za prikaz na mobilnih napravah. Na zahtevek odgovori z reduciranim številom točk.
4. Knjižnica *Highstock* prejme točke in jih prikaže na grafu.

Omenjeni način nam omogoča, da spletni vmesnik na vsak zahtevek odgovori s konstantnim številom točk. Odzivnost aplikacije bo enaka tako ob pregledu nekaj urnega kot tudi nekaj dnevnega intervala. Stopnja podrobnosti bo na manjšem intervalu večja, kar pa ne predstavlja težavo, ker pri večjem intervalu s prostim očesom niti ne moremo zaznati toliko podrobnosti zaradi prevelike količine podatkov.

Navigacija

Poleg navigacije na dnu prikaza posameznega grafa smo želeli omogočiti tudi navigacijo s kretnjami, ki je bolj domača uporabnikom mobilnih naprav.

Za pomik izbranega intervala smo uporabili kretnjo *swipe*, za spreminjanje velikosti intervala pa kretnjo *pinch-zoom*.

Osnovne kretnje na mobilnih naprav nam že omogoča knjižnica *jQuery mobile* [5] [11], vendar smo se odločili za uporabo naprednejše knjižnice za nadzor nad kretnjami *Hammer.js* [20]. Ker podpira tudi naprednejše kretnje, nam jih ni potrebno implementirati z uporabo detekcije dotika in pomika po zaslonu. Uporabimo lahko že obstoječe funkcije znotraj knjižnice.

```
1  Hammer(document.getElementById('container')})
2    .on("swipeleft swiperight", function(event) {
3      var x = event.gesture.touches[0].pageX;
4      var y = event.gesture.touches[0].pageY;
5      ...
6      // popravimo interval grafa
7      xAxis.setExtremes(xMin, xMax)
8    })
9    .on("pinch", function(event) {
10     var scale = event.gesture.scale;
11     ...
12     // popravimo interval grafa
13     xAxis.setExtremes(xMin, xMax)
14   });
```

Primer kode 5.3: Uporaba knjižnice Hammer.js

Ob vsaki izvedbi kretnje smo določili nove meje intervala za ogled grafa. To smo storili s pomočjo funkcije za nadzor teh mejnih vrednosti *xAxis.setExtremes(xMin, xMax)*.

Navigacijo s kretnjami smo omogočili samo na prostoru, kjer se prikazuje graf, kar je predstavljalo problem, ker se znotraj nje nahaja tudi navigacija knjižnice za prikazovanje grafa. Pri uporabi interne navigacije je potrebno navigacijo s kretnjami onemogočiti, saj sta si v nasprotnem primeru v konfliktu. Težavo smo rešili z detekcijo dotika znotraj interne navigacije. V primeru, da se je dotik zgodil s koordinatami, ki so v prostoru interne navigacije, smo kretnje onemogočili.

V času razvoja aplikacije je izšla nova verzija knjižnice *Highstock 1.3.2*,

ki je vsebovala tudi navigacijo s kretnjami. Pri spremembi intervala na grafu je potrebno paziti, kdaj se graf ponovno izriše, saj prepogosto izrisovanje vodi do neodzivnosti aplikacije. Zato smo sklepali, da bo aplikacija delovala bolje z uporabo internih kretenj knjižnice *Highstock*, saj ima boljši nadzor nad internim delovanjem kot kakšna druga knjižnica, ki bi *Highstock* upravljala od zunaj. Izkazalo se je, da zaradi prepogostega izrisovanja deluje še počasneje kot naša implementacija z uporabo knjižnice *Hammer.js*. Predvidevamo pa lahko, da se bo z nadaljnjim razvojem knjižnice to izboljšalo.

Filtracija prikaza

Vsak razsmernik sončne elektrarne predstavlja krivuljo na grafu. V primeru, da je krivulj veliko, postane graf nepregleden. S tem namenom smo implementirali rešitev za filtracijo razsmernikov. Ob pritisku na gumb z nazivom *Razsmerniki* se nam odpre meni, kjer je prikazan seznam vseh razsmernikov. Pri vsakem imamo s klikom nanj možnost omogočiti ali onemogočiti njegov prikaz na grafu.

Odpiranje in zapiranje menija je realizirano v celoti s *HTML5*, *CSS* in *JavaScript* s pomočjo knjižnice *jQuery mobile*. Meni prikazujemo, tako da spreminjamo *CSS* lastnost *margin-left*. V primeru, da je le-ta nastavljena na negativno vrednost širine bloka menija, se meni pomakne levo izven vidnega polja. S tem dosežemo efekt, da je zaprt. V nasprotnem primeru pa lahko nastavimo lastnost *maring-left* na nič, kar predstavlja odprt meni. Za lepšo uporabniško izkušnjo smo dodali efekt animacije, ki z manjšo zakasnitvijo postopoma spreminja lastnost *margin-left*. Pri odpiranju menija želimo vsebino aplikacije premakniti na desno, zato moramo tudi pri vsebini upoštevati ustrezno spremembo *margin-left*.

```
1 Menu.show = function(data) {
2   var duration = 300;
3   // odpiranje menija
4   $('#menu').animate({marginLeft: "0px"}, duration);
5   // pomik vsebine desno
6   $('
```

```
7       div[data-role="header"],
8       div[data-role="content"],
9       div[data-role="footer"]
10    ').animate({ marginLeft: _menuWidth }, duration);
11  };
12
13  Menu.hide = function(data) {
14    var duration = 300;
15    // zapiranje menija
16    $('#menu').animate({marginLeft: -1*_menuWidth}, duration)
          ;
17    // pomik vsebine na prvotno mesto
18    $('
19      div[data-role="header"],
20      div[data-role="content"],
21      div[data-role="footer"]
22    ').animate({ marginLeft: "0px" }, duration);
23  };
```

Primer kode 5.4: Prikaz menija za filtracijo razsmernikov

Vidnost krivulje nastavimo s funkcijo *series.setVisible(boolean)*, kar avtomatično sproži ponovni izris grafa. To zna predstavljati težavo pri hitrem klikanju uporabnika ob velikem številu razsmernikov, ker se ob vsakem kliku izriše celoten graf. V primeru, da bo to predstavljalo težavo, lahko vpeljemo zakasnitev pri izrisu grafa. Ob vsakem kliku za filtracijo nastavimo vidljivost s funkcijo *series.setVisible(boolean, false)*, kjer drugi argument pove knjižnici, da ne sproži ponovnega izrisa grafa. Ob kliku moramo tudi sprožiti zakasnjeno klicanje funkcije *chart.redraw()*, ki ponovno nariše graf. Tako bomo dosegli, da se graf izriše le enkrat na določen interval in ne ob vsakem kliku za filtracijo razsmernika. Dodatno je potrebno implementirati le zakasnjeno klicanje izrisa grafa, ki mora upoštevati samo en izris v določenem intervalu ne glede na število klikov.

Poglavje 6

Administracija sončnih elektrarn

6.1 Namen aplikacije

Ogledali smo si aplikacijo za pregled sončnih elektrarn, ki pa ne vsebuje funkcionalnosti za urejanje podatkov elektrarn. To pogosto predstavlja pomanjkljivost, saj tehnik nima možnosti urejanja podatkov na sami lokaciji elektrarne, kjer velikokrat ni prisotnosti računalnika. Urejanje podatkov preko mobilnega telefona je tudi hitrejše kot alternativna možnost priklopa prenosnega računalnika na mobilno omrežje in administriranje podatkov uporabniškega vmesnika na spletni strani.

6.2 Analiza zahtev

Aplikacija za administracijo sončnih elektrarn mora vsebovati naslednje funkcije:

- pregled seznama elektrarn in izbor posamezne elektrarne za nadaljnje urejanje;
- prikaz elektrarne na karti in možnost urejanja naslova lokacije;

- urejanje kontaktnih oseb posamezne elektrarne, ki so odgovorne za odpravo morebitnih težav;

- dodajanje slik, ki so zajete preko kamere na mobilni napravi, k elektrarni.

Varovanje podatkov mora biti zagotovljeno z nivojskim dostopom, kot je to zagotovljeno na spletni aplikaciji. Uporabniku se prikažejo samo elektrarne, pri katerih ima pravico urejanja, v primeru administratorja spletne aplikacije pa naj se prikaže možnost urejanja vseh elektrarn.

6.3 Izdelava prototipa

Podobno kot pri izdelavi prototipa za pregledovalnik sončnih elektrarn smo tudi tu uporabili orodje Codiqa [3] [8].

Aplikacijo smo razdelili na:

- prijavno okno,

- seznam sončnih elektrarn,

- urejanje sončne elektrarne, ki je razdeljeno na urejanje podatkov o sončni elektrarni, kontaktnih osebah in slikah.

Prijavno okno in seznam sončnih elektrarn sta enaka kot pri izdelavi prototipa za pregled nad sončnimi elektrarnami (Poglavje 5.3), saj vsebujeta povsem enake funkcionalnosti.

Urejanje sončnih elektrarn smo razdelili na več zavihkov (Slika 6.1) . V prvem zavihku urejamo podatke, kjer lahko nastavimo naziv in naslov. Prikaže se tudi karta z lokacijo elektrarne. Sledi zavihek z urejanjem kontaktnih oseb ter zavihek s slikami.

Slika 6.1: Urejanje sončne elektrarne

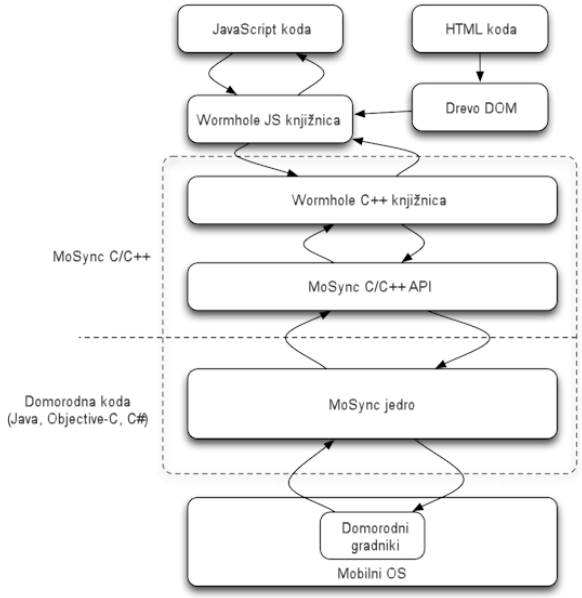
6.4 Uporaba domorodnega uporabniškega vmesnika

Pri izdelavi aplikacije za urejanje elektrarn smo uporabili del orodja Mosync [13], ki nam omogoča izgradnjo domorodnega uporabniškega vmesnika s pomočjo *HTML5* in *JavaScript* tehnologije. Uporabniški vmesnik je zato prilagojen posamezni platformi (Slika 6.2). To predstavlja dodatno prednost, saj je uporabnik že seznanjen z videzom in postavitvijo elementov aplikacije iz drugih aplikacij, ki jih uporablja na mobilni napravi.

(a) Android (b) Windows Phone (c) iOS

Slika 6.2: Primerjava domorodnega uporabniškega vmesnika

Vsaka platforma ima svoje elemente in različne načine njihovega prikazovanja v uporabniškem vmesniku. Naloga Mosync [13] je poenotenje osnovnih elementov in njihova prevedba iz HTML5 in JavaScript v domorodno obliko. To naredi na vsaki platformi drugače, vendar je celoten postopek programerju skrit. Na ta način omogoča pisanje uporabniškega vmesnika v enem jeziku, ki se nato prevede v jezik posamezne platforme.



Slika 6.3: Komunikacija z domorodnim uporabniškim vmesnikom

Na sliki 6.3 je razvidna pot komunikacije, ki se avtomatično opravlja ob uporabi domorodnega uporabniškega vmesnika. Pot navzdol predstavlja kreiranje elementov, pot navzgor pa nam omogoča povratno informacijo ob posameznih dogodkih, kot so pritisk na gumb, sprememba zavihka, nastavitev vrednosti polja ... Pri tem lahko opazimo, da je potrebnih veliko prehodov med nivoji, da pridemo od vrhnjega, kjer se izvaja *JavaScript*, do najnižjega z domorodno kodo. Samo delovanje domorodnih elementov je hitro, vendar se dolga pot odraža kot zakasnitev pri kreiranju elementov in odziv na njihove dogodke. Pojav lahko opazimo pri dinamičnem seznamu, ki smo ga uporabili za prikaz seznama elektrarn. Vsako vrstico z naslovom elektrarne

ustvarimo v *JavaScript* kodi in jo dodamo na seznam. Ob zagonu aplikacije lahko opazujemo zakasnitev, saj se vrstice seznama dodajajo ena za drugo, namesto da bi se prikazale vse naenkrat. Ko se celoten seznam kreira, je interakcija z njim hitra, dokler ne izberemo elektrarne, kar sproži povratno informacijo, ki jo obravnavamo v *JavaScript* kodi. Povratna informacija je hitrejša od kreiranja, vendar je zakasnitev še vedno prisotna.

6.5 Uporaba funkcionalnosti nižjega nivoja

6.5.1 Hranjenje podatkov

Pri izdelavi mobilnih aplikacij se pogosto pojavi zahteva po hranjenju podatkov na sami mobilni napravi. Poznamo več načinov hranjenja, vendar se po navadi odločimo za hranjenje v podatkovni bazi *SQLite* [14], ki omogoča enostavno hranjenje podatkov in njihovo poizvedovanje. Za poizvedbe uporabljamo jezik *SQL*, ki je razvijalcem intuitiven in enostaven za uporabo.

MoSync [13] omogoča uporabo *SQLite* neposredno iz jezika *JavaScript*, vendar z določeno pomanjkljivostjo. Njegova uporaba ni mogoča na platformi *Windows Phone*, saj tam domorodno ni podprt. Z nekaterimi knjižnicami ga lahko podpremo, vendar v osnovi *MoSync* to ni mogoče.

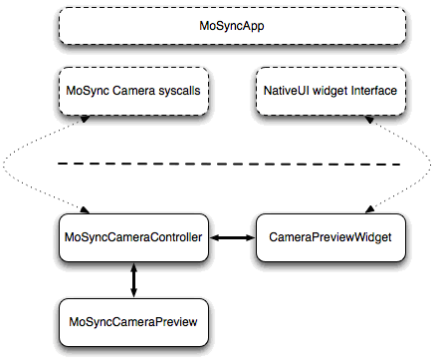
V aplikaciji za urejanje podatkov sončnih elektrarn nismo imeli potrebe po hranjenju podatkov v *SQLite*. Aplikacija hrani le žeton, preko katerega dinamično dostopa do vseh ostalih podatkov preko vmesnika na spletni strani. Iz tega razloga smo se odločili, da ne uporabimo *SQLite*, saj smo želeli podpreti delovanje aplikacije tudi na platformi *Windows Phone*.

Za hranjenje žetona je zadostovala navadna tekstovna datoteka, katere kreiranje in branje je podprto na vseh platformah, ki jih podpira *MoSync*.

6.5.2 Kamera

Kamera je eden izmed osnovnih senzorjev, ki jih najdemo v večini mobilnih naprav. V orodju *MoSync* [13] imamo možnost njene uporabe, vendar

nimamo dostopa do dodatnih funkcionalnosti, ki jih omogoča operacijski sistem pri zajemu slike. Ne moremo nastavljati filtrov, geolokacijskih podatkov, fokusa ... Omogoča nam le zajem slike, vse nastavitve pa so pred nastavljene v okolju *MoSync*. Kot prikazuje slika 6.4, se ob zahtevku za zajem slike zaženejo notranji programi, ki so implementirani glede na platformo zagona aplikacije. Ob uspešnem ali neuspešnem zajemu slike dobimo povratno informacijo v aplikaciji s podatkom lokacije slike ali pa opisom napake.



Slika 6.4: Implementacija zajema slike

6.6 Izdelava aplikacije

6.6.1 Upravljanje z žetonom

V aplikaciji potrebujemo mehanizem za upravljanje z žetonom. Z njim se namreč identificiramo spletnemu vmesniku, od katerega pridobimo podatke, do katerih imamo dostop. Mehanizem mora podpirati hranjenje žetona tudi po zaprtju aplikacije, saj bi se moral v nasprotnem primeru uporabnik vsakič znova prijaviti v sistem s svojim uporabniškim imenom in geslom. Obstajati pa mora tudi možnost, ki zagotavlja odstranitev žetona ob morebitni odjavi iz aplikacije. Več o samem varnostnem žetonu si lahko preberete v poglavju 5.4.1.

Za hranitev žetona smo izbrali hranjenje v tekstovni datoteki. Lahko bi

izbrali tudi podatkovno bazo *SQLite*, vendar v osnovi ni podprta na platformi *Windows Phone*. Ker je žeton sestavljen iz niza znakov, je hranjenje v datoteki povsem ustrezno.

Orodju *MoSync* [13] nam nudi nabor ukazov, s katerimi lahko preberemo in zapišemo datoteko (Primer kode 6.1). Za branje moramo slediti naslednjemu postopku:

- zahtevek za pridobitev instance datotečnega sistema,
- odpiranje datoteke znotraj datotečnega sistema z možnostjo nastavitve zastavice kreiranja ob neobstoju,
- inicializacija bralnika in branje datoteke.

```
1  window.requestFileSystem(LocalFileSystem.PERSISTENT, 0,
       function(fileSystem) {
2    fileSystem.root.getFile("token.txt", {create: true},
         function(fileEntry) {
3      fileEntry.file(function(file) {
4        var reader = new FileReader();
5        reader.onloadend = function(evt) {
6          var token = evt.target.result;
7          onRead(token);
8        };
9        reader.readAsText(file);
10     }, fail);
11   }, fail);
12 }, fail);
```

Primer kode 6.1: Branje datoteke

Ob vsakem klicanju funkcije imamo možnost obravnave napak, kar storimo z zadnjim argumentom. V našem primeru smo uporabili funkcijo *fail*, ki prejme sporočilo s podatkom o napaki. Zanimiva je tudi implementacija bralnika. Vsebino prebrane datoteke ne prejmemo pri ukazu *readAsText*, ampak moramo definirati postopek ob zaključku branja s funkcijo *onloadend*.

Opazimo lahko, da je interakcija z datotekami povsem asinhron postopek in moramo zato vnaprej določiti, katera funkcija se bo izvedla ob zaključku. V našem primeru smo definirali funkcijo *onRead*, ki prejme vsebino datoteke (žeton).

Pri pisanju datoteke je postopek podoben, le da ne zahtevamo inicializacijo bralnika, temveč pisalnika (Primer kode 6.2). Vsebino, ki jo želimo zapisati, posredujemo neposredno v funkcijo *write* pisalnika.

```
1 fileEntry.createWriter(function(writer) {
2   writer.write(token);
3 }, fail);
```

Primer kode 6.2: Pisanje datoteke

V določenih primerih moramo imeti možnost, da neposredno preverimo veljavnost žetona. V primeru neveljavnosti se žeton razveljavi in uporabniku se prikaže vmesnik za ponovno prijavo v aplikacijo. Preverjanje izvedemo enostavno z asinhronim zahtevkom na spletni vmesnik (Primer kode 6.3).

```
1 $.getJSON(API_URL+'/user/token/validate/token='+token,
    function(data) {
2   if(data.status == "OK")
3     onValid();
4   else
5     onInvalid();
6 });
```

Primer kode 6.3: Validacija žetona

6.6.2 Gradnja domorodnega uporabniškega vmesnika

HTML5

Pri izgradnji vmesnika v jeziku *HTML5* moramo upoštevati smernice, ki nam jih postavlja *MoSync*. Vse elemente, ki predstavljajo načrt za izgradnjo uporabniškega vmesnika, moramo postaviti v element z identifikator-

jem *NativeUI*. Za ta namen ob zagonu aplikacije uporabimo funkcijo *mosync.nativeui.initUI*.

Lastnosti posameznega elementa določa atribut s predpono *data-*, ki ji sledi ime lastnosti in vrednost. Vsakemu elementu moramo določiti njegov tip (*data-widgetType*). Najbolj pogosto uporabljamo tipe za definiranje gumba, vnosnega polja, teksta in slike. Z uporabo samo teh elementov ne moremo določiti prostorske umestitve elementa, zato imamo na voljo elemente za postavitev:

- *HorizontalLayout*: elementi si sledijo v horizontalnem zaporedju,
- *VerticalLayout*: elementi si sledijo v vertikalnem zaporedju,
- *RelativeLayout*: elementi si sledijo relativno med seboj.

V aplikaciji definiramo več zaslonov s pomočjo elementa *Screen*, nato pa med njimi preklapljamo glede na to, kje v aplikaciji se nahajamo. Če želimo uporabiti zavihke, jih lahko definiramo z elementom *TabScreen*, ki pa vsebuje več zaslonov *Screen*.

Nekateri elementi omogočajo proženje kode ob določenih dogodkih. Na ta način lahko v jeziku *HTML5* v kombinaciji z *JavaScript* določimo potek izvajanja aplikacije ob kliku na gumb. To storimo z atributom *data-onclick* in vsebino *JavaScript* kode, ki jo želimo izvesti ob prožitvi dogodka. V našem primeru sprožimo proces prijave.

```
1  <div id="NativeUI">
2    <div data-widgetType="Screen" id="login">
3      <div data-widgetType="VerticalLayout"
4        data-width="FILL_AVAILABLE_SPACE"
5        data-height="FILL_AVAILABLE_SPACE"
6        data-paddingLeft="40"
7        data-paddingRight="40"
8        data-paddingTop="40"
9        data-paddingBottom="40">
10       <div data-widgetType="EditBox"
```

```
11          id="username"
12          data-width="FILL_AVAILABLE_SPACE"
13          data-placeholder="Uporabnisko ime"
14          data-fontSize="30">
15        </div>
16        <div data-widgetType="EditBox"
17          id="password"
18          data-width="FILL_AVAILABLE_SPACE"
19          data-placeholder="Geslo"
20          data-editMode="password"
21          data-fontSize="30">
22        </div>
23        <div data-widgetType="HorizontalLayout"
24          data-width="FILL_AVAILABLE_SPACE"
25          data-height="WRAP_CONTENT"
26          data-childHorizontalAlignment="center"
27          data-paddingTop="5">
28          <div data-widgetType="Button"
29            id="loginButton"
30            data-text="Prijava"
31            data-fontSize="30"
32            data-onclick="Login.login(); ">
33          </div>
34        </div>
35      </div>
36    </div>
37  </div>
```

Primer kode 6.4: Domorodni uporabniški vmesnik v jeziku HTML5

JavaScript

Gradnja domorodnega uporabniškega vmesnika z *JavaScript* se od *HTML5* razlikuje v tem, da ga lahko dinamično spreminjamo tudi med izvajanjem aplikacije. Pri uporabi *HTML5* se namreč uporabniški vmesnik v celoti zgradi že ob zagonu aplikacije.

Manipulacijo nad elementi uporabniškega vmesnika lahko izvajamo preko

različnih metod, ki se nahajajo v objektu *mosync.nativeui.NativeWidgetElement*. Pogosto uporabljamo naslednje metode:

- *setProperty*: omogoča nastavljanje raznih vizualnih in funkcionalnih lastnosti,
- *getProperty*: pridobivanje vrednosti določene lastnosti,
- *addChild, addTo*: gradnja drevesne strukture elementov.

Do elementov dostopamo preko njihovega identifikatorja in funkcijo *document.getNativeElementById*, ki vrne objekt elementa *mosync.nativeui.NativeWidgetElement*.

V primeru kode 6.5 je prikazana dinamična gradnja seznama elektrarn v domorodnem uporabniškem vmesniku. Element z identifikatorjem *facilitiesList* že obstaja in je tipa *ListView*. Po pridobitvi žetona iz spletnega vmesnika pridobimo seznam elektrarn, do katerih imamo dostop. Za vsako elektrarno nato ustvarimo nov element tipa *ListViewItem*, ki ga pripnemo seznamu. Ob kreiranju elementa se ustvari ekvivalentni element v domorodni kodi. To je nekoliko dolgotrajnejši postopek, zato se elementi na seznamu ne prikažejo hkrati, temveč eden za drugim z manjšim zamikom. Zakasnitev ni moteča, vendar je opazna s prostim očesom. Na koncu določimo potek izvajanja aplikacije ob kliku na element v seznamu, saj v tem primeru želimo prikaz želene elektrarne. To storimo z metodo *addEventListener* in dogodkom *ItemClicked*.

```
1 var list = document.getNativeElementById("facilitiesList");
2 Token.get(function(token) {
3   $.getJSON(API_URL+'/api/getFacility/token='+token,
      function(data) {
4     $.each(data, function(key, facility) {
5       var item = mosync.nativeui.create("ListViewItem", "
          item"+key, {
6         text: facility.title,
7         fontSize: 40
```

```
8         });
9         item.addTo("facilitiesList");
10      });
11      list.addEventListener("ItemClicked", function(item,
            sParam, itemIndex) {
12        var facilityID = data[itemIndex].ID;
13        Facility.show(facilityID);
14      });
15    });
16 });
```

Primer kode 6.5: Domorodni uporabniški vmesnik v jeziku JavaScript

6.6.3 Prikazovalnik stanja obdelave podatkov

Pri posameznih delih aplikacije moramo za prikaz vsebine pridobiti podatke iz spletnega vmesnika. To pa predstavlja določeno zakasnitev, kar daje vtis neodzivnosti aplikacije. Uporabnika moramo zato obvestiti, da poteka postopek pridobivanja podatkov.

V primeru kode 6.6 smo implementirali prikaz obvestila. Sestavljeno je iz domorodnega elementa tipa *ModalDialog*, ki na zaslonu prikaže okno, in elementa *ActivityIndicator*, ki prikazuje vrteči se krog za ponazoritev obdelave podatkov. Sama struktura obvestila je definirana v jeziku *HTML5*, katero prikažemo ali skrijemo dinamično s pomočjo funkcij *Loader.show* in *Loader.hide*. Zaradi posebnosti pri elementu *ModalDialog* moramo za prikazovanje uporabiti metodi *showDialog* ter *hideDialog* in ne metodo *show* kot pri ostalih elementih.

```
1 <script>
2 Loader.show = function(title) {
3   var loader = document.getNativeElementById("loader");
4   if(typeof(title) != "undefined") {
5     loader.setProperty("title", title);
6   }
7   loader.showDialog();
8 };
```

```
9
10 Loader.hide = function() {
11   document.getNativeElementById("loader").hideDialog();
12 };
13 </script>
14
15 <div id="NativeUI">
16   <div data-widgetType="ModalDialog"
17     id="loader"
18     data-title="Loading ...">
19     <div data-widgetType="VerticalLayout"
20       data-width="FILL_AVAILABLE_SPACE"
21       data-height="FILL_AVAILABLE_SPACE"
22       data-paddingTop="20"
23       data-paddingBottom="20"
24       data-childHorizontalAlignment="center">
25       <div data-widgetType="ActivityIndicator"
26         data-inProgress="true">
27       </div>
28     </div>
29   </div>
30 </div>
```

Primer kode 6.6: Prikaz obvestila

6.6.4 Upravljanje z vnosnimi polji

V aplikaciji ima uporabnik možnost spremembe nekaterih podatkov. To mu zagotovimo preko vnosnih polj, ki so tipa *EditBox*. Po končanem vnosu s klikom na gumb sproži posredovanje podatkov na spletni vmesnik, ki jih nato shrani v bazo podatkov.

V primeru kode 6.7 je prikazano pridobivanje vsebine vnosnih polj. Iz njih pridobimo ime, telefon ter e-poštni naslov kontaktne osebe. To lahko storimo z metodo *getProperty* za branje vrednosti z imenom *text*. Ob tem se sproži komunikacija z domorodnim elementom, ki predstavlja vnosno polje. Zaradi zakasnitve se klic izvede asinhrono. V našem primeru to pomeni, da moramo

klicu funkcije posredovati tudi kodo, ki se bo izvedla ob uspešnem branju vrednosti. Pisanje kode tako postane dosti bolj nepregledno, saj moramo za pridobitev treh vrednosti ugnezditi več delov kode. Problem postane še toliko hujši, če imamo več vnosnih polj, kar je pogosto pri daljših obrazcih.

```
1  document.getNativeElementById("contact1Title").getProperty(
       "text", function(name, value) {
2    var contact1Title = value;
3    document.getNativeElementById("contact1Telephone").
         getProperty("text", function(name, value) {
4      var contact1Telephone = value;
5      document.getNativeElementById("contact1Email").
           getProperty("text", function(name, value) {
6        var contact1Email = value;
7        // save data
8        ...
9      });
10   });
11 });
```

Primer kode 6.7: Pridobivanje vsebine vnosnih polj

6.6.5 Uporaba elementa za prikaz spletne vsebine

V večini primerov želimo uporabiti domorodne elemente, saj je njihovo izvajanje hitrejše. Obstajajo pa izjeme, pri katerih želimo prikazati vsebino v spletni obliki z jezikom *HTML5* in *CSS*.

Pri prikazovanju slik posamezne elektrarne smo naleteli na težavo pri uporabi domorodnega elementa tipa *Image*. Sliko smo prikazali iz spletnega vira s pomočjo funkcije *mosync.resource.loadRemoteImage*. Problem se je pojavil v primeru, ko slika na spletnem viru ni obstajala ali pa je ime datoteke vsebovalo znake, ki niso v standardni tabeli znakov ASCIIčiteascii. Prišlo je do napake znotraj aplikacije, kar je posledično pomenilo nasilno ustavitev aplikacije. Prav tako ni bilo možnosti uporabe mehanizma za obvladovanje napak, saj v tem primeru enostavno ne bi prikazali težavnih slik.

Odločili smo se za uporabo domorodnega elementa *WebView* (Primer kode 6.8, v kateri smo prikazali slike elektrarne). Ustvarili smo datoteko *facilityImages.html*, ki smo jo prikazali znotraj elementa s pomočjo atributa *data-url*.

```
<div data-widgetType="WebView"
  id="imagesWebView"
  data-width="FILL_AVAILABLE_SPACE"
  data-height="FILL_AVAILABLE_SPACE"
  data-url="facilityImages.html">
</div>
```

Primer kode 6.8: Uporaba elementa WebView

Sam prikaz datoteke v elementu ni dovolj, saj želimo slike iz točno določene elektrarne. Kodo za prikaz slik moramo izvesti znotraj elementa, ker imamo samo tam dostop do njegove vsebine (Primer kode 6.9). To lahko storimo s funkcijo *mosync.nativeui.callJS*. Posredovati ji moramo ročico elementa in kodo, ki jo želimo izvesti. Ročico pridobimo s funkcijo *mosync.nativeui.getNativeHandleById* in identifikatorjem elementa. V našem primeru se bo znotraj elementa izvedla koda za prikaz slik, ki potrebuje identifikator elektrarne in žeton za spletni vmesnik.

```
Token.get(function(token) {
  var imagesHandle = mosync.nativeui.getNativeHandleById("
     imagesWebView");
  mosync.nativeui.callJS(imagesHandle, "Facility.showImages
     ("+facilityID+",'"+token+"')");
});
```

Primer kode 6.9: Izvajanje kode znotraj elementa WebView

6.6.6 Zajem in pošiljanje slike

Ob pregledu slik posamezne elektrarne ima uporabnik možnost zajema slike preko kamere na mobilni napravi, ki jo nato doda k preostalim. Postopek poteka v treh korakih:

- zajem slike s pomočjo programa v platformi *MoSync*,
- predogled in potrditev zajete slike,
- pošiljanje slike na spletni vmesnik.

Zajem slike je enostaven, saj zanj poskrbi program znotraj platforme *MoSync*. Celoten postopek sprožimo s funkcijo *navigator.device.capture.captureImage*, ki ji posredujemo kodo za izvajanje ob uspešnem zajetju in napaki (Primer kode 6.10). Za predogled slike prav tako poskrbi program platforme *MoSync*. V primeru, da se uporabnik strinja, se izvede posredovana koda, ki prejme lokacijo zajete slike. V primeru napake pa uporabniku prikažemo obvestilo s številko napake.

```
1 navigator.device.capture.captureImage(
2   function(mediaFiles) {
3     var filePath = mediaFiles[0].fullPath;
4     Facility.uploadPicture(facilityID, filePath);
5   },
6   function(error) {
7     alert("Zajem slike ni mogoc. Napaka: " + error.code);
8   }
9 );
```

Primer kode 6.10: Zajem slike

Po pridobitvi slike jo je potrebno poslati na spletni vmesnik, ki jo obdela in shrani v bazo podatkov (Primer kode 6.11). Za ta namen kreiramo objekt razreda *FileTransfer*, z njegovo metodo *upload* pa sliko nato posredujemo spletnemu vmesniku. Določiti moramo, kakšno datoteko pošiljamo, kar storimo z novim objektom razreda *FileUploadOptions*. Po uspešnem pošiljanju ponovno prikažemo vse slike elektrarne. Na ta način jih osvežimo, da se prikaže tudi dodana slika. To storimo s funkcijo *mosync.nativeui.callJS*, saj se mora koda izvesti znotraj elementa tipa *WebView*, kjer so prikazane slike. V primeru napake uporabniku prikažemo obvestilo s številko napake.

```
1  var options = new FileUploadOptions();
2  options.fileKey = "image";
3  options.fileName = filePath.substr(filePath.lastIndexOf('/'
       ) + 1);
4  options.mimeType = "image/jpeg";
5  options.params = null;
6
7  var transfer = new FileTransfer();
8  Token.get(function(token) {
9    transfer.upload(
10     "file://" + filePath,
11     API_URL+"/api/setImages/facilityID="+facilityID+"/token
           ="+token,
12     function(result) {
13       var imagesHandle = mosync.nativeui.
             getNativeHandleById("imagesWebView");
14       mosync.nativeui.callJS(imagesHandle, "Facility.
             showImages("+facilityID+",'"+token+"')");
15       alert("Slika je bila uspesno poslana");
16     },
17     function(error) {
18       alert("Napaka pri posiljanju slike: " + error.code);
19     },
20     options);
21 });
```

Primer kode 6.11: Pošiljanje slike

6.6.7 Pomanjkljivosti

Pri izdelavi aplikacije smo se srečali z določenimi težavami. Pogosto so se pojavljale napake, o katerih razhroščevalnik ni podal nobene koristne informacije, tudi pri uporabi kode iz primera na uradni spletni strani. Kot primer lahko izpostavimo, da ogrodje *MoSync* omogoča prikazovanje slik iz spletnega vira v domorodnem uporabniškem v vmesniku, vendar se ob vsaki morebitni napaki pri pridobivanju slike celotna aplikacija zapre zaradi neobravnavane

napake. Napaka se pojavi, če slika ne obstaja na spletnem naslovu ali pa v imenu vsebuje šumnike. Pravilen način bi bila možnost obravnave napake in ne zaprtje celotne aplikacije.

Pojavljale so se tudi neskladnosti med posameznimi mobilnimi platformami. Najbolj kritična je obravnava vnosnega polja za geslo pri domorodnem uporabniškem vmesniku na platformi *Windows Phone*. Polje se prikaže kot navadno vnosno polje in zato ob vnosu gesla ni maskirano. Na ostalih platformah pa deluje kot pričakovano.

Pogrešali smo tudi določene osnovne funkcionalnosti, ki bi jih pričakovali pri gradnji mobilne aplikacije. Tako ni bilo mogoče kreirati menija v okviru *HTML5* in *JavaScript*. Ker smo v meni želeli dodati samo gumb za odjavo, to ni bila tako velika težava, saj smo ga lahko umestili v uporabniški vmesnik.

Ker je gradnja domorodnega uporabniškega vmesnika s pomočjo *HTML5* in *JavaScript* še v zgodnji fazi razvoja, lahko pričakujemo popravke pomanjkljivosti v prihodnjih izdajah ogrodja *MoSync*.

Poglavje 7

Sklepne ugotovitve

Ogrodje *MoSync* nam omogoča hitro in relativno enostavno izdelavo mobilnih aplikacij, ki so podprte na vseh prevladujočih mobilnih platformah. S pomočjo *HTML5* in *JavaScript* lahko zgradimo aplikacijo izključno v spletnih tehnologijah. Kljub poenostavitvi razvoja imamo še vedno možnost uporabe večine senzorjev na mobilni napravi. Ogrodje vzpostavi prehod med aplikacijo in strojno opremo mobilne naprave, kar nam omogoča njeno uporabo. Zaradi spletnih tehnologij nismo prikrajšani za domorodni uporabniški vmesnik, saj ga lahko zgradimo kar s pomočjo *HTML5* in *JavaScript*. Ogrodje poskrbi, da se razvijalcu ni potrebno ukvarjati, na kakšen način so stvari implementirane na posamezni platformi, in mu omogoča, da se lahko osredotoči na funkcionalnosti aplikacije.

Pri uporabi ogrodja *MoSync* se moramo zavedati določenih pomanjkljivosti. Ob gradnji mobilne aplikacije smo imeli občutek, da je veliko stvari potrebno še dodelati. Nekatere funkcionalnosti, kot je gradnja domorodnega uporabniškega vmesnika iz spletnih tehnologij, so še v začetnih fazah razvoja in je zato še veliko prostora za izboljšave.

V nasprotju s pričakovanji je izvajanje domorodnega uporabniškega vmesnika počasneje kot pri aplikaciji, napisani v domorodni kodi. Razlog je komunikacija med domorodnimi elementi in kodo *JavaScript*. Dokler komunikacija ni potrebna, pa je izvajanje hitro in brez zaznavnih upočasnitev.

Kot primer lahko izpostavimo seznam, ki ga uporabljamo za izbor sončne elektrarne. Zakasnitev je opazna pri kreiranju seznama in odzivu pri izboru elektrarne. V obeh primerih steče komunikacija s kodo *JavaScript*. Pri pregledovanju seznama pa komunikacija ni potrebna in je zato delovanje tekoče.

Pojavljajo se tudi nesoglasja pri prikazovanju vsebine (vnosno polje za geslo) na različnih platformah. Ker uporabljamo ogrodje za enostavnejši razvoj medplatformnih aplikacij, pričakujemo, da se nam ne bo potrebno ukvarjati z različnim delovanjem aplikacije glede na izbrano platformo.

Kljub težavam, na katere smo naleteli ob izdelavi aplikacije, nam ogrodje predstavlja alternativni razvoj, s katerim je gradnja aplikacije hitrejša in enostavnejša. Avtorji ogrodja z vsako novo izdajo odpravljajo pomanjkljivosti in dodajajo potrebne funkcionalnosti. V prihodnosti lahko pričakujemo izboljšavo orodja in še enostavnejšo izdelavo medplatformnih mobilnih aplikacij.

Literatura

[1] S. Allen, V. Graupera, L. Lundrigan, *Pro Smartphone Cross-Platform Development*, New York: Springer Science, 2010, pogl. 2.

[2] J. Kuan, *Learning Highcharts*, Birmingham: Packt Publishing, 2012.

[3] T. A. Peters, L. Bell, *The handheld library: mobile technology and the librarian*, California: ABC-CLIO, 2013, str. 89–90

[4] M. Peyravian, N. Zunic, "Methods for protecting password transmission", *Computers & Security*, št. 5, zv. 19, str. 466–469, 2000.

[5] J. Reid, *jQuery Mobile*, Sebastopol: O'Reilly Media, 2011.

[6] (2013) Titanium mobile. Dostopno na: http://www.appcelerator.com/platform/titanium-platform/

[7] (2013) RhoMobile. Dostopno na: http://www.motorolasolutions.com/US-EN/Business+Product+and+Services/Software+and+Applications/RhoMobile+Suite

[8] (2013) Codiqa. Dostopno na: https://codiqa.com

[9] (2013) Highcharts, Highstock. Dostopno na: http://www.highcharts.com

[10] (2013) jQuery. Dostopno na: http://jquery.com

[11] (2013) jQuery. Dostopno na:
http://jquerymobile.com

[12] (2013) PhoneGap. Dostopno na:
http://phonegap.com

[13] (2013) MoSnyc. Dostopno na:
http://www.mosync.com

[14] (2013) SQLite. Dostopno na:
http://www.sqlite.org

[15] (2013) Tabela znakov ASCII. Dostopno na:
http://www.asciitable.com

[16] (2013) Eclipse. Dostopno na:
http://www.eclipse.org

[17] (2013) Android SDK. Dostopno na:
http://developer.android.com/sdk/

[18] (2013) Windows Phone SDK. Dostopno na:
http://developer.windowsphone.com

[19] (2013) XCode. Dostopno na:
https://developer.apple.com/xcode/

[20] (2013) Hammer.js. Dostopno na:
http://eightmedia.github.io/hammer.js/

[21] (2013) SHA-1. Dostopno na:
http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf

[22] (2013) JSON. Dostopno na:
http://tools.ietf.org/html/rfc4627