Mika Karenius

PhoneGap-ohjelmistokehyksen hyödyntäminen alustariippumattomassa mobiilisovelluskehityksessä

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Mediatekniikan koulutusohjelma
Insinöörityö
23.10.2012

Metropolia

Tiivistelmä

Tekijä Otsikko Sivumäärä Aika	Mika Karenus PhoneGap-ohjelmistokehyksen hyödyntäminen alustariippumattomassa mobiilisovelluskehityksessä 53 sivua 23.10.2012
Tutkinto	insinööri (AMK)
Koulutusohjelma	mediatekniikka
Suuntautumisvaihtoehto	digitaalinen media
Ohjaaja	viestintäpäällikkö Willy Toiviainen yliopettaja Petri Vesikivi

Insinöörityössä selvitettiin PhoneGap-ohjelmistokehyksen hyödyntämistä alustariippumattomassa mobiilisovelluskehityksessä ja sitä, millaisia haasteita sen käyttöönotto luo sovelluskehittäjälle. Insinöörityö toteutettiin Suomen Punaisen Ristin Veripalvelulle.

Insinöörityössä käytiin läpi PhoneGapin arkkitehtuuri, jonka käyttöliittymäkerroksena toimii WebKit-komponentti. WebKit-komponenttia ohjaa PhoneGapin JavaScript-moottori, kun taas laitteen natiivitoiminnallisuuksia ja -ominaisuuksia ohjaa PhoneGapin natiivimoottori. Työssä myös tutkittiin PhoneGapin tarjoamat ohjelmointirajapinnat ja se, miten niitä hyödynnetään mobiiliohjelmoinnissa. Insinöörityössä myös selvitettiin, miten PhoneGapiä voidaan laajentaa itsetehtyjen liitännäisien avulla.

Työn lopputuloksena syntyi iOS-alustalle toteutettu mobiiliverkkosovellus, jonka avulla sovelluksen käyttäjät saataisiin käymään verenluovutuksessa nykyistä useammin muistuttamalla heitä uudesta verenluovutusmahdollisuudesta. Sovellus toimii käyttäjälle muistutustoiminnallisuuden lisäksi myös informaatiokanavana. Sovellus julkaistaan myöhemmin sovelluskaupassa. Alustariippumattoman toteutuksen ansiosta sovellus voidaan myöhemmin helposti toteuttaa myös usealle eri mobiilialustalle.

Kokemukset PhoneGapin hyödyntämisestä sovelluskehityksessä osoittivat sen olevan erityisen hyödyllinen sen mahdollistaman paketoimisen ja natiivisovelluskehitystä huomattavasti loivemman oppimiskäyrän takia.

Avainsanat	PhoneGap, HTML5, JavaScript, alustariippumaton, sovelluskehitys, mobiili

Metropolia

Abstract

Author Title Number of Pages Date	Mika Karenius Exploiting the PhoneGap-framework in cross-platform mobile development 53 pages 23rd October 2012
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructor	Willy Toiviainen, Head of Communications Petri Vesikivi, Principal Lecturer

The thesis studied the benefits of the PhoneGap framework in cross-platform mobile development, and the challenges it creates for the application developer. The application created in the thesis was carried out for Finnish Red Cross Blood Service.

The thesis analysed the PhoneGap architecture, where the WebKit component functions as the user interface layer. PhoneGap's JavaScript engine controls the WebKit component whereas PhoneGap's native engine controls the device's native functionalities and features. The study also examined PhoneGap's interfaces and how to exploit them in mobile development, and how PhoneGap can be extended with custom-made plugins.

The final result of the project was a mobile application created for the Blood Service which allows users to make blood donations more frequently, by reminding them when it is possible to make a new donation. Besides the reminding feature, the application also works as an information source. The application will be released later in the Apple AppStore. Because of the cross-platform implementation, the application can later easily be published in more than one mobile platform.

This experience of exploiting PhoneGap in application development showed it to be particularly useful owing to its potential to act as an application wrapper, and because it allows a much more gentle learning curve when compared to the development of native applications.

Keywords	PhoneGap, HTML5, JavaScript, cross-platform, application development, mobile

Sisällys

Lyhenteet

Ajax	Asynchronous JavaScript and XML, ohjelmointitekniikka, jonka avulla voidaan siirtää tietoa selaimen ja palvelimen välillä ilman koko verkkosivun uudelleen lataamista.
CSS	Cascading Style Sheets, kokoelma tyyliohjeita, jotka yhdistetään ennakkoon määritellyllä tavalla yhdeksi säännöstöksi, joka kuvaa WWW-sivun ulkoasua.
DOM	Document Object Model, ohjelmointirajapinta, joka mahdollistaa HTML- ja XML-dokumenttien puurakenteisiin käsiksi pääsemisen.
IDE	Integrated Development Environment, kehitysympäristö, sovellus, joka tarjoaa riittävät puitteet ja työkalut sovelluskehitykselle.
JSON	JavaScript Object Notation, JavaScriptin tietorakenteeseen perustuva yksinkertainen tiedonsiirtomuoto.
SDK	Software Development Kit, ohjelmointikehityssarja, sarja sovelluskehitystyökaluja, jotka mahdollistavat sovelluskehityksen tietylle alustalle.

1 Johdanto

Mobiilialustamarkkinoiden pirstoutuminen on saanut aikaan sen, että sovelluskehittäjän on yhä vaikeampi tavoittaa suurta mobiilikäyttäjäkuntaa omalla sovelluksellaan. Mikäli haluaisi kehittää sovelluksen usealle mobiilialustalle perinteisellä tapaa, pitäisi sovellus toteuttaa jokaisella alustalle erikseen. Tämä lisäisi valtavasti työtaakkaa ja tarkoittaisi useiden eri sovelluskehitystyökalujen lataamista ja asentamista. Sovelluskehittäjän tulisi myös osata useita ohjelmointikieliä. Alustariippumaton sovelluskehitys pyrkii ratkaisemaan nämä ongelmat tarjoamalla muun muassa mahdollisuuden käyttää yhteistä ohjelmointikieltä sovelluksen kehityksessä.

Suomen Punaisen Ristin Veripalvelu on voittoa tavoittelematon ja toiminnallisesti erillinen osa Suomen Punaista Ristiä. Veripalvelu hoitaa keskitetysti koko maan veripalvelutoimintaa. Sen toimintaan kuuluu muun muassa verenluovuttajien rekrytointi, verenluovutusten järjestäminen ja verivalmisteiden tuotanto, varastointi ja jakelu sairaaloihin. Veripalvelun toiminnan kannalta elintärkeää on uusien verenluovuttajien rekrytointi ja vanhojen verenluovuttajien säännölliset verenluovutuskäynnit.

Vanhojen verenluovuttajien epäsäännölliset verenluovutuskäynnit ja sellaisen toiminnallisen ja mielenkiintoisen järjestelmän puute, joka muistuttaisi vanhoja verenluovuttajia uudesta verenluovutusmahdollisuudesta edellisen käynnin jälkeen, on Veripalvelun toiminnan ongelma. Veripalvelu haluaisi saada jokaisen vanhan verenluovuttajan käymään verenluovutuksessa keskimäärin nykyisen 1,8 käyntikerran sijaan 2 kertaa vuodessa. Muutostarve ei siis ole suuri.

Insinöörityön tavoitteena on suunnitella ja toteuttaa Suomen Punaisen Ristin Veripalvelulle mobiilisovellus muistuttamaan vanhoja verenluovuttajia uudesta verenluovutusmahdollisuudesta edellisen käynnin jälkeen. Sovellus toimisi käyttäjälle muistutuksen lisäksi samalla myös informaatiokanavana. Sovelluksen toteuttamisen ohella tutkin insinöörityössä, kuinka PhoneGap-ohjelmistokehystä voidaan hyödyntää tämänkaltaisessa alustariippumattomassa mobiilisovelluskehityksessä.

2 Alustariippumaton mobiilisovelluskehitys

2.1 Haasteet

Mobiilikäyttöjärjestelmien pirstoutuminen on suurin syy alustariippumattoman mobiilisovelluskehityksen isoimpaan haasteeseen – tietotaidon ylläpitämiseen eri mobiilialustojen ominaisuuksista ja ohjelmointikielistä (Ghatol & Patel 2012: 8). Kuten taulukko 1 osoittaa, tällä hetkellä maailman älypuhelinkannassa on kuusi muita selkeästi suositumpaa käyttöjärjestelmää.

Taulukko 1. Maailmanlaajuiset älypuhelimien käyttöjärjestelmien markkinaosuudet vuoden 2011 viimeisellä neljänneksellä (Gartner Says Worldwide Smartphone Sales Soared in Fourth Quarter of 2011 With 47 Percent Growth 2012).

Mobiilikäyttöjärjestelmä	Markkinaosuus (%), 2011 viimeinen neljännes	Markkinaosuus (%), 2010 viimeinen neljännes
Android	50,9	30,5
iOS	23,8	15,8
Symbian OS	11,7	32,3
Blackberry OS	8,8	14,6
Bada	2,1	2,0
Windows Phone	1,9	3,4
Muut	0,8	1,5
Yhteensä	100,0	100,0

Nämä kaikki kuusi käyttöjärjestelmää toimivat omalla alustallaan, jotka tukevat osittain eri ohjelmointikieliä. Ohjelmoijalle asian tekee vieläkin hankalammaksi ikävä tosiasia, että jokainen mainituista alustoista tarvitsee oman ohjelmointikehityssarjan (SDK) ja osa vielä oman integroidun kehitysympäristön (IDE). Kaikki SDK:t eivät ole tietokoneen laitevaatimuksiltaan samanlaisia vaan, kuten taulukosta 2 käy ilmi, ne vaihtelevat paljon. Mikäli ohjelmoija haluaa sovelluksensa tukevan esimerkiksi neljää yleisintä mobiilikäyttöjärjestelmää, hänellä on oltava Windows- ja Mac-tietokone sekä useita eri kehitysympäristöjä asennettuina niihin.

Ohjelmoijan kannalta haastavampi asia on kuitenkin taulukon 2 osoittama ohjelmointikielien levinneisyys eri mobiilialustoissa. Yhteistä ohjelmointikieltä tai -ympäristöä ei ole olemassa (Constantinou ym. 2011: 18). Vaikka natiivisovelluksia on mahdollista kehittää muillakin kuin alustojen natiiveilla ohjelmointikielillä, hyöty on usein pieni rajapintojen puutteiden ja uusien ongelmien syntymisen takia (Allen ym. 2010: 5). Ohjelmoijan tulisikin näin ollen osata C++-, Objective-C- ja Java-ohjelmointikielet. Natiivisovelluksen kehitys usealle alustalle samanaikaisesti on paitsi hidasta myös vaikeaa.

Taulukko 2. Sovelluskehityksen vaatimukset eri mobiilialustoissa (Ghatol & Patel 2011: 11).

Mobiilikäyttöjärjestelmä	Tietokoneen käyttöjärjestelmä	Kehitysympäristö	Ohjelmointikieli
Android	Windows/Mac/Linux	Eclipse + Java + Android-lisäke	Java
iOS	Mac	Xcode	Objective-C
Symbian OS	Windows/Mac/Linux	Carbide.c++	C++
Blackberry OS	Windows	Eclipse + JDE + Java	Java
Bada	Windows	Bada	C++
Windows Phone	Windows	Visual Studio 2010	C#/.NET/Silverlight/ WPF

2.2 Vaihtoehtoinen menetelmä

Kaikilla mobiilialustoilla on kuitenkin yksi yhteinen tekijä, joka mahdollistaa natiivisovelluskehitystä helpomman ja nopeamman lähestymistavan: kaikki älypuhelimet sisältävät nykyään verkkoselaimen. Kuudesta suosituimmasta mobiilialustasta viidessä on WebKit-pohjainen selainmoottori, ainoastaan Windows Phone on poikkeus omalla Internet Explorer 7 -pohjaisella selainmoottorillaan. Aivan kuten taulukosta 3 käy ilmi, verkkoselaimen hyödyntäminen yhteisenä alustana sovelluskehityksessä mahdollistaa uusimpien verkkoteknologioiden, kuten HTML5:n ja CSS3:n ja JavaScript-komentosarjakielen, käyttämisen yhteisenä ohjelmointikielenä. Tämän lisäksi se tarjoaa sovelluskehittäjälle mahdollisuuden natiivirajapintojen käyttämiseen. (Ghatol & Patel 2011: 12–13; Balaz ym. 2011: 109.)

Taulukko 3. Ohjelmointikielivaatimukset mobiiliverkkosovelluksen ohjelmoinnissa käytettäessä verkkoselainta yhteisenä alustana.

Mobiilikäyttöjärjestelmä	Ohjelmointikieli
Android	HTML CSS JavaScript
iOS	
Symbian OS	
Blackberry OS	
Bada	
Windows Phone	

Yksinkertaisimmillaan mobiiliverkkosovellus on verkkosivu, johon otetaan yhteys puhelimen verkkoselaimella ja jolla on mobiilisovellukselle tyypillisiä toiminnallisuuksia. Yhtenäisyys mobiilialustojen selainmoottoreiden kesken tarkoittaa, että mobiiliverkkosivut näyttävät pieniä eroavaisuuksia lukuun ottamatta kaikilla alustoilla samalta eikä alustakohtaisia muokkauksia tarvitse näin ollen kehitysvaiheessa välttämättä suorittaa. (Constantinou ym. 2011.) Tällaisen sovelluksen, tai verkkosivun, yleinen ongelma on suorituskyky. Koska verkkoselainmoottorin täytyy lukea ja näyttää HTML- ja JavaScript-koodi suoraan lennosta ja näin ollen toimia tulkkina koodin ja mobiilialustan välissä, verkkosovelluksen suorituskyvyssä jäädään aina natiiviohjelmointikielellä ohjelmoidun sovelluksen suorituskyvystä selvästi. Myös puutteellinen mobiilirajapintojen käyttömahdollisuus on eräs yleisistä ongelmista mobiiliverkkosovelluskehityksessä. Puhelimelle tyypillisiin toiminnallisuuksiin päästään käsiksi ainoastaan selainmoottorin kautta ja vain, mikäli selain tukee niitä. (Castledine ym 2011: 3–4.)

Ennen alustariippumattomia ohjelmistokehyksiä osaksi mobiiliverkkosovelluksien puutteiden ja nopean kehittämismahdollisuuden takia useat ohjelmoijat ja yhteisöt käyttivät selainkomponentin upottamista natiivisovellukseen. Teknisesti tämä mahdollisti kaikkien natiivirajapintojen käytön mobiiliverkkosovelluksessa, jolloin siitä tuli natiivisovellukseksi naamioitu verkkosovellus. Myös PhoneGap, alustariippumaton ohjelmistokehys, käyttää tätä tekniikkaa hyväkseen. (Allen ym. 2010: 9–13.)

3 PhoneGap-ohjelmistokehys

3.1 Alkuvaiheet

PhoneGap-ohjelmistokehyksen ensimmäiset kehitysversiot syntyivät vuoden 2008 elokuussa San Franciscossa järjestetyssä iPhone Dev Camp -tapahtumassa. Siellä pieni joukko kanadalaisen yrityksen, Nitobin, työntekijöitä keksi ratkaisun selainkomponentin upottamiseksi iPhonen natiivisovellukseen. Aluksi he keskittyivät ainoastaan iPhoneen, ja vasta kun sen tuki oli hyvä, he alkoivat kehittämään tukea Android-puhelimille. Tämä tapahtui kuitenkin nopeasti, sillä Nitobi julkaisi tuen Androille ja Blackberrylle jo kahden kuukauden päästä.

PhoneGapin voitettua People's Choice -palkinnon vuoden 2009 helmikuussa Web 2.0 Expo -tapahtumassa tieto PhoneGapistä ja sen mahdollisuuksista levisi nopeasti isomman yleisön keskuudessa ja PhoneGap sai taakseen vankan joukon mobiilisovelluskehittäjiä. Myös isot mobiilimaailman yritykset, kuten IBM, SonyEricsson, Symbian ja Palm, näkivät PhoneGapissä valoisan tulevaisuuden ja alkoivat tukea sen kehittämistä. Lokakuussa vuonna 2011 Adobe hankki Nitobin itselleen, ja samalla PhoneGap luovutettiin Apache Software Foundationille, jotta varmistettiin sen avoin hallintamalli tulevaisuudessakin. (Gowell & McWherter 2012: 309.)

Taulukosta 4 nähdään PhoneGapin varsin nopea julkaisutahti ja kehitys. Ensimmäisen vakaan version, PhoneGap 0.6.0:n, jälkeen siihen on lisätty tuki neljälle muulle mobiilialustalle. Tätä kirjoitettaessa PhoneGap on edennyt 1.6.1-versioonsa ja tukee jo seitsemää yleisintä mobiilialustaa.

Taulukko 4. PhoneGapin historia ja kehitys (Abdullah 2011).

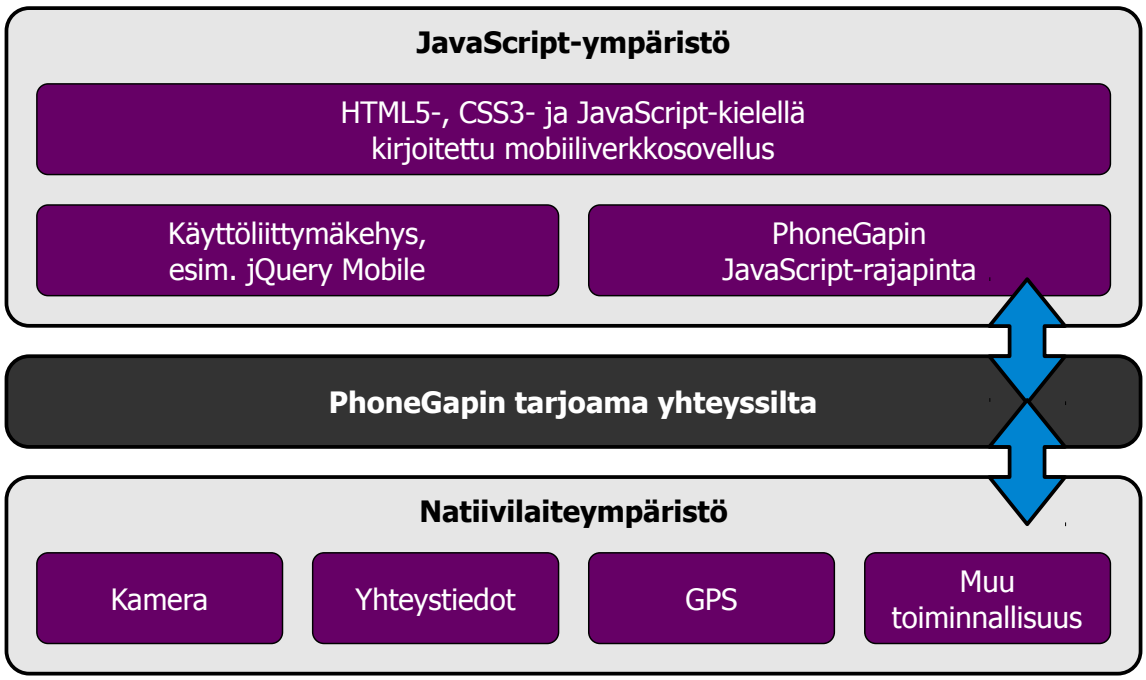
Päivämäärä	Tapahtuma
Elokuu 2008	PhoneGap esitellään ensimmäisen kerran. Tuki ainoastaan iPhonelle.
Lokakuu 2008	Tuki Androidille ja Blackberrylle.
Helmikuu 2009	Ensimmäinen vakaa versio, PhoneGap 0.6.0, julkaistaan.
Huhtikuu 2009	Voittaa Web 2.0 Expo -tapahtumassa

	People's Choice -palkinnon.
Elo–syyskuu 2009	Tuki Windows Mobilelle ja Nokia S60:lle.
Lokakuu 2010	Apple hyväksyy PhoneGapillä kehitettyjä sovelluksia AppStoreen.
Joulukuu 2010	Tuki Palmille.
Heinäkuu 2011	Tuki Symbian^3:lle. Versio 1.0.0 julkastaan.
Syyskuu 2011	Tuki Windows Phone 7:lle.
Huhtikuu 2012	Versio 1.6.1 julkaistaan.

3.2 Arkkitehtuuri

Mobiiliverkkosovelluksen, jonka kehittämiseen on käytetty PhoneGap-ohjelmistokehystä, arkkitehtuuri mukailee kuvan 1 esittämän mallin rakennetta. PhoneGap toimii puhelimen toiminnallisuuksien ja HTML5-, CSS3- ja JavaScript-ohjelmointikielillä kirjoitetun sovelluksen välissä mahdollistaen niiden keskinäisen kommunikoinnin. Tyypillisimmillään PhoneGapillä kehitetty mobiiliverkkosovellus sisältää sovelluskehittäjän itse HTML5- ja CSS3-kielellä kirjoittaman verkkosivun, joka toimii sovelluksen perustana, ja JavaScript-kielellä kirjoittaman tiedoston, jolla ohjataan käyttöliittymää ja sovelluksen toiminnallisuuksia sekä voidaan hallita laitteen ominaisuuksia ja toiminnallisuuksia. Näiden lisäksi sovellukseen on usein myös upotettu käyttöliittymäkehys, jolla pyritään ehostamaan käyttöliittymää ja usein myös nopeuttamaan käyttöliittymän kehitystä sen tarjoamien valmiiden komponenttien ansiosta. (Ghatol & Patel 2011: 17–18.)

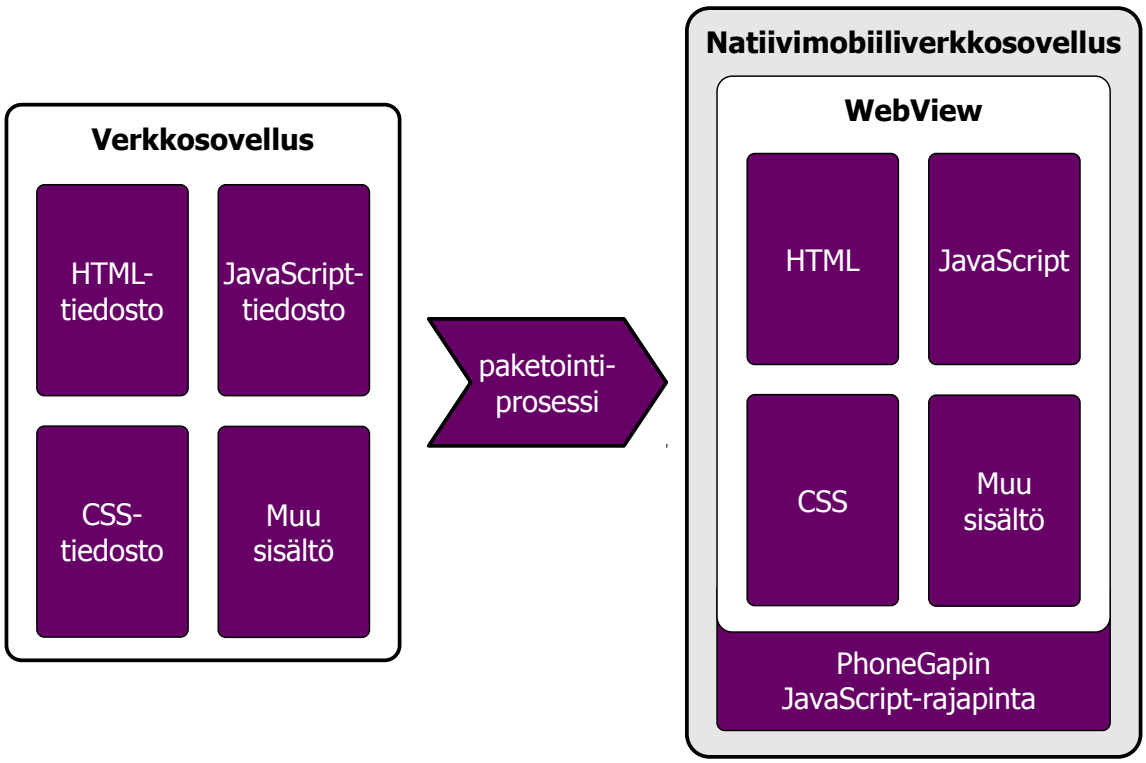
Jokaisessa mobiilialustassa on yksilöllinen puhelimen toiminnallisuuksien toteutus, ja niiden kaikkien mahdollinen hallitseminen olisi yksittäiselle sovelluskehittäjälle todella haasteellista. Se ei ainoastaan edellyttäisi useiden ohjelmointikielien tuntemusta vaan myös kunkin toteutuksen täsmällistä tietämystä. PhoneGap-ohjelmistokehyksessä on toteutettu kunkin mobiilialustan natiivikomponentti, joka kommunikoi laitteen kanssa ja toimii laitteen omassa ympäristössä. Sovelluskehittäjälle PhoneGap tarjoaa yhtenäisen JavaScript-rajapinnan laitteen toiminnallisuuksien ja sovelluksen välille kaikilla PhoneGapin tukemilla mobiilialustoilla. (Castledine ym. 2011: 197–199.)



Kuva 1. PhoneGapillä kehitetyn mobiilisovelluksen arkkitehtuuri (Ghatol & Patel 2011: 18).

Yhteyssillan lisäksi PhoneGap tarjoaa sovelluskehittäjälle toisen tärkeän ominaisuuden. Se paketoi kehittäjän tuottamat verkkodokumentit kuvan 2 mukaisesti mobiilialustan natiivisovellukseksi. Tämä paketointiprosessi mahdollistaa verkkosovelluksen lisäämisen eri mobiilialustojen sovelluskauppaan. Mobiiliverkkosovellus voi olla staattinen, niin että sovelluksen sisältö luodaan jo asennusvaiheessa sovelluksen sisältä löytyvien verkkodokumenttien pohjalta, dynaaminen, jolloin sisältö ladataan ulkopuoliselta palvelimelta ja tarjotaan näin ollen myös mahdollisia päivityksiä ilman uudelleen asennuksia, tai näiden kahden sekoitus, jossa vain osa sisällöstä ladataan palvelimelta.

Käyttäjä on sovelluksessa vuorovaikutuksessa käyttöliittymäkerroksena toimivan WebView-komponentin kanssa, johon mobiiliverkkosovelluksen käynnistyessä aloitusverkkosivu latautuu. Koska se täyttää puhelimen näyttöalueen kokonaan eikä siinä muun muassa ole verkkoselaimelle tyypillisistä osoitepalkkia, saadaan helposti vääristynyt vaikutelma, että sovellus olisi kirjoitettu kokonaan natiiviohjelmointikielellä. WebView-komponentin avulla voidaan hakea ja näyttää verkkosivuja natiivisovelluksen sisällä. Tämä komponentti löytyy jokaisesta PhoneGapin tukemasta mobiilialustasta, joskin sen nimi vaihtelee alustasta riippuen. Muun muassa Androidissa komponentti on WebView, iOS:ssä UIWebView ja Windows Phonessa WebBrowser. Komponentista voidaan kuitenkin käyttää yleisnimeä WebView. (Wargo 2012: 6–7.)



Kuva 2. PhoneGapin paketointiprosessi (Wargo 2012: 6).

WebView on PhoneGapin kannalta oleellinen myös siksi, että sen kautta natiiviympäristön kirjastot ja ohjelmistokehys altistuvat mobiilisovelluksen käyttöön. PhoneGapissä on toteutettu WebView-komponentin delegaattiluokka, jota kutsutaan, kun WebView'n sisältö muuttuu tai WebView'lle lähetetään URL-pyyntö. Kun PhoneGap havaitsee URL-pyynnön, delegaatti pysäyttää sen eikä se näin ollen pääse vaikuttamaan WebView-komponentin document.location-olioon, ennen kuin PhoneGap on tutkinut ja reagoinut siihen. (Trice 2012.)

File-skeeman sisältävään pyyntöön, joka yleensä edustaa pyyntöä ladata paikallinen tiedosto, PhoneGap reagoi lataamalla sen sisällön paikallisesti Webview-komponenttiin. Pysäyttäessään URL-pyynnön, jolla on HTTP-skeema ja joka edustaa pyyntöä ladata etäpalvelimella oleva tiedosto, PhoneGap ensin tarkistaa, onko etäpalvelimen osoite-avaruus mobiiliverkkosovelluksen sallittujen osoitteiden joukossa. Mikäli se on, PhoneGap lataa tiedoston paikallisesti WebView-komponenttiin, muutoin se aukeaa puhelimen oletusverkkoselaimeen ja on näin ollen sovelluksen kontekstin ulkopuolella. Mailto-, sms- ja tel-skeemallisen pyynnön PhoneGap siirtää puhelimen tarkoituksenmukaisen oletussovelluksen, esimerkiksi tekstiviestisovelluksen, avattavaksi ja käsiteltäväksi.

URL-pyyntö, jolla on gap-skeema, edustaa pyyntöä suorittaa PhoneGap-sovelluskehyksen ohjelmointirajapinnan komento.

Keskustellakseen puhelimen natiiviympäristön kanssa gap-skeemallisen URL-pyynnön tulee noudattaa seuraavanlaista protokollaa:

```
gap://liitännäinen.metodi?argumentti1=arvo1&argumentti2=
arvo2
```

jossa *liitännäinen* on PhoneGapin tarjoama rajapinta puhelimen toiminnallisuuteen, *metodi* on *liitännäisessä* määritelty metodi ja *argumentti1=arvo1&argumentti2=arvo2* on *metodille* välitettävä URL-koodattu lista argumenteista. (MacFadyen 2012.)

3.3 Ohjelmointirajapinnat

PhoneGapin tarjoamat JavaScript-ohjelmointirajapinnat mahdollistavat pääsyn osaan puhelimen toiminnallisuuksista, jotka muutoin olisivat vain natiivikielellä mahdollisia. PhoneGap noudattaa verkkoteknologiastandardien kattojärjestön W3C:n mukaisia verkkoselaimen ohjelmointirajapintojen määrityksiä niiltä osin kun niitä julkaistaan. Tällä hetkellä valmiit määritykset löytyvät vasta sijaintitieto-, tiedosto- ja muistiohjelmointirajapinnoille. Osassa eri mobiilialustojen oletusselaimista nämä määritykset on jo toteutettu, jolloin PhoneGap käyttää selaimen toteutusta oman toteutuksensa sijaan. W3C:n mukaisten määrityksien lisäksi PhoneGap mahdollistaa myös muiden eri alustojen natiiviominaisuuksien ja -toiminnallisuuksien käytön sen omia ohjelmointirajapintoja käyttäen. Kuten taulukosta 5 voi havaita, ohjelmointirajapintojen tuki kaikilla mobiilialustoilla ei ole täydellinen, osaksi alustakohtaisien ja osaksi puhelimien rajoituksien takia.

Taulukko 5. PhoneGapin tuki natiivirajapinnoille eri puhelimissa.

Rajapinta	Android	iPhone	Symbian	Blackberry	Bada	WP7
Kiihtyvyysmittari	✔	✔	✔	✔[2]	✔	✔
Kamera	✔	✔	✔	✔[2]	✔	✔
Kompassi	✔	✔[1]	✘	✘	✔	✔
Yhteystiedot	✔	✔	✔	✔[2]	✔	✔

Tiedosto	✔	✔	✘	✔[2]	✘	✔
Sijainti	✔	✔	✔	✔	✔	✔
Media	✔	✔	✘	✘	✘	✔
Yhteys	✔	✔	✔	✔	✔	✔
Ilmoitus	✔	✔	✔	✔	✔	✔
Muisti	✔	✔	✔	✔[2]	✘	✔

[1] Tuki ainoastaan iPhone 3GS- tai uudemmalla laitteella.
[2] Tuki ainoastaan Blackberry OS 5.x- tai uudemmalla laitteella.

PhoneGapin omat ohjelmointirajapinnat on toteutettu samalla tavalla kuin liitännäiset toteutetaan. Voidaan siis todeta PhoneGapin ohjelmointirajapintojen olevan sovelluskehittäjälle valmiiksi tarjottuja alustariippumattomia liitännäisiä. PhoneGapin JavaScript-ohjelmointirajapinnoissa on toteutettu useita olioita ja metodeja, joita sovelluksen JavaScript-ympäristöstä kutsumalla saadaan yhteys natiiviympäristössä toteutettuihin metodeihin. PhoneGapin JavaScript-ohjelmointirapinnat on toteutettu ylikuormittamalla globaalia window.navigator-oliota, ja muutamaa poikkeusta lukuun ottamatta niitä myös kutsutaan window.navigator-olion kautta seuraavasti:

```
navigator.rajapinta.metodi(argumentti, [argumentti]);
```

jossa *rajapinta* on PhoneGapin tarjoama rajapinta natiiville toiminnallisuudelle, *metodi* on *rajapinnassa* toteutettu metodi ja *argumentti* on *metodille* välitettävä argumentti. Usein *metodille* välitetään argumentteina kaksi takaisinkutsujafunktiota, joita kutsutaan sen mukaan, onko *metodi* suoritettu onnistuneesti vai epäonnistuneesti. Tällöin koodi on esimerkkikoodi 1 mukainen. (Kosmaczewski 2012: 104; API Reference.)

```
navigator.rajapinta.metodi(onnistunutTakaisinkutsu,
epäonnistunutTakaisinkutsu);
```

Esimerkkikoodi 1. Rajapinnan metodikutsu takaisinkutsujafunktiolla.

Kiihtyvyysmittari

Kiihtyvyysmittari on sensori, joka mittaa ja seuraa paikkatietoa kolmiulotteisessa maailmassa. Sillä on mahdollista saada laitteen x-, y-, ja z-koordinaatit ja seurata niiden liikettä ajan kuluessa. Käytännössä tämä tarkoittaa, että kiihtyvyysmittarirajapintaa voi käyttää puhelimen liikkeen, kallistuksen ja kiihtyvyyden mittaamiseen ja tarkkailuun. Esimerkkikoodissa 2 käyttäjälle ilmoitetaan onnistuneena rajapintakutsuna laitteen senhetkinen kiihtyvyys x-, y- ja z-tasolla ja epäonnistuneena kutsuna näytetään virheilmoitus.

```
navigator.accelerometer.getCurrentAccelerometer(
    successCallback,
    errorCallback
);

function successCallback(acceleration) {
    alert('kiihtyvyys x: ' + acceleration.x +
          'kiihtyvyys y: ' + acceleration.y +
          'kiihtyvyys z: ' + acceleration.z);
}

function errorCallback(error) {
    alert('Tapahtui virhe: ' + error);
}
```

Esimerkkikoodi 2. Laitteen kiihtyvyyden näyttäminen PhoneGapin tarjoaman rajapinnan avulla.

Kamera

Kamerarajapinnalla saadaan yhteys laitteen oletuskamerasovellukseen ja voidaan näin ollen hakea kuva sovellukseen suoraan laitteen kamerasta tai laitteen kuva-albumista. Esimerkkikoodissa 3 käyttäjän kuva-albumista valitsema kuva näytetään DOM-elementissä onnistuneena rajapintakutsuna ja epäonnistuneena rajapintakutsuna näytetään virheilmoitus.

```
navigator.camera.getPicture(
    successCallback,
    errorCallback,
    {
      quality: 50,
      destinationType: destinationType.FILE_URI,
      sourceType:pictureSource.PHOTOLIBRARY
    }
);

function successCallback(imgURI) {
    var image = document.getElementById('cameraImg');
    image.src = imageURI;
}
```

```
function errorCallback(error) {
    alert('Tapahtui virhe: ' + error);
}
```

Esimerkkikoodi 3. Kuva-albumista valitun kuvan näyttäminen PhoneGapin tarjoaman rajapinnan avulla.

Kompassi

Kompassirajapinnalla saadaan noudettua laitteen kompassin osoittama suunta. Suuntaa voidaan myös tarkkailla määrätyn väliajoin. Kompassirajapintaa voidaan käyttää myös ilmoituksen saamiseksi, kun määritelty poikkeama suuntauksessa ylittyy. Esimerkkikoodissa 4 käyttäjälle ilmoitetaan onnistuneena rajapintakutsuna laitteen kompassin osoittama suunta ja epäonnistuneena kutsuna näytetään virheilmoitus.

```
navigator.compass.getCurrentHeading(
    successCallback,
    errorCallback
);

function successCallback(heading) {
    alert('Suunta: ' + heading);
}

function errorCallback(error) {
    alert('Tapahtui virhe: ' + error);
}
```

Esimerkkikoodi 4. Laitteen kompassin osoittaman suunnan näyttäminen PhoneGapin tarjoaman rajapinnan avulla.

Yhteystiedot

Yhteystiedot-rajapinnalla saadaan yhteys laitteen yhteystietoihin. Sen avulla voidaan luoda ja hakea yhteystietoja laitteen tietokannasta. Esimerkkikoodissa 5 laitteen yhteystietoihin lisätään tietue ja käyttäjälle ilmoitetaan onnistuneena rajapintakutsuna tallennuksen onnistuminen ja epäonnistuneena kutsuna näytetään virheilmoitus.

```
var contact          = navigator.contacts.create();
contact.displayName  = 'Mikko';
contact.nickname     = 'Mikko';

var name             = new ContactName();
name.givenName       = 'Mikko';
name.familyName      = 'Kuustonen';

contact.name         = name;

contact.save(successCallback, errorCallback);
```

```
function successCallback(contact) {
    alert('Tallennus onnistui');
}

function errorCallback(error) {
    alert('Tapahtui virhe: ' + error);
}
```

Esimerkkikoodi 5. Yhteystiedon tallentaminen PhoneGapin tarjoaman rajapinnan avulla.

Tiedosto

Tiedostorajapinta on PhoneGapissa toteutettu W3C:n File API -määrityksen mukaisesti. Sillä voidaan liikkua laitteen tiedostojärjestelmän sisällä sekä lukea, luoda ja listata laitteen tiedostoja ja hakemistoja. Esimerkkikoodissa 6 käyttäjälle näytetään onnistuneena rajapintakutsuna laitteen juurihakemistossa olevan text.txt-tiedoston sisältö ja epäonnistuneena kutsuna virheilmoitus.

```
window.requestFileSystem(
    LocalFileSystem.PERSISTENT,
    0,
    successFS,
    errorCallback
);

function errorCallback(error) {
    alert('Tapahtui virhe: ' + error);
}

function successFS(fileSystem) {
    fileSystem.root.getFile(
        'text.txt',
        null,
        gotFileEntry,
        errorCallback
    );
}

function gotFileEntry(fileEntry) {
    fileEntry.file(readFile, errorCallback);
}

function readFile(file) {
    var reader        = new FileReader();
    reader.onloadend  = function (event) {
        alert('Tiedoston sisältö: ' + event.target.result);
    };
    read.readAsText(file);
}
```

Esimerkkikoodi 6. Tiedoston lukeminen ja sen sisällön näyttäminen PhoneGapin tarjoaman rajapinnan avulla.

Sijainti

Sijaintirajapinta on PhoneGapissa toteutettu W3C:n Geolocation API -määrityksen mukaisesti. Sen avulla laitteen GPS-sensori on sovelluksen käytettävissä ja sillä on mahdollista saada laitteen maantieteellinen sijainti ja tarkkailla sitä määrätyn väliajoin. Esimerkkikoodissa 7 käyttäjälle ilmoitetaan onnistuneena rajapintakutsuna laitteen senhetkisen sijainnin koordinaatit (leveys- ja pituusaste) ja epäonnistuneena kutsuna näytetään virheilmoitus.

```
navigator.geolocation.getCurrentPosition(
    successCallback,
    errorCallback
);

function successCallback(position) {
    alert('Leveysaste: ' + position.latitude +
          'Pituusaste: ' + position.longitude);
}

function errorCallback(error) {
    alert('Tapahtui virhe: ' + error);
}
```

Esimerkkikoodi 7. Laitteen sijainnin näyttäminen PhoneGapin tarjoaman rajapinnan avulla.

Media

Mediarajapinnan avulla sovelluksessa voidaan toistaa ja tallentaa äänitiedostoja. Se mahdollistaa muun muassa myös äänitiedoston kokonaispituuden ja toistettavan kohdan saamisen. Esimerkkikoodissa 8 luodaan Media-olio etäpalvelimella sijaitsevasta äänitiedostosta ja onnistuneena rajapintakutsuna käyttäjälle ilmoitetaan tiedoston pituus ja epäonnistuneena kutsuna näytetään virheilmoitus.

```
var url     = 'http://www.domain.com/audioFile.mp3',
    myAudio = new Media(url, showDuration, errorCallback);

myAudio.play();

function showDuration() {
    if (myAudio) {
        alert('Tiedoston pituus:' + myAudio.getDuration());
    }
}

function errorCallback(error) {
    alert('Tapahtui virhe: ' + error);
}
```

Esimerkkikoodi 8. Äänitiedoston toistaminen PhoneGapin tarjoaman rajapinnan avulla.

Yhteys

Yhteysrajapinta tarjoaa pääsyn laitteen tele- ja verkkoyhteystietoihin. Sillä saadaan tieto teleyhteyden tilasta ja verkkoyhteyden tyypistä. Esimerkkikoodissa 9 näytetään käyttäjälle senhetkisen aktiivisen verkkoyhteyden tyyppi.

```
var type   = navigator.network.connection.type,
    states = {
        Connection.UNKNOWN:  'Ei tietoa',
        Connection.ETHERNET: 'Ethernet',
        Connection.WIFI:     'Langaton',
        Connection.CELL_2G:  '2G',
        Connection.CELL_3G:  '3G',
        Connection.CELL_4G:  '4G',
        Connection.NONE:     'Ei yhteyttä'
};

alert('Yhteyden tyyppi: ' + states[type]);
```

Esimerkkikoodi 9. Verkkoyhteyden tyypin näyttäminen PhoneGapin tarjoaman rajapinnan avulla.

Ilmoitus

Ilmoitusrajapinnalla voidaan käyttäjän huomion saamiseksi näyttää huomio- tai vahvistusikkuna, toistaa puhelimen oletusäänimerkki tai laukaista laitteen värinä. Esimerkkikoodissa 10 näytetään käyttäjälle vahvistusikkuna, jossa on kaksi nappia, ja siihen reagoineena takaisinkutsujafunktiona näytetään käyttäjälle hänen painamansa napin indeksi.

```
var title        = 'Tämä on otsikko',
    message      = 'Tämä viesti näkyy ikkunassa',
    buttonLabels = 'Nappi 1, Nappi 2';

navigator.notificator.confirm(
    message,
    showButtonIndex,
    title,
    buttonLabels
);

function showButtonIndex(index) {
    alert('Painoit nappia: ' + index);
};
```

Esimerkkikoodi 10. Vahvistusikkunan näyttäminen PhoneGapin tarjoaman rajapinnan avulla.

Muisti

Muistirajapinta on PhoneGapissa toteutettu W3C:n Web SQL Database- ja Web Storage-määrityksien mukaisesti. Muistirajapintaa käyttämällä on mahdollista tallentaa tietoja puhelimen muistiin. Menetelmiä on kolme: verkkotietokanta (SQLite Database), paikallismuisti (localStorage) ja istuntomuisti (sessionStorage). Verkkotietokanta pohjautuu SQL-tietokantaan, jossa tietoa voidaan tallentaa taulukoihin. Tietokannan enimmäiskoko on rajoitettu viiteen megatavuun. Tietoa voidaan lisätä, poistaa ja muokata ylläpito-operaatioiden avulla sekä hakea kyselykomennoilla. Pakallismuistiin voidaan tallentaa tietoa merkkijonona avain-arvopareina. Istuntomuisti toimii paikallismuistin tapaan, mutta poikkeaa paikallismuisti- ja verkkotietokantaratkaisuista pysyvyyden osalta. Istuntomuistiin tallennettu tieto pyyhkiytyy muistista sovelluksen lopetettua, kun taas paikallismuisti ja verkkotietokanta ovat pysyviä muistiratkaisuja. Esimerkkikoodissa 11 ensin tallennetaan arvo *mikko* paikallimuistin *nimi*-avaimelle ja tämän jälkeen haetaan se paikallismuistista, näytetään se käyttäjälle ja tyhjennetään paikallismuisti kokonaan.

```
var key   = 'nimi',
    value = 'mikko';

window.localStorage.setItem(key, value);

value     = window.localStorage.getItem(key);

if (value) {
    alert(value);
    window.localStorage.clear();
}
```

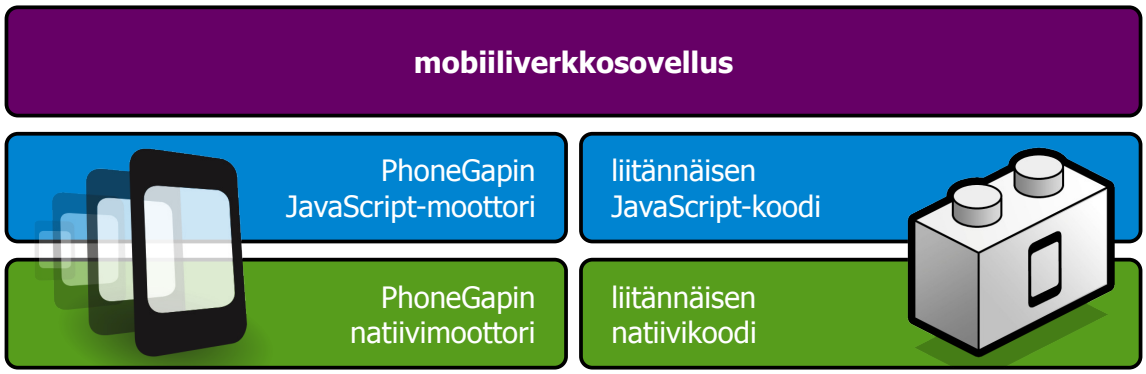
Esimerkkikoodi 11. Paikallismuistin hyödyntäminen PhoneGapin tarjoaman rajapinnan avulla.

3.4 Liitännäiset

PhoneGap-liitännäinen on eräänlainen PhoneGapin laajennus, jolla saadaan yhteys laitteen tiettyyn natiiviominaisuuteen tai -toiminnallisuuteen, mitä PhoneGap ei muuten mahdollistaisi. Liitännäiset ovat PhoneGapin tarjoamien rajapintojen kanssa yhteneviä ja toteutetaan samalla tavalla. Liitännäisen perusrunko koostuu JavaScript- ja natiiviohjelmointikielellä kirjoitetuista tiedostoista.

JavaScript-tiedostossa määritellään metodit, joilla kutsutaan liitännäisen toiminnallisuuksia. Natiiviohjelmointikielellä kirjoitettujen tiedostojen ansiosta PhoneGap-ohjelmistokehys pystyy keskustelemaan laitteen natiiviominaisuuksien ja -toiminnallisuuksien kanssa, joita JavaScript-ympäristöstä on kutsuttu. Voidaan siis todeta, että liitännäisen Javascript-tiedosto toimii sovelluksen rajapintana liitännäiselle ja natiiviohjelmointikielellä kirjoitetut tiedostot toimivat rajapintana puhelimen natiiviominaisuuksiin ja -toiminnallisuuksiin.

Kuvassa 3 on esitetty PhoneGapin arkkitehtuuri liitännäisten osalta. Liitännäisen JavaScript-koodilla laajennetaan PhoneGapin valmista JavaScript-moottoria, kun taas liitännäisen natiivikoodilla laajennetaan PhoneGapin valmista natiivimoottoria.



Kuva 3. PhoneGapin tarjoaman valmiin moottorin ja liitännäisen välinen arkkitehtuuri.

Koska liitännäisen tarkoituksena on saada yhteys laitteen natiivitoiminnallisuuteen tai -ominaisuuteen, se ei ole alustariippumaton. Mikäli halutun natiivitoiminnallisuuden haluaa toteuttaa useissa eri mobiilialustassa, täytyy jokaiselle alustalle tehdä oma natiiviympäristön toteutus. Tämä tarkoittaa, että JavaScript-tiedosto ja natiiviohjelmointikielellä kirjoitetut tiedostot täytyy tehdä jokaiselle alustalle erikseen.

3.4.1 Liitännäisen JavaScript-puolen toteutus

Koska liitännäisen JavaScript-rajapinta on sovelluksessa se, jota sovelluksen sisällä koodista käsin kutsutaan, se on myös liitännäisen tärkein osa. JavaScript-puolen toteutus koostuu kolmesta kohdasta: liitännäisen ilmentymän määrittelystä, liitännäisen toiminnallisuuden lisäämisestä ja liitännän rekisteröinnistä. Seuraavassa käsitellään liitännäisen toteutus iOS-mobiilialustalla. Tämän liitännäisen avulla luetaan laitteen akun varaus

ja näytetään se prosentteina DOM-elementissä. Liitännäistä kutsutaan nimellä *Battery.* Projektin *www*-kansioon luodan uusi tyhjä tiedosto, ja se nimetään *Battery.js*.

Liitännäisen ilmentymän määrittely

Tyhjän JavaScript-tiedoston alkuun määritellään liitännäisen ilmentymä, joka tässä tapauksessa on *Battery,* seuraavasti:

```
function Battery() {
};
```

Liitännäisen rekisteröinti

Liitännäisen rekisteröinti PhoneGapiin toteutetaan lisäämällä luokkametodi *Battery*-olioon. Metodi suoritetaan PhoneGapin alustaessa itsensä, ja suorituksen aikana luodaan *Battery*-olion ilmentymä. Se on pakollinen toimenpide, sillä liitännäisen toiminnallisuudet ovat ilmentymän metodeita eikä niitä siten voida kutsua, ennen kuin ilmentymä on luotu. Suorituksen jälkeen luotuun ilmentymään saadaan yhteys *window.plugins*-olion kautta. Rekisteröinti kokonaisuudessaan toteutetaan seuraavasti:

```
Battery.install = function () {

    if (!window.plugins) {
        window.plugins = {};
    }

    window.plugins.Battery = new Battery();

    return window.plugins.Battery;
};

PhoneGap.addConstructor(Battery.install);
```

Liitännäisen toiminnallisuuden lisääminen

Liitännäisen toiminnallisuus lisätään *Battery*-olion prototyyppiä ylikuormittamalla seuraavasti:

```
Battery.prototype.showLevel = function (type, win, fail) {
    PhoneGap.exec(win,fail,'Battery','showLevel',[type]);
};
```

jossa *showLevel* on ilmentymän metodi, jota kutsutaan sovelluksen JavaScript-ympäristöstä, *win* on metodille välitettävä onnistuneen metodin suorittamisen takaisinkutsuja-

funktio ja *fail* on metodille välitettävä epäonnistuneen metodin suorittamisen takaisinkutsujafunktio.

Liitännäisen toiminnan kannalta on ehdottaman tärkeää, että sen kommunikointi natiiviympäristön kanssa delegoidaan PhoneGapille kutsumalla suorituksen jossain vaiheessa *PhoneGap.exec*-metodia ja välittämällä sille tarvittavat parametrit. Nämä parametrit ovat edellä mainittujen *win*- ja *fail*-takaisinkutsujafunktioiden lisäksi rekisteröidyn liitännäisen natiiviluokan nimi, *'Battery'*, natiiviluokassa kutsuttavan funktion nimi, *'showLevel'* , ja natiivifunktiolle välitettävä parametritaulukko eli *[type]*.

Liitännäisen kutsuminen

JavaScript-ympäristöstä liitännäistä kutsutaan sen rekisteröidyllä nimellä ja siihen toteutetulla metodilla. Metodille annetaan parametreina tieto, joka halutaan välittää natiivikoodille, ja takasinkutsujafunktiot. Parametreista annetaan ensimmäisenä tieto, sitten onnistunut takaisinkutsujafunktio ja viimeisenä epäonnistunut takaisinkutsujafunktio. Esimerkkikoodissa 12 kutsutaan edellä toteutetun *Battery*-liitännäisen *showLevel*-metodia. Tietona sille annetaan *'power'*, ja onnistuneena takakaisinkutsujafunktiona käyttäjälle näytetään natiivikoodilta välitetty *tulos* eli akun varauksen prosentuaalinen arvo. Epäonnistuneena takakaisinkutsujafunktiona käyttäjälle näytetään natiivikoodilta välitetty *virhe* eli virheilmoitus.

```
window.plugins.Battery.showLevel(
    'power',
    function(tulos) {
        alert('Akun varaus: ' + tulos + '%');
    },
    function(virhe) {
        alert('Virhe: ' + virhe);
    }
);
```

Esimerkkikoodi 12. Liitännäisen kutsuminen JavaScript-ympäristöstä.

3.4.2 Liitännäisen natiivipuolen toteutus

Liitännäisen natiivipuoli iOS-ympäristössä toteutetaan ensiksi luomalla projektin *plugins*-kansioon Objective-C-luokka, joka perii PhoneGapissa toteutetun *PGPlugin*-luokan. Koska JavaScript-puolella kutsuttavalle *PhoneGap.exec*-metodille välitetyt parametrit edustavat natiivipuolen luokkaa ja luokkametodia, täytyy natiivipuolella luodun luokan ja sii-

nä määritellyn luokkametodin nimen täsmätä parametrien kanssa. Tässä tapauksessa luokan nimi on *Battery*.

Koska liitännäisessä ei ole kuin yksi kutsuttava metodi, luokan rajapintatiedoston – *Battery.h* – tulee näyttää seuraavalta:

```
#import <Foundation/Foundation.h>
#import <PhoneGap/PGPlugin.h>

@interface Battery : PGPlugin {

}

-(void) showLevel:(NSMutableArray*)arguments
         withDict:(NSMutableDictionary*)options;

@end
```

jossa *showLevel* on *Battery*-luokan luokkametodi, joka ei palauta mitään, mutta ottaa kaksi argumenttia; *arguments* on vapaasti muokattava taulukko ja saatu JavaScript-puolelta sekä *options* on vapaasti muokattava kirjasto ja hyödytön, sillä se on PhoneGapin toteutuksessa vanhentunut.

Liitännäisen natiiviluokan toteutustiedostossa – *Battery.m* – toteutetaan rajapintatiedostossa määritellyt luokkametodit. Luokkametodin perusrunko on liitännäisessä yksinkertainen. Siinä määritellään viittaus JavaScript-puolen funktioon, josta kyseistä natiivipuolen luokkametodia on kutsuttu, ja mahdollisesti myös kyseisen JavaScipt-funktion kautta välitettyyn parametriin. Tämän jälkeen suoritetaan koodi, jolla saadaan yhteys haluttuun natiivitoiminnallisuuteen tai -ominaisuuteen. Sitten luodaan PhoneGapissa toteutettu *PluginResult*-olio, jolle annetaan suorituksen onnistumisen tai epäonnistumisen mukaan jokin ennalta määritellyistä tiloista. Esimerkiksi onnistuneena suorituksena olio saa tilakseen *PGCommandStatus_OK*. Mikäli liitännäisen natiivipuolen halutaan välittävän jotakin JavaScript-puolen jommallekummalle takaisinkutsujafunktiolle, se välitetään oliolle sen luomisen aikana merkkijonona. JavaScript-ympäristössä ajettava oikea takaisinkutsujafunktio saadaan kutsumalla *PluginResult*-olion *toSuccessCallbackString*- tai *toErrorCallbackString*-metodia, jolle välitetään luokkametodin alussa määritelty viittaus JavaScript-puolen funktioon. Lopuksi WebView-komponentin käsketään suorittaa juuri saatu takaisinkutsujafunktio kutsumalla *PGPlugin*-luokalta perittyä *writeJavascript*-metodia ja välittämällä sille aiemmin saatu takaisinkutsujafunktio.

```
#import ”Battery.h”

@implementation Battery

-(void) showLevel:(NSMutableArray*)argument
         withDict:(NSMutableDictionary)options {

  PluginResult* result = nil;
  NSString* jsString   = nil;
  NSString* callbackId = [arguments objectAtIndex:0];
  NSString* type       = [arguments objectAtIndex:1];

  @try {

    float statusRaw;
    NSUInteger status;
    NSString* valueStr = nil;

    statusRaw = [[UIDevice currentDevice] batteryLevel];

    if (statusRaw < 0) {
        status   = 50;
    } else {
        status = statusRaw * 100;
    }

    valueStr = [NSString stringWithFormat:@”%d”, status];

    result   = [PluginResult
                    resultWithStatus:PGCommandStatus_OK
                     messageAsString:valueStr];

    jsString = [result toSuccessCallbackString:callbackId];

  } @catch (NSException *exception) {
    valueStr = @”Akun varausta ei voida lukea!”;

    result   = [PluginResult
                    resultWithStatus:PGCommandStatus_ERROR
                     messageAsString:valueStr];

    jsString = [result toErrorCallbackString:callbackId];

  } @finally {

    [self writeJavascript: jsString];
  }

}
```

Esimerkkikoodi 13. Liitännäisen toteutustiedosto.

Esimerkkikoodissa 13 on viittaus JavaScript-funktioon saatu luokkametodille välitetyistä argumenteista. On huomioitavaa, että tämä viittaus on aina argumenttitaulukon ensimmäinen olio ja varsinainen luokkametodille JavaScript-puolelta välitetty parametri on argumenttitaulukon toinen olio. Esimerkkikoodissa 13 luokkametodille välitetty *type*-argumentti saadaan siis seuraavasti:

```
NSString* type       = [arguments objectAtIndex:1];
```

joskin se on esimerkissä vain havainnollistamisen takia. Esimerkissä yritetään lukea laitteen akun varauksen arvo, joka on *0–1* tai *-1*, mikäli akun varauksen seuranta ei ole käytössä, ja muuttaa se joko vastaavaksi prosenttiarvoksi tai oletusarvoksi *50*. Mikäli siinä onnistutaan, prosenttiarvo välitetään *PluginResult*-oliolle. Epäonnistuneesta yrityksestä lukea akun varaus oliolle välitetään virheilmoitus. Lopuksi esimerkkikoodissa 13 käsketään WebView-komponentin suorittaa takaisinsuorittajafunktio seuraavasti:

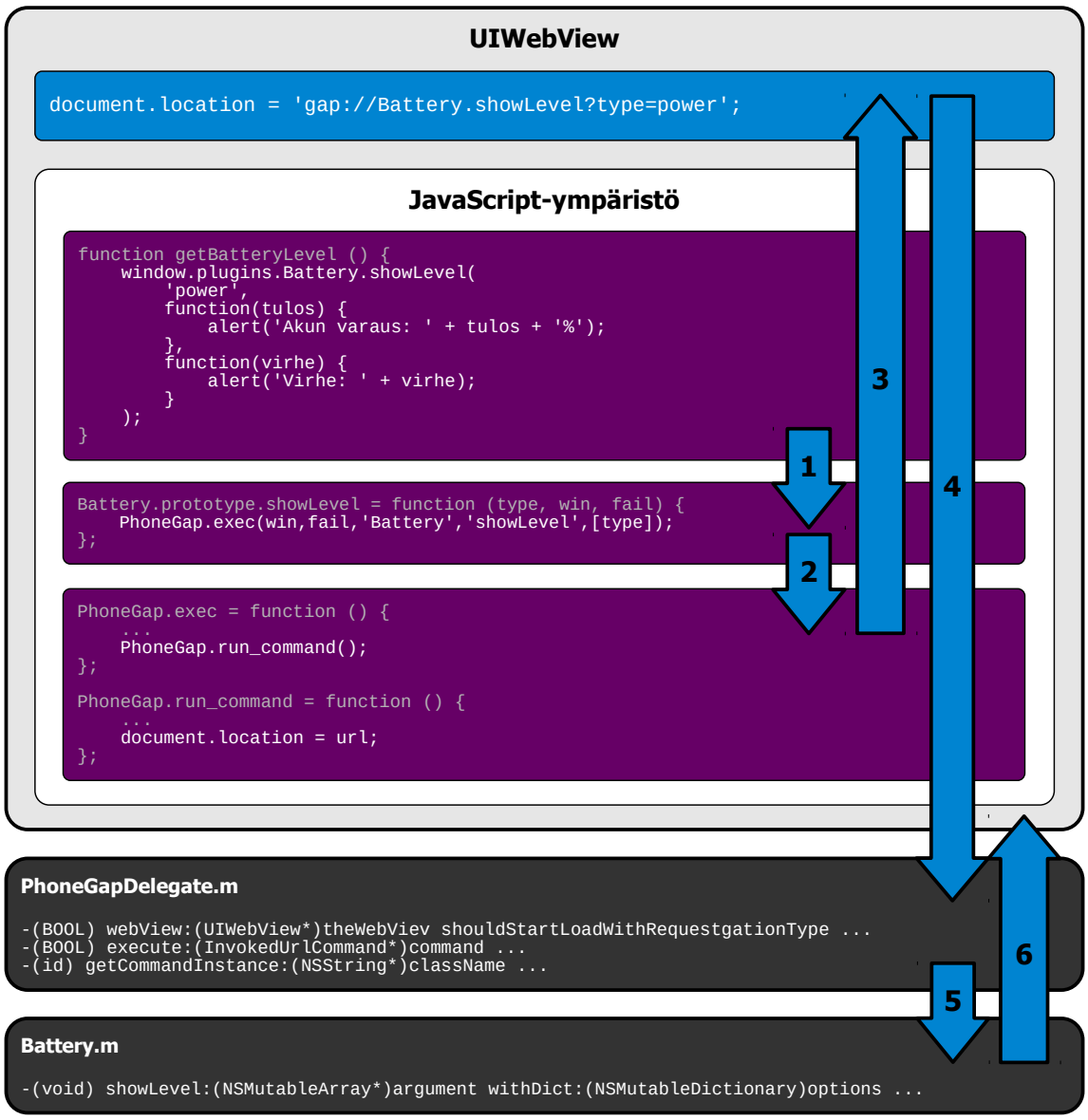
```
[self writeJavascript: jsString];
```

Natiiviluokka täytyy vielä rekisteröidä projektiin, jotta PhoneGap tietää, mitä luokkaa sen täytyy kutsua *PhoneGap.exec*-metodin suorituksen aikana. Rekisteröinti tehdään lisäämällä *PhoneGap.plist*-tiedoston *Plugins*-kirjaston sisälle *avain-arvo*pari, jossa avaimena on natiiviluokan nimi. Arvona on se merkkijono, joka annettiin *PhoneGap.exec*-metodille parametrinä. Tässä tapauksessa avaimena on *Battery* ja arvona *Battery*.

3.4.3 Liitännäisen arkkitehtuuri

Koska WebView-komponentin kautta kulkeva kommunikointi JavaScript- ja natiiviympäristön välillä tapahtuu merkkijonoilla eikä olioilla, valtavien tietomäärien yhtäaikainen liikuttelu ympäristöjen välillä ei ole suotavaa – eikä aina edes mahdollista. Kuvassa 4 on esitetty tarkemmin, mitä JavaScript- ja iOS-ympäristössä tapahtuu, kun liitännäistä kutsutaan JavaScript-ympäristöstä käsin. Kuvan esimerkissä on käytetty edellä toteutettua *Battery*-liitännäistä esimerkkinä.

Kohdassa 1 sovelluksen JavaScript-puolelta kutsutaan liitännäisen JavaScript-tiedostossa määriteltyä liitännäisen ilmentymän metodia, joka kutsuu PhoneGapissä toteutettua *PhoneGap*-olion *exec*-metodia ja välittää sille liitännäisen tarvittavat parametrit. Kun *exec*-metodia suoritetaan, kutsutaan siinä kohdan 2 mukaisesti lopulta *PhoneGap*-olion *run_command*-metodia. Tämän metodikutsun seurauksena UIWebView-komponentille lähetetään pyyntö muuttaa *document.location*-oliota. Kohdassa 3 on esitetty, miltä *document.location*-olioon vaikuttava URL-pyyntö näyttää *Battery*-liitännäisen *showLevel*-metodin kutsumisen jälkeen.



Kuva 4. Liitännäisen kutsumisen aikana tapahtuva vuorovaikutus iOS-ympäristössä.

Koska PhoneGapissä on natiiviympäristössä toteutettu UIWebView-komponentin delegaattiluokka, UIWebViewDelegate-luokalta perivä *PhoneGapDelegate*-luokka, PhoneGap pystyy havaitsemaan kohdassa 3 tapahtuneen URL-pyynnön ja tarvittaessa estämään URL-pyynnön vaikutuksen *document.location*-olioon. Kohdassa 4 *PhoneGapDelegate*-luokka havaitsee ja nappaa URL-pyynnön. Kun URL-pyyntö on tarkistettu ja todettu gap-skeemalliseksi, siitä poimitaan liitännäisen natiiviluokan nimi ja luokkametodi. Tämän jälkeen kohdan 5 mukaisesti *PhoneGapDelegate*-luokan metodista kutsutaan liitännäisen natiiviympäristössä toteutettua luokkametodia. Kun liitännäisen luokkametodissa on suoritettu tarvittavat natiivitoiminnallisuuteen tai -ominaisuuteen liittyvät operaatiot, UIWebView-komponentille ilmoitetaan suoritettava takaisikutsujafunktio. Kuten

kohdassa 6 on kuvattu, liitännäisen natiivipuolelta välitetään tieto takaisinkutsujafunktioden avulla UIWebView-komponentille, joka lopulta suorittaa toisen JavaScript-ympäristössä kohdan 1 mukaisesti toteutetuista takaisinkutsujafunktioista.

3.5 Ohjelmointiympäristöt

Vaikka PhoneGap mahdollistaakin alustariippumattoman mobiilisovelluskehittämisen ja koodin uudelleen käyttämisen eri alustoja varten, asettavat alustat itsessään kehittäjän kannalta hieman rajoituksia ja ongelmia. Seuraavassa niistä yleisimmät.

Android

Android-alustalle tähtäävän kehittäjän on asennettava Eclipse-kehitysympäristö sekä Android ADT- ja PhoneGapin Android-liitännäinen. Google vaati 25 dollarin kertamaksullisen kehityslisenssin vasta, kun sovelluksen haluaa julkaista Android Marketissa.

iOS

Applen iPhone- , iPad- ja iPod Touch -tuotteisiin tähdätessä on kehittäjällä oltava Mac-tietokone, Xcode-kehitysympäristö ja PhoneGapin iOS-liitännäinen. Ennen kuin sovelluksen toiminnallisuuksia voi kokeilla näissä laitteissa, kehittäjän on hankittava Apple Development Program -kehityslisenssi. Se maksaa 99 dollaria vuodessa.

Symbian OS

Symbian-puhelimia tukeakseen kehittäjällä tulee olla Windows-tietokone ja siihen asennettuna Cygwin-työkalusarja. Myös Symbian s60 -ohjelmointikehityssarjan pitää olla asennettuna yhdessä PhoneGapin Symbian-liitännäisen kanssa.

BlackBerry OS

Saadakseen sovelluksen BlackBerry-laitteisiin tulee kehittäjän rekisteröityä ilmaiseen BlackBerryn sovelluskehitysohjelmaan ja hankkia RIM-koodiavaimet. Näiden lisäksi kehittäjän on ladattava BlackBerry WebWorks -ohjelmointikehityssarja, PhoneGapin Black-

Berry-liitännäinen ja Apache Ant. Kehitysympäristöksi käy mikä tahansa Java-kehitysympäristö, joka toimii Windows XP- tai Windows 7 -käyttöjärjestelmässä.

Bada

Toteuttaakseen sovelluksen Bada-alustalle kehittäjällä täytyy olla Windows-käyttöjärjestelmällä varustettu tietokone, johon on asennettu Bada-ohjelmointikehityssarja ja -kehitysympäristö. Näiden lisäksi sovelluksen kehittäminen PhoneGapin avulla vaatii PhoneGapin Bada-liitännäisen asentamisen.

Windows Phone

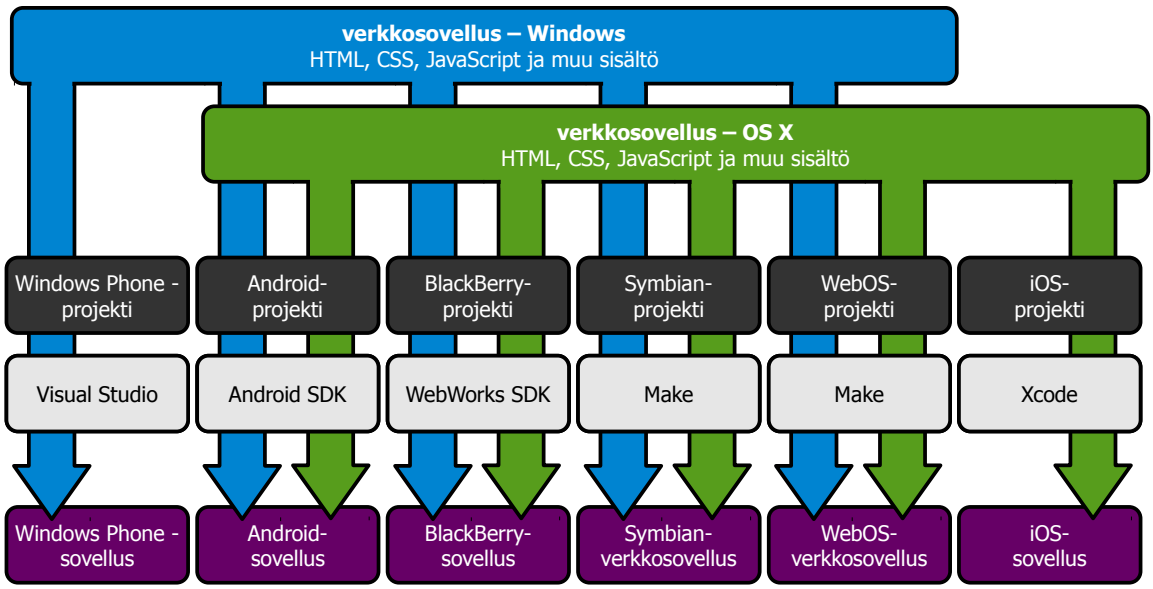
Windows Phone -puhelimissa toimivan sovelluksen kehittäminen PhoneGapilla vaati Windows Development Program -kehityslisenssin. Applen tavoin Microsoft on omaksunut 99 dollarin vuosimaksun. Myöskään Windows Phone -puhelimissa ei sovellusta voi kokeilla ilman lisenssiä.

3.6 Työnkulku

PhoneGapin hyöty eri mobiilialustoille tähtäävien sovelluksien kehittämisessä joko JavaScript- ja natiiviympäristön edustamana yhteisenä yhteyssiltana tai natiivisovelluksen paketoijana on ilmeinen. Maksimaalisen hyödyn saavuttaminen vaatii sovelluskehittäjältä kuitenkin paljon. Vaikka PhoneGapin tarjoamat rajapinnat laitteen natiivitoiminnallisuuteen ja -ominaisuuteen ovat jokaisella PhoneGapin tukemalla mobiilialustalla yhtenevät, itse PhoneGap-ohjelmistokehys on jokaisella tuetulla alustalla erilainen. Tämän takia sovelluskehittäjän on muun muassa tehtävä oma sovellusprojektinsa jokaiselle alustalle erikseen.

Eri projektien luomisen lisäksi tulee jokaiseen projektiin tuoda kyseisen mobiilialustan PhoneGap-versio. Koska PhoneGap on toteutettu eri tavalla eri alustalla, se saattaa myös toimia eri tavalla alustasta riippuen. Osaksi tämän vuoksi sovelluksen eri mobiilialustoille tehtyjen projektien testaaminen useilla eri mobiilialustojen laitteilla on välttämätöntä. On itsestään selvää, että sovelluksen testaaminen eri laitteilla ja ongelmien korjaaminen on aikaa vievää.

Sovelluskehittäjälle oman haasteensa tuo myös se ikävä tosiasia, että suunnatakseen sovelluksensa esimerkiksi Windows Phone- ja iOS-laitteille kehittäjällä täytyy olla Windows- ja OS X -käyttöjärjestelmät asennettuna tietokoneelle. Usein tämä on mahdollista vain kahdella erillisellä tietokoneella. Kuvassa 5 on esitetty eräs sovelluskehityksen varjopuolista – käännösprosessi eri mobiilialustoille. Sovelluskehittäjällä täytyy olla eri alustoille tähdätäkseen eri sovelluskehitystyökalut, joilla verkkosovellus voidaan paketoida mobiiliialustan natiivisovellukseksi, ja mahdollisesti myös eri tietokoneet. Näiden haasteiden lisäksi PhoneGap-projektin rakenne vaihtelee mobiilialustasta riippuen.



Kuva 5. PhoneGap-projektin käännösprosessi eri mobiilialustoille.

Kaiken edellä mainittujen kehitysvaiheen haasteiden helpottamiseksi PhoneGap tarjoaa pilvipalveluna PhoneGap Buildin. Sen avulla kehittäjä voi kirjoittaa verkkosovelluksen HTML-, CSS- ja JavaScript-koodin haluamallaan sovelluskehitystyökalulla ilman tarvetta ladata kaikkia kuvassa 5 esitettyjä sovellustyökaluja ja mobiilialustan tarvitsemia sovelluskehityssarjoja.

PhoneGap Buildin käyttöönotto vaatii palveluun rekisteröinnin, josta on ilmainen ja maksullinen vaihtoehto. Molemmat tilit mahdollistavat rajoittamattoman määrän avoimen lähdekoodin sovelluksien kääntämisen. Ilmaisella tilillä voi tämän lisäksi ylläpitää yhtä yksityistä sovellusta, kun taas maksullisella tilillä yksityisten sovelluksien määrä on nostettu 25:een. Avoimen lähdekoodin sovelluksen käyttäjän kirjoittamat tiedostot tulee PhoneGap Buildissa ladata GitHub-versionhallintajärjestelmästä. Yksityisen sovelluk-

sen lähdekoodin voi ladata PhoneGap Buildiin arkistoituna Zip-pakettina. Käyttäjän ei tarvitse lähdekoodin lataamisen lisäksi kuin määritellä sovelluksen kokoonpano erillisessä *config.xml*-tiedostossa, jonka tulee olla lähdekoodin kanssa sen juurihakemistossa. XML-tiedostossa määritellään muun muassa, minkä PhoneGap-version kanssa sovellus halutaan kääntää ja halutaanko puhelimien lisäksi tukea myös muita mobiililaitteita.

PhoneGap Build paketoi lähdekoodin lataamisen jälkeen eri mobiilialustojen natiivisovelluksiksi, joiden sovelluspaketit ovat ladattavissa suoraan PhoneGap Buildin sivuilta. On kuitenkin huomioitava, että PhoneGap Buildin käyttäminen vaatii kehittäjältä mobiilialustan sovelluskehityslisenssin, eli esimerkiksi iOS-mobiilialustalle paketointi vaatii kehittäjältä iOS-sovelluskehitysohjelman lisenssitietojen syöttämisen lähdekoodin latauksen yhteydessä. Tätä kirjoitettaessa PhoneGap Build ei myöskään tue natiivikoodin lataamista lähdekoodin mukana, joten liitännäisiä ei sovelluksessa voi hyödyntää. Palveluun on kuitenkin lisätty erillinen tuki viralliselle ChildBrowser-liitännäiselle, jolla voidaa näyttää etäpalvelimella olevia verkkosivuja PhoneGapilla paketoituun sovellukseeen upotettuna.

4 Suomen Punaisen Ristin Veripalvelun muistutussovellus

4.1 Sovelluksen vaatimukset

Insinöörityönä tuli toteuttaa mobiilisovellus Suomen Punaisen Ristin Veripalvelulle. Sovellus tuli tehdä Android-, iOS- tai Windows Phone -alustalle, ja sen tuli toimia itsenäisesti eli ilman palvelinpuolen backendiä. Sen jakelun käyttäjille tuli olla mobiilialustan virallisen sovelluskaupan kautta mahdollista. Sovellukselle asetettiin myös useita muita vaatimuksia.

Yksi tärkeimmistä vaatimuksista oli toteuttaa sovellukseen ominaisuus muistuttaa käyttäjää uudesta verenluovutusmahdollisuudesta. Tämä edellytti, että käyttäjä voisi tallentaa verenluovutuksiensa ajankohdat ja sukupuolensa sovellukseen pysyvästi. Näiden tietojen pohjalta sovellus pystyisi laskemaan seuraavan mahdollisen verenluovutusajankohdan ja muistuttamaan siitä käyttäjää. Asetuksista tuli pystyä kytkemään muistutuspalvelu pois päältä ja määrittelemään muistutuksien tiheys.

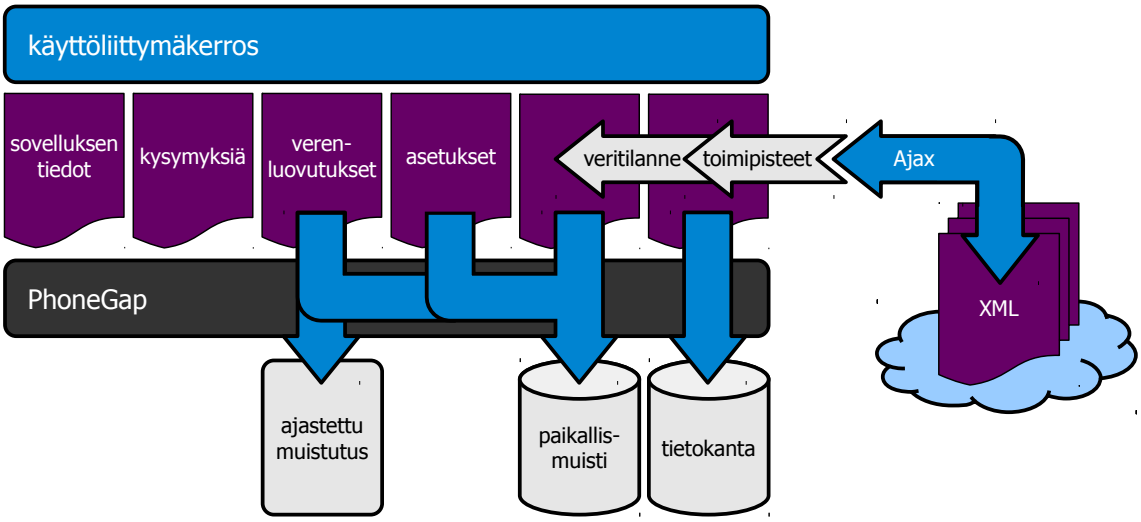
Sovelluksen asetuksiin tuli muistutuspalveluun liittyvän asetuksen ja käyttäjän sukupuolen lisäksi pystyä tallentamaan käyttäjän veriryhmä. Sovellukseen täytyi toteuttaa ominaisuus ladata ja näyttää käyttäjälle Veripalvelun etäpalvelimella sijaitseva päivittäin muuttuva veritilanne, joka olisi XML-tiedostona.

Sovelluksesta tuli veritilanteen ohella voida ladata ja näyttää Veripalvelu-toimipisteiden yhteystiedot Veripalvelun etäpalvelimelta, jotka niin ikään olisivat XML-tiedostona. Toimipisteiden yhteystietoja tuli voida tarkastella listana ja Suomen kartalla.

Sovelluksen kehitysvaiheen vaatimuksena oli ottaa huomioon turvallisuuteen ja tietoturvaan liittyvät seikat ja mahdolliset uhat. Myös mahdollisuus tulevaisuudessa siirtää sovellus muille mobiilialustoille tuli jo suunnittelu- ja kehitysvaiheessa huomioida. Sovellus tuli toteuttaa siten, että se olisi mahdollisimman helppokäyttöinen eikä näin ollen vaatisi erillistä ohjetta. Sovelluksen ulkoasun tuli myös mahdollisuuksien mukaan mukailla Veripalvelun graafista ohjeistusta.

4.2 Arkkitehtuuri

Mobiiliverkkosovellus koostuu kuvan 6 mukaisesti kuudesta pääkomponentista. Sovelluksen tiedoista käyttäjä löytää vastuuvapauslausekkeen ja tiedot Metropolian ja Veripalvelun yhteystyöstä sovelluksen kehityksessä. Sieltä käyttäjä voi myös halutessaan pyyhkiä kaikki tallennetut tiedot pois puhelimen muistista.



Kuva 6. Veripalvelun muistutussovelluksen arkkitehtuuri.

Kysymyksiä-sivulla on esitetty lista Veripalvelun toimintaan ja verenluovutukseen liittyvistä useimmin kysytyistä kysymyksistä ja annettu niihin vastaukset.

Käyttäjän on mahdollista selata, lisätä ja poistaa verenluovutuskertojaan omalla verenluovutuksien hallintasivulla. Sivulla näkyy verenluovutuksien lisäksi myös käyttäjän seuraava mahdollinen verenluovutuspäivämäärä ja keskimääräinen hemoglobiini verenluovutuskerroilla. Seuraavan mahdollisen verenluovutuspäivämäärän sovellus laskee automaattisesti käyttäjän määritellyn sukupuolen mukaan. Kuten kuvasta 6 voi havaita, käyttäjän lisäämät verenluovutuskerrat tallennetaan paikallismuistiin. Tämän lisäksi PhoneGapin liitännäisen avulla ajastetaan käyttäjän muistuttamiseksi ilmoitus. Ilmoitus näytetään käyttäjälle ensimmäisen kerran, kun hänen on taas mahdollista luovuttaa verta. Tämän jälkeen käyttäjää muistutetaan aina, kun hänen sovelluksen asetuksista määrittelemänsä ajanjakso on kulunut.

Myös käyttäjän määrittelemät asetukset tallennetaan paikallismuistiin. Asetuksista löytyy niin sovelluksen toimintaan liittyviä kuin myös käyttäjäkohtaisia asetuksia. Oman

sukupuolen ja veriryhmän lisäksi käyttäjä voi asetuksista määritellä, kuinka usein hän haluaa sovelluksen muistuttavan mahdollisuudesta luovuttaa verta uudelleen. Lisäksi asetuksista määritellään, sallitaanko muistutuksen yhteydessä puhelimen vakioäänimerkki ja värinä.

Päivittäin muuttuva veritilanne on käyttäjän mahdollista ladata Veripalvelun etäpalvelimelta milloin vain. Käyttäjää pyydetään lataamaan veritilanne, kun hän ensimmäisen kerran siirtyy veritilanne-sivulle. Kuten kuvassa 6 on esitetty, veritilanne ladataan Ajax-tekniikalla ja latauksen onnistuttua tallennetaan paikallismuistiin. Todellisuudessa veritilanne vaihtelee päivätasolla paljon, ja se myös päivitetään Veripalvelun etäpalvelimelle päivittäin. Näiden takia käyttäjää huomautetaan ja pyydetään lataamaan päivitetty veritilanne, mikäli viimeisestä latauksesta on kulunut yli vuorokausi.

Veripalvelutoimipisteiden yhteystiedot voi käyttäjä veritilanteen tapaan hakea Veripalvelun etäpalvelimelta. Ensimmäisen sovelluksen käyttökerran aikana käyttäjää pyydetään lataamaan yhteystiedot. Koska veripalvelutoimipisteiden aukioloajoissa saattaa olla ajankohtaisia tietoja, käyttäjälle huomautetaan, mikäli viimeisestä yhteystietojen latauksesta on kulunut yli viikko. Käyttäjä voi kuitenkin milloin vain ladata päivitetyt yhteystiedot palvelimelta. Yhteystiedot ladataan Ajax-tekniikalla ja kuvan 6 mukaisesti onnistuneen latauksen jälkeen tallennetaan tietokantaan.

4.3 Teknisen toteutustyylin ja kehitysympäristön valinta

Päädyimme yhdessä Veripalvelun kanssa siihen, että sovelluksen tekninen toteutus tehdään PhoneGap-ohjelmointikehyksen avulla. PhoneGap mahdollistaa lähes koko sovelluksen koodin uudelleen käyttämisen useamman mobiilialustan tukemiseksi ja näin ollen myös helpomman jatkokehityksen kuin natiivikoodilla toteutettu sovellus. Koska sovelluksen vaatimuksien toteuttaminen teknisesti ei paperilla ollut hirveän haasteellista eikä vaatinut puhelimen resurssien kovaa käyttöä, emme nähneet estettä PhoneGapin käytölle. Mobiilialustaksi valitsimme Applen iOS:n.

Sovelluksen kehityksessä on PhoneGapin ohella käytetty jQuery-kirjastoa ja jQuery Mobile -ohjelmistokehystä. jQuery on avoimen lähdekoodin JavaScript-kirjasto, jolla pyritään helpottamaan DOM-elementtien käsittelyä ja selkeyttämään ja yksinkertaistamaan

sovelluksen JavaScript-koodia. jQueryssa on toteutettu kirjaston tarjoamat toiminnallisuudet eri selaimille erikseen. Tämä myös vähentää huomattavasti selaimien välisiä ongelmia. Veripalvelulle tehdyssä sovelluksessa jQuery-kirjastoa käytettiin koodin saamiseen helppolukuisemmaksi, DOM-elementtien käsittelyyn ja helpottamaan Ajax-tekniikan käyttöä.

jQuery Mobile on jQueryyn ja sen käyttöliittymäkirjastoon, jQuery UI, pohjautuva ohjelmistokehys. Sillä luodaan helposti muokattavia käyttöliittymiä puhelimiin ja tabletteihin kehitettäviin verkkosovelluksiin. Sen avulla pystytään helpottamaan sovelluksen kehittämistä, minkä päämääränä on samalla tavalla toimivan sovelluksen lisäksi myös samalta näyttävä sovellus eri mobiilialustoilla. jQuery Mobile tukee muun muassa iOS-, Android-, Symbian- ja Windows Phone -alustoja, joskin esimerkiksi Androidin eri versioiden ja puhelinmallien välisiä eroja tuen puolesta löytyy aika paljon.

jQuery Mobilen tarjoamien lukuisten käyttöliittymäkomponenttien ja kosketustapahtumien ohella se mahdollistaa sovelluksen sisällön lataamisen Ajax-tekniikalla. Tämän ansiosta näytettävää verkkosivua ei uuden sisällön latautuessa tarvitse ladata kokonaan uudelleen. Koska sivut ladataan Ajax-tekniikan avulla, sivujen välinen siirtyminen voidaan animoida natiivisovellukselle tyypillisillä tavoilla. Sivut rakennetaan HTML5-standardia noudattaen semanttisella merkintätavalla. Sovelluksen kaikki sivut voidaan toteuttaa yhdessä HTML-tiedostossa.

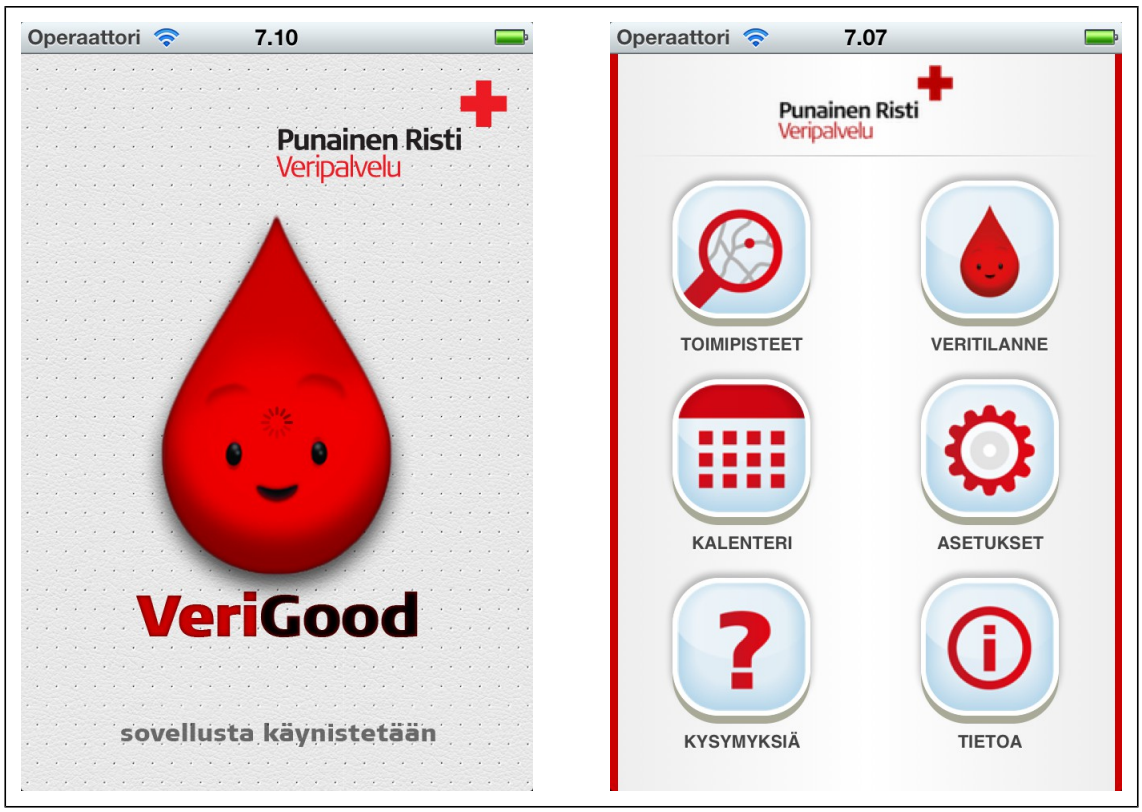
4.4 Sovelluksen kuvaus

Käyttöliittymä

Veripalvelulle insinöörityönä tehdyn mobiilisovelluksen käyttöliittymä mukailee natiivisovelluksille tyypillisiä rakenteita. Sovelluksen käyttöliittymää suunniteltaessa tutkin Googlen ja Applen virallisia käyttöliittymäohjeistoja Android- ja iPhone-puhelimille. Ohjeistoista löytyi tietoa muun muassa siitä, miten käyttöliittymä pitäisi hyvän tavan mukaan rakentaa ja millaiset rakenteet luetaan huonon tavan mukaisiksi. Veripalvelun muistutussovellus on toteutettu näiden niin sanottujen hyvien tapojen pohjalta.

Sovelluksen käynnistyessä käyttäjälle näytetään ensin kuvan 7 vasemmanpuoleinen käynnistyskuva. Kuva näkyy käyttäjälle, kunnes sovelluksen alustus on suoritettu eli

PhoneGap on valmis käytettäväksi sovelluksen sisällä. Kun sovellus on käynnistynyt, käyttäjälle näytetään kuvan 7 oikeanpuoleisen näkymän kaltainen päävalikko. Päävalikosta käyttäjä voi kuvakkeita painamalla siirtyä kuvakkeen edustamalle sovelluksen komponentin sivulle.



Kuva 7. Sovelluksen käynnistyskuva ja päävalikko.

Kaikki sovelluksen sivut päävalikkoa lukuun ottamatta noudattavat kuvassa 8 nähtävien sivujen rakennetta. Sivun yläosassa on palkki, jonka vasemmalla puolella on nappi edelliselle sivulle siirtymiseen ja oikealla puolella on mahdollinen sivun toiminnallisuuteen liittyvä nappi. Palkissa on myös nappien alapuolella sivun otsikko selkeästi luettavissa. Palkin lisäksi sivulla on koko jäljellä olevan ruudun täyttävä ja sivun varsinaisen sisällön sisältävä kääre. Kuvassa 8 on esitetty sovelluksen *tietoa*-sivu, josta käyttäjä voi lukea sovelluksen yleiset tiedot. Käyttäjän on mahdollista myös poistaa kaikki sovelluksen tallentamat tiedot painamalla yläpalkin oikealla puolella sijaitsevaa nappia. Ennen tietojen lopullista tuhoamista käyttäjältä kuitenkin kysytään toimenpiteelle vahvistus kuvan 8 oikean puolen havainnollistamalla tavalla.

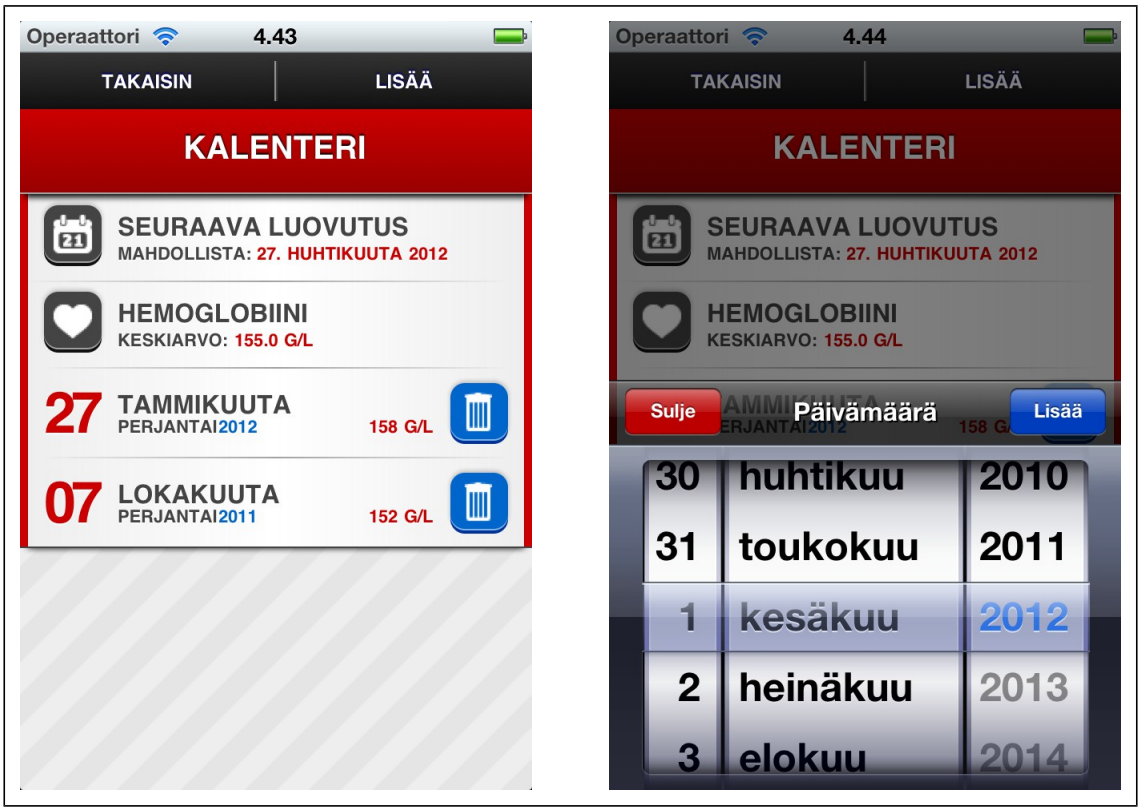
Kuva 8. Sovelluksen tietoa-sivu ja vahvistusikkuna.

Veripalvelun useimmiten kysytyt kysymykset ja niiden vastaukset käyttäjä pystyy tarkastamaan sovelluksen *kysymyksiä*-sivulta. Kysymykset on esitetty listan muodossa, kuten kuvassa 9 vasemmalla on havainnollistettu. Listaelementtiä painamalla käyttäjä saa kysymyksen ja siihen liittyvän vastauksen näkymään.

Kuvassa 9 oikealla on nähtävissä sovelluksen *asetukset*-sivu. Asetukset on kysymyksien tapaan esitetty listan muodossa, ja ne aukeavat käyttäjän määritettäväksi listaelementtiä painamalla. Asetuksen arvojen lisäksi aukinaisessa listaelementissä näkyy asetuksen lyhyt kuvaus. Jotta asetuksien nopea tarkastaminen olisi mahdollista, mikäli käyttäjä on määritellyt asetuksen, näytetään asetuksen arvo myös listaelementin otsikossa. Kuvasta 9 oikealta on havaittavissa, että käyttäjä on äänimerkkiasetusta lukuun ottamatta määritellyt kaikki asetukset.

Kuva 9. Sovelluksen kysymyksiä- ja asetukset-sivu.

Käyttäjä voi sovelluksessa myös tallentaa omia verenluovutuksien ajankohtiaan sovelluksen muistiin. Kuvassa 10 vasemmalta puolelta käy ilmi käyttäjän kaksi muistiin tallentamaa verenluovutusaikaa, sovelluksen automaattisesti laskema verenluovutuksien keskimääräinen hemoglobiini ja seuraava mahdollinen verenluovutusaika. Käyttäjä voi lisätä uuden verenluovutusajan ensin painamalla oikealla ylhäällä olevaa *lisää*-nappia ja sitten syöttämällä luovutusajankohdan ja hemoglobiinin. Kuvassa 10 oikealla on esitetty uuden verenluovutusajan lisääminen sovelluksen muistiin. Samalla tavalla kuin kuvassa, käyttäjä lisää oman hemoglobiininsa myös kiekkoa pyörittämällä ja oikean arvon ollessa valittuna painamalla sitten *lisää*-nappia.



Kuva 10. Sovelluksen verenluovutuskalenterin näkymä ja verenluovutuksen lisääminen.

Mikäli käyttäjän seuraava mahdollinen uusi verenluovutusaika on jo mennyt ohi ja käyttäjä on sallinut asetuksista muistutuksen, häntä muistutetaan uudesta verenluovutusmahdollisuudesta kuvan 11 mukaisesti. Mikäli käyttäjä on sovelluksen asetuksista sallinut äänimerkin muistutuksen yhteydessä, puhelin päästää vakioäänimerkin muistutuksen lauetessa. Vasemmalla puolella kuvassa 11 on esitetty juuri lauenneen muistutuksen näkyminen ilmoituspalkissa ja sovelluksen kuvakkeen päälle ilmestynyt tapahtumamerkki. Tapahtumamerkki pysyy kuvakkeen päällä, kunnes käyttäjä on reagoinut ilmoitukseen tai käynnistänyt sovelluksen uudelleen kuvaketta painamalla. Oikealla puolella kuvassa näkyy myöhemmin käyttäjän alas vetämä ilmoitusruutu. Ilmoituksessa näytetään käyttäjän viimeisin sovellukseen merkitsemä verenluovutusaika ja toivotetaan hänet tervetulleeksi luovuttamaan verta uudelleen. Sovelluksen ilmoitustapahtumaa painamalla käyttäjä pääsee suoraan sovelluksen kalenterinäkymään.

Kuva 11. Uuden verenluovutusmahdollisuuden muistutus käyttäjälle.

Veripalvelun päivittäistä veritilannetta on sovelluksessa mahdollista tarkastella kuvan 12 esittämällä tavalla. Vasemmalla puolella kuvaa on esitetty käyttäjän Veripalvelun etäpalvelimelta lataama veritilanne. Veritilanne esitetään veriryhmien mukaan listan muodossa, ja listaelementtiä painamalla veriryhmän veritilanne aukeaa käyttäjän tarkasteltavaksi. Mikäli käyttäjä on asetuksista määritellyt oman veriryhmänsä, näytetään sen veritilanne hänelle automaattisesti sivun auetessa.

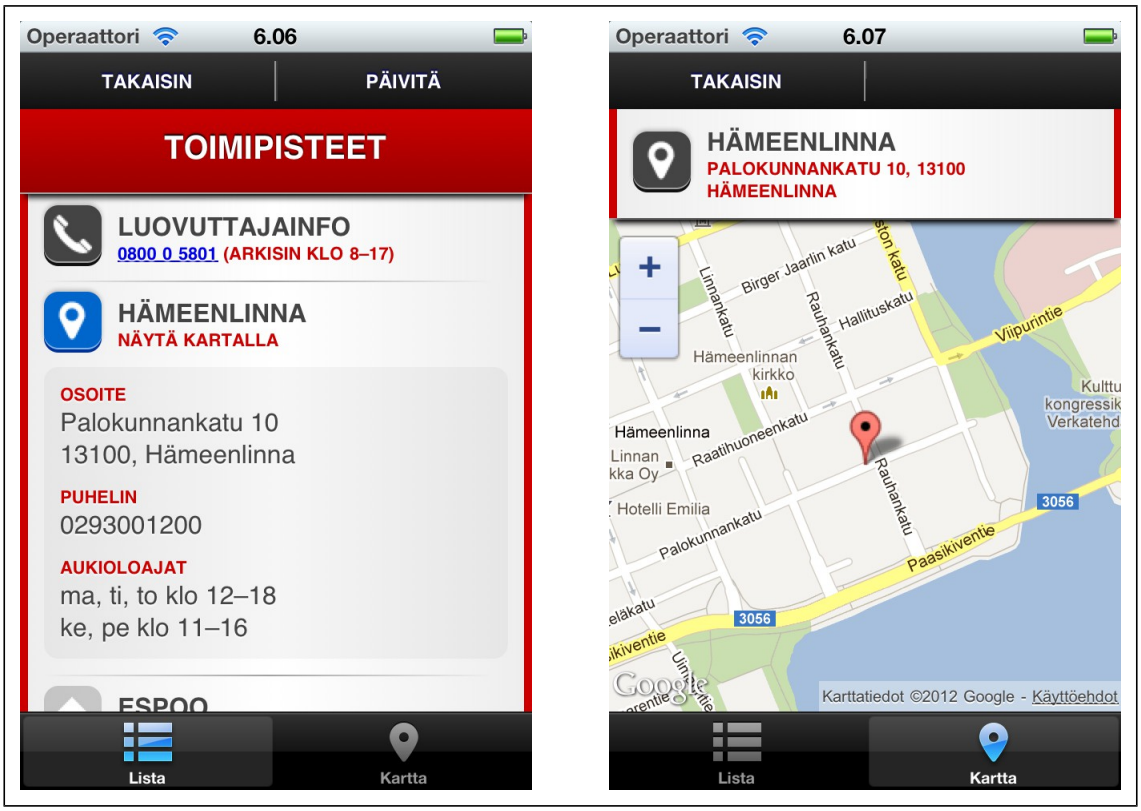
Koska veritilanne elää päivittäin ja eri veriryhmien veren tarve saattaa hetkessä muuttua kiireelliseksi, käyttäjää huomautetaan kuvassa 12 oikealla esitetyllä tavalla ladatun veritilanteen ollessa vähintään vuorokauden vanha. Käyttäjän on mahdollista itse ladata ja päivittää veritilanne milloin vain painamalla ruudun yläosassa olevan palkin oikeassa reunassa olevaa *päivitä*-nappia.

Kuva 12. Veritilanteen esittäminen käyttäjälle ja huomautus veritilanteen ollessa vanha.

Veritilanteen tapaan sovelluksessa esitetään Veripalvelun etäpalvelimelta ladattavissa olevat toimipisteiden yhteystiedot listan muodossa. Listaelementtiä painamalla veripalvelutoimipisteen yhteystiedot aukenevat käyttäjälle. Kuvassa 13 vasemmalla ovat nähtävissä käyttäjän valitseman Hämeenlinnan veripalvelutoimipisteen yhteystiedot.

Yhteystietoja on mahdollista myös tarkastella Suomen kartalla. Käyttäjä voi siirtyä listan ja kartan välillä ruudun alaosassa olevien välilehtien avulla. Mikäli käyttäjä painaa *kartta*-välilehteä, kaikki veripalvelutoimipisteet näkyvät yhtä aikaa Suomen kartalla. Käyttäjä voi myös painaa auki olevan yhteystiedon kaupungin nimeä tai sen vieressä olevaa nastakuvaketta, jolloin osoitetiedot aukenevat kuvassa 13 oikealla esitetyllä tavalla Suomen kartalle.

Koska Veripalvelun eri toimipisteissä saattaa silloin tällöin olla tavallisesta poikkeavia aukioloaikoja, käyttäjää huomautetaan vanhentuneen veritilanteen latauksen tapaisesti myös mahdollisesti vanhentuneista yhteystiedoista. Huomautus näytetään käyttäjälle, mikäli edellisestä yhteystietojen lataamisesta on kulunut yli viikko.



Kuva 13. Veripalvelutoimipisteiden yhteystietojen selaaminen listan muodossa ja kartalla.

Sovelluksen käyttöliittymä on toteutettu jQuery Mobilen avulla, mikä mahdollistaa niin sanotun yhden sivun sovellus -mallin hyödyntämisen. Tätä mallia käyttämällä sovelluksen rakenne on voitu toteuttaa usean HTML-dokumentin sijaan yhdellä HMTL-dokumentilla. Veripalvelun muistutussovelluksessa kaikki sivut eli kuusi pääkomponenttia on upotettu yhteen HTML-dokumenttiin, jossa näkyvissä on kuitenkin vain yksi sivu kerrallaan. Sivulta toiselle siirtyminen on animoitu jQuery Mobilen tarjoamalla liukutehosteella luomaan parempi illuusio natiivisovelluksesta.

Sivun mallipohja

Sivun mallipohja on jQuery Mobilen mallinpohjan mukainen. Sivu alkaa HTML5:ssa määritellyllä *doctype*-attribuutilla. Linkitykset sovelluksen käyttämiin JavaScript-kirjastoihin ja tyylitiedostoihin tehdään sivun *head*-tunnisteessa. *ViewPort*-metatunnisteessa määritellään sivun mitat ja tehdään sivun skaalaminen käyttäjälle mahdottomaksi. *Body*-tunniste sisältää kaikki sovelluksen sisällä esittämät sivut, joiden välillä käyttäjä voi siirtyä. Esimerkkikoodissa 14 on esitetty sivun mallipohjan *head*-tunnisteen rakenne.

```
<!DOCTYPE HTML>
<html>
<head>
  <title>PhoneGap</title>
  <meta charset="utf-8" />
  <meta name="viewport"
    content="width=device-width, initial-scale=1.0,
             maximum-scale=1.0, user-scalable=no" />
  <meta name="format-detection" content="telephone=no" />

  <!-- STYLE SHEETS -->
  <link rel="stylesheet"
    href="css/jquery.mobile-1.1.0.min.css" />
  <link rel="stylesheet" href="css/veripalvelu-min.css" />

  <!-- CODE BASE -->
  <script charset="utf-8" src="phonegap-1.3.0.js"></script>
  <script src="http://maps.google.com/maps/api/js?
    sensor=false"></script>
  <script src="js/jquery-1.7.min.js"></script>
  <script type="text/javascript">
    // DEFAULTS TO BE SETUP -- MUST BE BEFORE JQM LIBRARY
    $(document).bind("mobileinit", function(){
      // APPLY OVERRIDES HERE
      $.mobile.defaultPageTransition = "slide";
      $.mobile.loadingMessageTextVisible = true;
    });
  </script>
  <script src="js/jquery.mobile-1.1.0.min.js"></script>
  <script src="js/jquery.ui.map.min.js"></script>

  <!-- PHONEGAP PLUGINS -->
  <script src="js/applicationPreferences-min.js"></script>
  <script src="js/LocalNotification-min.js"></script>
  <script src="js/TabBar.js"></script>
  <script src="js/DatePicker-min.js"></script>

  <!-- MAIN FUNCTIONALITY -->
  <script charset="utf-8" src="js/veripalvelu.js"></script>
</head>
<body>
  <!-- PAGES ARE IMPLEMENTED HERE -->
</body>
<html>
```

Esimerkkikoodi 14. Sovelluksen *index.html*-dokumentin head-tunniste.

Sovelluksen kaikki seitsemän sivua, päävalikkoa lukuun ottamatta, on esimerkkikoodissa 15 esitetyn sovelluksen *asetukset*-sivun tapaisesti rakennettu. Sivu on kääritty *div*-elementin sisään. Elementin *data-role*-attribuutissa määritellään sivun kyseisen elementin rooli, joka tässä tapauksessa on *page*. Muut sovelluksessa käytetyt elementtien roolit ovat *header* ja *content*. *Header*-rooli määrittelee div-elementin toimimaan navigointipalkin tapaisesti sivun ylätunnisteessa. *Content*-rooli määrittelee sivun varsinaisen sisällön. Sisältö skaalautuu koko ruudulle, *header*-elementin alaosasta aina ruudun alaosaan asti.

```
<!-- SETTINGS -->
<div id="settings-page" data-role="page">

  <div data-role="header">
    <div class="ui-grid-a buttonBarTop">
      <a href="#dashboard-page" id="homeButton"
         class="ui-block-a" data-rel="back">Takaisin</a>
    </div>
    <h1>Asetukset</h1>
  </div>
  <!-- /HEADER -->

  <div data-role="content">
    <div id="settingsList">
      <div class="detailsContainer"></div>
      <div class="listContainer">
        <h3 id="group" class="profileSetting">Veriryhmä
          <span>Määritä asetus</span></h3>
        <h3 id="gender" class="profileSetting">Sukupuoli
          <span>Määritä asetus</span></h3>
        <h3 id="interval" class="appSetting">Muistutus
          <span>Määritä asetus</span></h3>
        <h3 id="sound" class="appSetting">Äänimerkki
          <span>Määritä asetus</span></h3>
      </div>
    </div>
  </div>
  <!-- /CONTENT -->

</div>
<!-- /PAGE -->
```

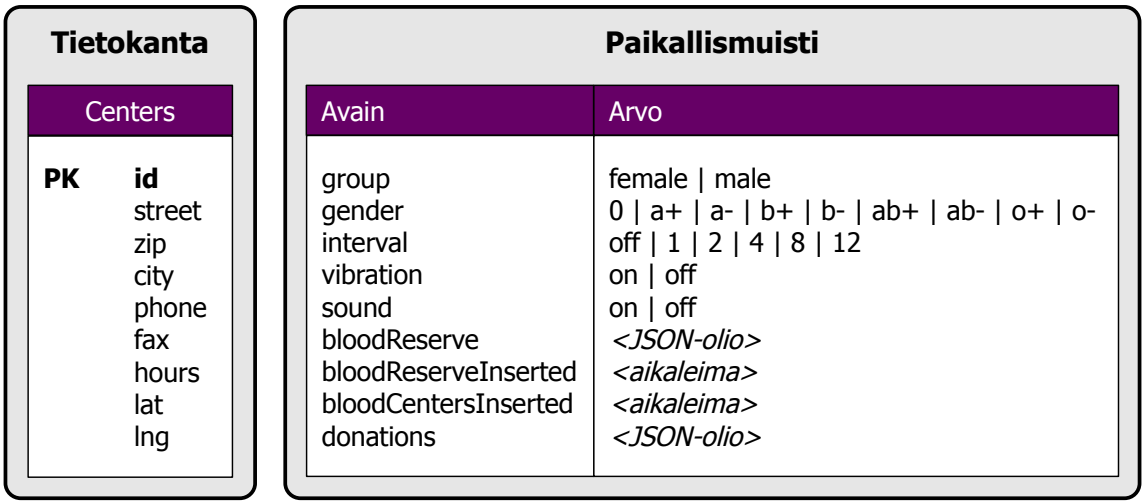
Esimerkkikoodi 15. Sovelluksen yksittäisen sivun rakenne.

Esimerkkikoodia 15 ja kuvassa 9 oikealla nähtävää *asetukset*-sivua vertaamalla voidaan todeta käyttäjän määriteltävissä olevan asetuksen olevan käärittynä *div*-elementtiin, joka kuuluu *detailsContainer*-luokkaan. Elementin sisältö eli asetuksen lyhyt kuvaus ja mahdolliset arvot luodaan dynaamisesti JavaScriptilla ajon aikana riippuen siitä, mitä listan elementtiä käyttäjä on painanut. Listan muodossa esitetyt asetukset sen sijaan ovat käärittynä *listContainer*-luokalliseen *div*-elementtiin, jossa jokainen listan elementeistä on *h3*-otsikkoelementti. Sovelluksen alustusvaiheessa jokaisen sivun nappeina toimiviin linkkeihin ja listan elementteihin luodaan jQuerylla painalluskuuntelija, joka laukaistuaan kutsuu kuuntelijassa määriteltyä takaisinkutsujafunktiota ja suorittaa napin tai elementin halutun toiminnan.

Rajapinnat

Veripalvelun muistutussovellus hyödyntää PhoneGapin tarjoamia ilmoitus- ja muistirajapintaa. Sovelluksessa käytetään paikallismuistia ja yhtä tietokantaa, joihin PhoneGapin rajapintaa käyttämällä saadaan yhteys. Kuvassa 14 on kuvattu sovelluksessa käytettä-

vän paikallismuistin ja tietokannan rakenne. Tietokannassa on yksi taulukko, jossa on yhdeksän kenttää. Taulukon *id*-kenttä toimii taulukon perusavaimena. Muut taulukon kenttien nimet ovat *street*, *zip*, *city*, *phone*, *fax*, *hours*, *lat* ja *lng*.



Kuva 14. Sovelluksen tietokannan ja paikallismuistin rakenne.

Veripalvelutoimipisteiden yhteystiedot ladataan etäpalvelimelta käyttäen Ajax-tekniikkaa. Tässä sovelluksessa *Asynchronous JavaScript and XML* -tekniikka mahdollistaa HTTP-pyynnön lähettämisen asynkronisesti taustalla. Lopulta HTTP-pyyntö palauttaa palvelimella sijaitsevan XML-tiedoston, josta tämän jälkeen parsitaan oleelliset tiedot. Nämä tiedot syötetään tietokannan taulukon tietoja vastaaviin kenttiin. Esimerkkikoodissa 16 on esitetty, miltä Veripalvelun etäpalvelimella sijaitsevan toimipisteiden yhteystiedot sisältävän XML-tiedoston yksi tietue näyttää.

```
<row>
  <toimisto_id>1</toimisto_id>
  <nimi>Helsingin Kivihaan veripalvelutoimisto</nimi>
  <nimi_sv>Stenhagens blodtjänstbyrå i
           Helsingfors</nimi_sv>
  <katuosoite>Kivihaantie 7</katuosoite>
  <katuosoite_sv>Stenhagsvägen 7</katuosoite_sv>
  <katuosoite_lisatiedot/>
  <katuosoite_lisatiedot_en/>
  <katuosoite_lisatiedot_sv/>
  <postinumero>00310</postinumero>
  <postitoimipaikka>Helsinki</postitoimipaikka>
  <postitoimipaikka_sv>Helsingfors</postitoimipaikka_sv>
  <aukioloajat>ma-to klo 11-18 pe klo 9-16</aukioloajat>
  <aukioloajat_en>Mon-Thu 11-18 Fri 9-16</aukioloajat_en>
  <aukioloajat_sv>må-to 11-18 fr 9-16</aukioloajat_sv>
  <yhteystiedot>Vaihde: 09 58011 Maksuton luovuttajainfo:
              0800 0 5801 (arkisin klo 8-17)</yhteystiedot>
  <yhteystiedot_en>tel. 09 58 011</yhteystiedot_en>
  <yhteystiedot_sv>tfn 09 58011 (växel) Infotelefon/
                  rådgivning tfn 0800 0 5801
```

```
                    (vardagarna kl. 8–17)</yhteystiedot_sv>
  <latitude>60.2125020</latitude>
  <longitude>24.9033213</longitude>
  <sivun_viitenimi>Kivihaka</sivun_viitenimi>
  <sivun_viitenimi_en>Kivihaka_en</sivun_viitenimi_en>
  <sivun_viitenimi_sv>Stenhagen</sivun_viitenimi_sv>
  <lisatiedot/>
  <lisatiedot_en/>
  <lisatiedot_sv/>
</row>
```

Esimerkkikoodi 16. Veripalvelutoimipisteen yhteystieto XML-muodossa.

Sovelluksen paikallismuistiin tallennetaan käyttäjän valitsemat asetukset, verenluovutuskerrat, Veripalvelun etäpalvelimelta ladattava veritilanne ja tiedot siitä, koska Veripalvelutoimipisteet ja veritilanne on viimeksi ladattu. Kuvan 14 mukaisesti jokainen sovelluksen käyttäjän valitsema asetus tallennetaan omana *avain-arvo*parina. Avaimina ovat *group, gender, interval, vibration* ja *sound*. Käyttäjän veriryhmän*, group*-avaimen, arvo on joko *0*, mikäli veriryhmää ei ole vielä asetettu, tai jokin veriryhmistä. Käyttäjän sukupuolen, *gender*-avaimen, arvo on joko *female* tai *male*. Asetuksen, kuinka usein käyttäjä haluaa muistutuksen verenluovutuksesta seuraavan mahdollisen luovutuspäivän jälkeen, *interval*-avaimen arvo on *1*, *2*, *4*, *8*, *12* tai *off*, mikäli muistutukset halutaan kytkeä pois päältä. Asetuksen, haluaako käyttäjä muistutuksen yhteydessä värinähälytyksen, *vibration*-avaimen arvo on joko *on* tai *off*. Asetuksen, haluaako käyttäjä muistutuksen yhteydessä puhelimen yleisen äänimerkin, *sound*-avaimen arvo on niin ikään joko *on* tai *off*.

Käyttäjän verenluovutuskerrat tallennetaan *donations*-avaimena paikallismuistiin JSON-muodossa. JSON, *JavaScript Object Notation*, on yksinkertainen tiedonsiirtomuoto. Koska se perustuu suoraan JavaScriptin tietorakenteeseen, JSON-muotoinen tieto on sellaisenaan sovelluksen käytettävissä. JSON-muotoinen tieto on kuitenkin ennen paikallismuistiin tallentamista muutettava merkkijonoksi, sillä W3C:n määrityksien mukaisesti paikallismuistiin voidaan tallentaa ainoastaan merkkijonon muodossa olevaa tietoa. Paikallismuistista noudettava merkkijono voidaan kuitenkin yhtä lailla muuttaa takaisin JSON-muotoon. Esimerkkikoodissa 15 on esitetty kahta verenluovutuskertaa kuvaavan JSON-olion rakenne, jossa *timestamp* on verenluovutuksen päivämäärän Unix-aikaleima millisekunneissa ja *hemoglobin* on käyttäjän hemoglobiini verenluovutushetkellä numeerisessa muodossa.

```
[
  {
    'timestamp'  : '1325397600000',
    'hemoglobin' : '130'
  },
  {
    'timestamp'  : '1306904400000',
    'hemoglobin' : '124'
  }
]
```

Esimerkkikoodi 17. Verenluovutuskertaa kuvaavan JSON-olion rakenne.

Veripalvelutoimipisteiden tavoin etäpalvelimelta ladataan päivittäin muuttuva veritilannetta kuvaava XML-tiedosto Ajax-tekniikka käyttämällä. Esimerkkikoodissa 18 on esitetty yksi veritilanteen sisältävän XML-tiedoston tietueista.

```
<item>
  <group>AB+</group>
  <state>D</state>
  <description>
    Veriryhmäsi varastotilanne on kohtuullinen. Tulethan
    kuitenkin lähipäivinä lähimpään veripalvelutoimistoon
    tai liikkuvan veripalvelun tilaisuuteen, sillä tuoreita
    verivalmisteita tarvitaan jatkuvasti.
  </description>
</item>
```

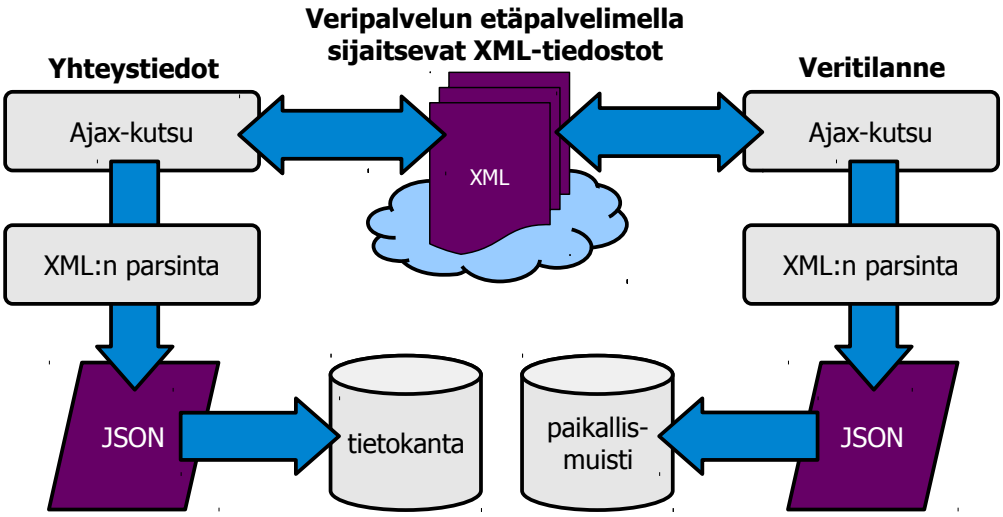
Esimerkkikoodi 18. Veritilannetta kuvaavan XML-tiedoston tietue.

Jokainen veriryhmän tilannetta edustavista tietueista sisältää veriryhmän nimen, tilan ja kuvauksen. Kun XML-tiedostosta parsitaan jokainen veriryhmää edustava tietue yksitellen, lopputuloksena saadaan veritilanne kullekin veriryhmälle erikseen. Parsinnan yhteydessä muodostetaan JSON-olio, johon veritilanteet syötetään. Esimerkkikoodissa 19 on esitetty veritilannetta kuvaavan JSON-olion rakenne. Lopulta verenluovutuskertojen tavoin JSON-olio muutetaan merkkijonoksi ja tallennetaan paikallismuistiin *bloodReserve*-avaimena.

```
[
  {
    'group' : 'AB+',
    'state' : 'D',
    'desc'  : 'Veriryhmäsi tilanne on kohtuullinen.'
  }
]
```

Esimerkkikoodi 19. Veritilanne kuvaavan JSON-olion rakenne.

Kuvassa 15 on havainnollistettu sovelluksen sisällä tapahtuvan tiedon lataamisen ja tallentamisen aikana tapahtuva vuorovaikutus.



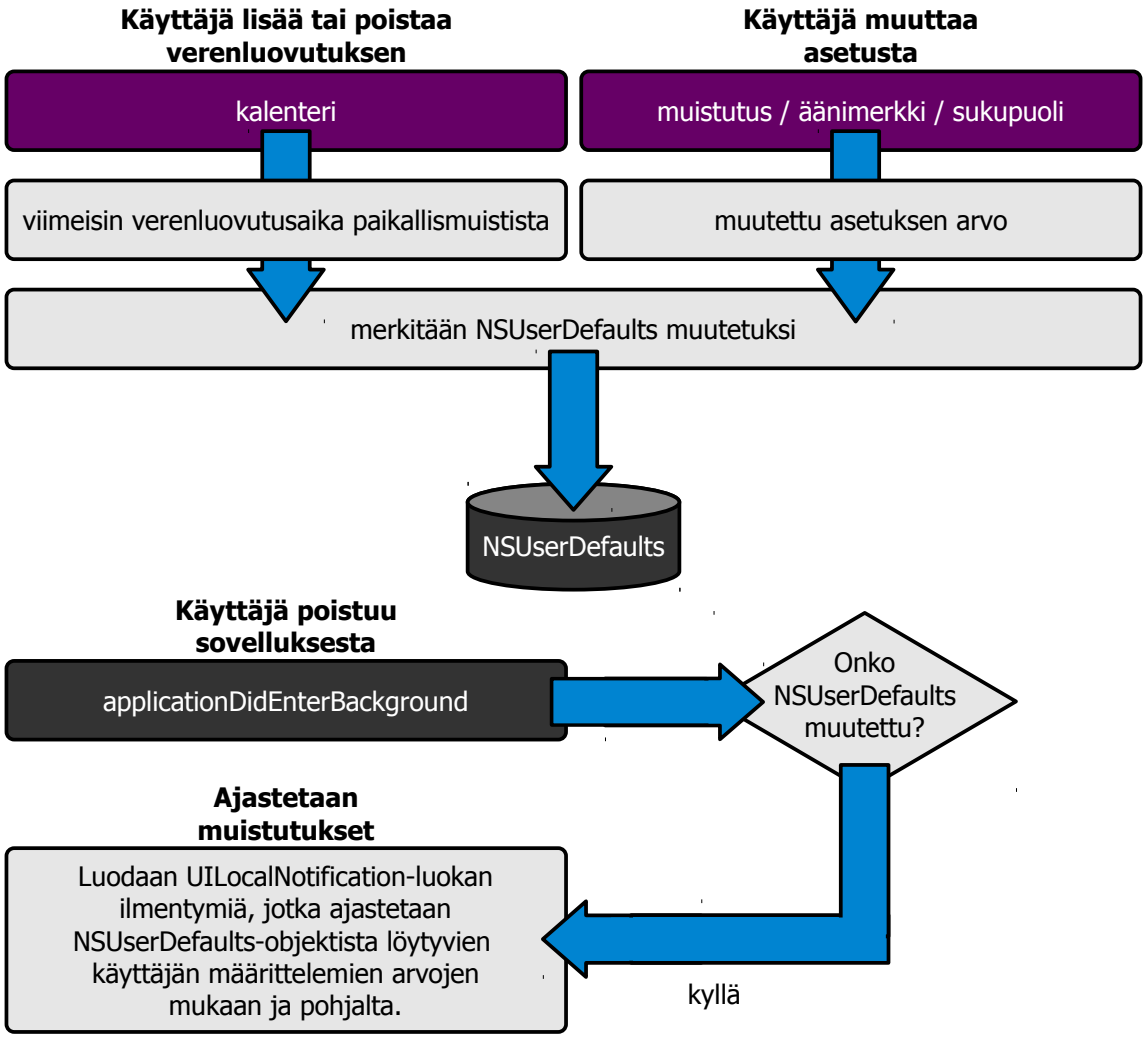
Kuva 15. Sovelluksen tiedon lataamisen ja tallentamisen välinen vuorovaikutus.

Ilmoitusrajapinnan, jolla näytetään käyttäjälle huomiota vaativa ikkuna esimerkiksi datayhteyden ollessa pois päältä, ja edellä esitetyn muistirajapinnan lisäksi sovelluksessa käytetään virallisia PhoneGap-liitännäisiä. Sivulla 38 olevan kuvan 13 ruudun alaosassa oleva välilehtipalkki on toteutettu TabBar-liitännäisellä. Välilehtipalkki luodaan samalla kertaa kuin toimipisteet-sivu lisätään ensimmäisen kerran DOM-puurakenteeseen eli kun käyttäjä ensimmäisen kerran siirtyy kyseiselle sivulle. Tämän jälkeen välilehtipalkki näytetään aina, kun käyttäjä siirtyy toimipisteet-sivulle – mutta ennen kuin varsinainen sivu näkyy käyttäjälle. Samoin välilehtipalkki piilotetaan aina, kun käyttäjä siirtyy pois toimipisteet sivulta päävalikko-sivulle – mutta ennen kuin päävalikko näkyy käyttäjälle. Nämä sivujen siirtymiseen liittyvät toimenpiteet on toteutettu jQuery Mobilen tarjoamien sivuihin sidottujen tapahtumakuuntelijoiden avulla.

Sivulla 35 olevan kuvan 10 kalenteri-sivulla näkyvä päivämäärän valitsin on toteutettu mukautetulla DatePicker-liitännäisellä. Virallinen DatePicker-liitännäinen tarjoaa nimensä mukaisesti ainoastaan päivämäärän valitsimen, kun taas sovelluksessa täytyy käyttäjän pystyä valitsemaan myös oma hemoglobiini. Tämä on toteutettu natiivikoodissa siten, että UIDatePicker-ilmentymä asetetaan UIActionSheet-ilmentymän sisään. Käyttäjän painaessa *kalenteri*-sivulla olevaa *lisää*-nappia UIActionSheet-ilmentymä näytetään käyttäjälle. Päivämäärän valitsemisen jälkeen UIDatePicker- ja UIActionSheet-ilmentymät vapautetaan ja luodan uusi UIActionSheet-ilmentymä, jonka sisään luodaan hemoglobiinille mukautettu UIPickerView-ilmentymä. Hemoglobiinin valitsemisen jälkeen vii-

meinenkin UIActionSheet ja UIPickerView vapautetaan ja JavaScript-puolelle kirjoitetaan takaisin käyttäjän valitsema päivämäärä ja hemoglobiini.

Sovelluksen muistutustoiminnallisuus on toteutettu PhoneGapin applicationPreferences-liitännäisen avulla ja mukauttamalla sovelluksen AppDelegate-natiiviluokkaa. ApplicationPreferences-liitännäinen mahdollistaa olioiden tallentamisen sovelluksen käyttämään muistiin ja niihin helpon pääsyn natiivikoodista käsin. AppDelegate-luokassa on toteutettu UIApplicationDelegate- ja UIWebViewDelegate-protokollien metodit, joista muistutuksen kannalta *applicationDidEnterBackground*-metodi on erityisen tärkeä. Kuvassa 16 on kuvattu muistutuksen toiminnan kannalta tärkeimmät sovelluksessa tapahtuvat toiminnot ja niiden välinen vuorovaikutus.



Kuva 16. Sovelluksen muistutuksen toiminnan tärkeimmät toiminnot ja vuorovaikutukset.

Kun käyttäjä muuttaa sovelluksessa muistutuksen, sukupuolen tai äänimerkin asetusta, tallennetaan muuttunut arvo paikallismuistin lisäksi myös natiiviin NSUserDefaults-objektiin applicationPreferences-liitännäistä käyttäen. Samoin, mikäli käyttäjä lisää tai poistaa verenluovutusmerkinnän, tarkistetaan viimeisin verenluovutuspäivämäärä ja tallennetaan se NSUserDefaults-objektiin. Kun näistä tapahtumista toteutuu edes toinen, NSUserDefaults-objektiin lisätään myös boolean-arvo *true*. Aikanaan, kun käyttäjä poistuu sovelluksesta ja *applicationDidEnterBackground*-metodia kutsutaan automaattisesti sovelluksen puolesta, metodissa ensin tarkistetaan juuri mainittu boolean-arvo. Mikäli sen arvo on *true* eli käyttäjä on joko muuttanut asetuksia tai verenluovutuksiaan, jatketaan metodin suorittamista. Siinä NSUserDefaults-objektista haetaan kaikki objektiin tallennetut tiedot, joiden pohjalta luodaan tarvittava määrä UILocalNotification-luokan ilmentymiä ja ajastetaan ne.

4.5 Testaus ja jatkokehitys

Sovelluksen testaus suoritettiin sovelluskehityksen aikana tapahtuvan oman testauksen ohella Veripalvelun henkilökunnan avulla. Heille annettiin useamman kerran muutamaksi viikoksi kerrallaan käyttöön toisen sukupolven iPod Touch 2G -laite, johon silloinen uusin sovellusversio ladattiin. Testaajana toimineelta henkilökunnalta saadun palautteen mukaan sovellukseen tehtiin kehitysvaiheessa muutoksia. Testilaitteina toimi henkilökunnalle annetun iPod Touch 2G -laitteen lisäksi iPhone 3GS- ja HTC WildFire S -puhelin. Suuri osa testauksesta keskittyi kuitenkin mahdollisten JavaScript-ohjelmavirheiden etsimiseen ja korjaamiseen sekä sovelluksen JavaScript-toiminallisuuden testaamiseen. Näihin käytin enimmäkseen aina uusimpia Safari- ja Chrome-selaimia. Koodin optimointiin käytin apuna verkosta löytyvää *JSPerf*-sivustoa, jonka käyttäjät voivat luoda ja jakaa erilaisia JavaScript-testejä. Testit suoritetaan sivuston palvelimelta, ja niiden tarkoitus on vertailla eri JavaScript-koodien suoritusnopeutta.

Eräs mahdollisena jatkokehityksenä toteuttava sovelluksen lisäominaisuus on verenluovutusajan varausmahdollisuus. Se olisi kytkettynä Veripalvelun nykyisen verkossa tehtävän verenluovutusajan varauspalvelun kanssa. Sovelluksessa käyttäjä antaisi hänelle sopivan päivämäärän, minkä jälkeen sovellus lataisi ja näyttäisi Veripalvelun etäpalvelimelta kyseisen päivän varaustilanteen. Sopivan ajan löydyttyä käyttäjä syöttäisi lomakkeeseen tarvittavat tiedot ja lähettäisi ne palvelimen käsiteltäväksi. Koska tämänhetki-

nen Veripalvelun ajanvarausjärjestelmä on ulkopuolisen tahon ylläpitämä ja näin ollen myös muutokset järjestelmään ovat mahdollisia, mobiilisovelluksessa toteutetun varausjärjestelmän jatkuvaa toimivuutta ei pystyisi takaamaan.

Yhteystietojen löytäminen myös liikkuvan Veripalvelun osalta on yksi mahdollinen jatkokehityksen aihe. Liikkuvia Veripalvelu-toimipisteitä on satoja, joten kaikkien niiden lataaminen suoraan sovellukseen lisäisi tiedonsiirtotarvetta huomattavasti. Jotta lisäominaisuus olisi järkevä toteuttaa, Veripalvelun etäpalvelimelle tulisi luoda palvelinpuolen komentosarja, joka mahdollisesti palauttaisi käyttäjän hakeman paikkakunnan kaikki liikkuvat Veripalvelu-toimipisteet.

Liikkuvien Veripalvelu-toimipisteiden yhdistäminen tämänhetkiseen muistutusominaisuuteen ja käyttäjän paikallistamiseen tai käyttäjälle tärkeisiin osoitteisiin oli hyvä idea pitää käyttäjä tietoisena lähellä sijaitsevista verenluovutuspaikoista. Tämä tarkoittaisi kuitenkin arkaluontoisen tiedon – paikkatieto ja kotiosoite tai muuta sellaista – tallentamista tai välittämistä käyttäjästä.

Sovellus julkaistaan Applen AppStoressa. Ennen julkaisua sovellusta testataan kuitenkin vielä Applen tarjoamalla Ad Hoc -testausmenetelmällä, jossa sovelluspaketti jaetaan suljetun testiryhmän käyttöön. Sovelluksen asentaminen muihin kuin ennalta määriteltyihin laitteisiin ei ole mahdollista. Ad Hoc -testauksella saadaan laaja testaajakunta, ja siitä tulevan palautteen pohjalta voidaan vielä vaikuttaa lopulliseen AppStoreen julkaistavaan sovellukseen.

4.6 Sovelluksen rajoitukset ja haavoittuvuus

Vaikka PhoneGap tarjoaa varsin hyvän ohjelmistokehyksen alustariippumattomaan sovelluskehitykseen, sillä on myös rajoituksensa. Yleisesti ottaen PhoneGap on rajoittunut siihen, mitä natiiviohjelmointikehityssarja ja sovelluksen alla oleva WebView-komponentti mahdollistavat. Koska WebView-komponenteissa on eroavaisuuksia eri mobiilialustojen kesken, myös tuki HTML5- ja CSS3-ominaisuuksille on eriävä.

Koska mobiilialustojen JavaScript-moottori on yksisäikeinen, on lukkiutuminen resurssien jatkuvan varaamisen vuoksi mahdollista. Lukkiutumisen estämiseksi PhoneGapin ohjelmointirajapinnoissa käytetään takaisinkutsujafunktioita. Asynkronisuuden takia

useat sisäiset takaisinkutsujafunkiot voivat olla hankalia toteuttaa ja vaikeita tulkita. (Castledine ym 2011: 200.)

Avain kuten kaikilla verkkosivuilla, myös PhoneGapilla on tietoturvaongelmia. Koska PhoneGapin toteuttamisessa on käytetty apuna muiden tekemiä ohjelmointikirjastoja, se on haavoittuvainen käytetyistä kirjastoista mahdollisesti löytyville tietoturvaongelmille.

Mobiilialustoista pyritään aina tekemään käyttäjälle mahdollisimman turvallisia. Sama koskee sovelluksia. Käyttäjä turvataan kaikin mahdollisin keinoin haitallisilta sovelluksilta. Tämä ei ole kuitenkaan molemminpuolista. Mobiilialustan ja laitteen turvallisuustekijät eivät turvaa sovellusta haitalliselta käyttäjältä. PhoneGapissä ei ole mahdollista täysin turvata sovelluksen sisältöä. Tämä tarkoittaa, että kuka vain, jolla on sovellus asennettuna laitteeseen, voi pienellä vaivalla purkaa sovelluspaketin ja päästä käsiksi sovelluksen sisältöön. Tämä ilmiö ei esiinny ainoastaan PhoneGapissä vaan on yleismaailmallinen ongelma sovelluskehityksessä. Koska PhoneGapin avulla kehitetyn sovelluksen HTML-, CSS- ja JavaScript-tiedostoja ei käännetä rakentamisen aikana, sovelluspaketin purkamisen jälkeen koko *www*-kansion sisältö on luettavissa selkokielisenä. Natiivikoodi sen sijaan on purkamisen jälkeen vielä käännettynä ja sen tulkkaaminen vaatikin hieman enemmän ponnisteluja.

Koska PhoneGapin toiminta perustuu WebView-komponenttiin ja sen mahdollisuuteen saada yhteys natiivikoodiin JavaScript-ympäristöstä, laitteen verkkoselaimen turvallisuutta edistävään *sandbox*-mekaniikasta tulee haavoittuvainen. Tämä käytännössä altistaa kaikki PhoneGapin tarjoamat ohjelmointirajapinnat minkä vain haitallisen ulkopuolisen verkkosivun ulottuville. PhoneGapissä on kuitenkin toteutettu turvallisuusmalli, jolla hallitaan ulkopuolisiin domaineihin pääsyä. Esimerkiksi, jotta sovelluksesta saisi yhteyden Googlen palvelimille, kehitysvaiheessa täytyy Googlen domain, www.google.com, lisätä sovelluksen asetuksiin.

Arkaluontoisen tiedon – puhelimen osoitekirjan, paikkatietojen tai muuta sellaista – karkaaminen vääriin käsiin on mahdollista lähinnä vain itse sovelluskehittäjän aikaan saamana tahallisena haitallisen sovelluksen kehittämisenä. Mikäli sovelluksessa haetaan tai välitetään tietoa ulkopuoliselle etäpalvelimelle, jonka turvallisuudesta tai tarkoitus-

perästä ei ole takeita, arkaluontoisen tiedon leviämien etäpalvelimen haitallisen koodin takia on kuitenkin mahdollista.

Veripalvelulle toteutetussa mobiiliverkkosovelluksessa ei ole kuitenkaan käytetty ulkopuolisen tahon ylläpitämää etäpalvelinta, josta tietoa haetaan. Sovelluksessa ei myöskään ole mahdollista ladata ulkoista mahdollisesti haitallista verkkosivua tai komentosarjaa, jonka avulla ulkopuolinen taho pääsisi käsiksi sovelluksen tai laitteen tietoihin. Sovelluksessa ei myöskään käsitellä tai tallenneta käyttäjän kannalta arkaluontoisia tietoja. Mahdollinen lähdekoodin kääntäminen ja tulkkaaminen ei myöskään paljasta mitään Veripalvelun toimintaa uhkaavia tietoja.

5 Päätelmiä

Suomen Punaisen Ristin Veripalvelulle toteutettu mobiiliverkkosovellus oli ensimmäinen PhoneGapillä toteuttamani sovellus. Kehitysvaiheessa en juurikaan törmännyt PhoneGapin takia ongelmiin. Tämä osaksi varmasti sen takia, että käytin sovelluksessa PhoneGapin tarjoamista natiivirajapinnoista ainoastaan muisti- ja ilmoitusrajapintaa. PhoneGapillä olikin sovelluksen kehityksessä lähinnä vain paketoijan rooli. Siinä roolissaan se suoriutui moitteitta.

Ainoan haasteen PhoneGap antoi testauksen osalta. Kun sovelluksen HTML-, CSS- ja JavaScript-ohjelmointikielillä kirjoitettujen sovelluksen toiminallisuuksia voi hyvin testata tietokoneen verkkoselaimissa ja niiden mukana tulevien erilaisten testaustyökalujen avulla, PhoneGapin tarjoamien rajapintojen toiminnallisuuksia voi testata ainoastaan laitteessa. Tämä hidastaa huomattavasti sovelluksen ohjelmavirheiden korjaamista.

Koska PhoneGapin avulla paketoitu sovellus ajetaan WebView-komponentissa ja käyttöliittymä toteutetaan HTML- ja CSS-ohjelmointikielellä, sovellus ei erilaisten mobiilialustojen verkkoselainmoottoreiden HTML- ja CSS-toteutuksien takia välttämättä toimi kaikissa laitteissa samalla tavalla. Eritoten Androidin useiden eri versioiden kesken HTML- ja CSS-tuki on osittain erilainen.

Yksittäisen sovelluksen ohjelmointi yhdelle mobiilialustalle PhoneGapin mahdollistamien uusimpien verkkoteknologioiden, kuten HTML5:n ja CSS3:n sekä JavaScriptin, avulla ei välttämättä ole kuitenkaan sen nopeampaa kuin natiiviohjelmointi. PhoneGapin varsinainen vahvuus tuleekin esille vasta, kun sovelluksen haluaa levittää useammalle mobiilialustalle. Tällöin sovelluksesta riippuen suuren osan koodista voi hyödyntää myös muiden alustojen sovelluskehityksessä.

Käytin sovelluksen käyttöliittymän toteuttamisessa jQuery Mobile -ohjelmistokehystä, joka alkuun tuntui todella hienolta, kätevältä ja toimivalta. Myöhemmin kuitenkin kaduin päätöstä. jQuery Mobilella rakennetun käyttöliittymän raskaampi muokkaaminen niin HTML-, CSS- kuin JavaScript-koodista käsin on haasteellista. jQuery Mobile luo ehostettujen HTML-elementtien aikaan saamiseksi ajon aikana useita erilaisia sisäkkäisiä HTML-elementtejä, joilla lähes kaikilla on eri tyyliluokat. Muokkaamista varten oikean elementin ja tyyliluokan löytäminen on työlästä. Havaitsin sivujen siirtymisen aika-

na esitettyjen animaatioiden epäkäytännöllisyyden, mikäli siirryttävällä tai siirretyllä sivulla oli yli ruudun pituudelta HTML-elementtejä. Tällöin siirtymäanimaatioissa etenkin Android 2.3. -version puhelimessa esiintyi ylimääräistä nykimistä ja välähtelyä. iOS 4.3 -version iPhone 3GS -puhelimessa siirtymät toimivat hyvin.

Sovelluksen saaminen Applen AppStoreen vaatii Apple Development Program -kehityslisenssin hankkimisen. Koska toteutin sovelluksen Suomen Punaisen Ristin Veripalvelulle, kehityslisenssi tuli Veripalvelun puolesta. Sovellus tuli myös linkittää ja allekirjoittaa kehityslisenssiin kuuluvalla salausavaimella. Salausavaimet hallinnoidaan Applen kehittäjäportaalista käsin. Tämä toimenpide tehtiin yhdessä Veripalvelun edustajan kanssa. Testiversioon ja AppStoreen pyrkivän sovelluksen versioon täytyi luoda omat salausavaimet. Näiden eri avaimien hallinta ja sovelluksen kääntämiseen linkittäminen Xcode-ohjelmalla oli myös ensikertalaiselle hieman monimutkainen prosessi.

On kuitenkin todettava, että käytti PhoneGapia sitten ainoastaan mobiiliverkkosovelluksen paketoimiseen tai myös PhoneGapin tarjoamien natiivirajapintojen takia, sen hyödyntämisen tuomat edut alustariippumattomassa sovelluskehityksessä ovat ilmeiset. PhoneGap ei ainoastaan nopeuta itse sovelluskehitystä vaan myös tarjoaa natiiviohjelmointia huomattavasti loivemman oppimiskäyrän PhoneGap-sovelluskehityksessä käytettyjen HTML-, CSS- ja JavaScript-ohjelmointikielien takia.

Lähteet

Abdullah, Shazron. 2011. History of PhoneGap. Verkkodokumentti. <phonegap.pbworks.com>. July 12, 2011. Luettu 20.5.2012.

Allen, S., Graupera, V. & Lundrigan, L. 2010. Pro Smartphone Cross-Platform Development: iPhone, Blackberry, Windows Mobile and Android Development and Distribution. New York: Apress.

API Reference. 2012. Verkkodokumentti. Adobe. <http://docs.phonegap.com/en/1.3.0>. Luettu 20.5.2012.

Balaz, A., Bartels, B., Bieh, M., Bloor, R., Brady, C., Getzmann, P., Graf, O., Gülle, R., Harty, J., Iliescu, O., Johnson, A., Johnson, G., Kapetanakis, M., Koch, M., Messerschmidt, T., Nowak, P., Repty, A., Rouffineau, T., Schmidt, A., Shuqair, M., Tabor, M., Virkus, R. & Weller, J. 2011. Mobile Developer's Guide To The Galaxy. Bremen: Enough Software GmbH + Co.

Castledine, E., Eftos, M. & Wheeler, M. 2011. Build Mobile Websites and Apps for Smart Devices. Collingwood: SitePoint Pty.

Constantinou, A., Kapetanakis, M., Schuermans, S., Vakulenko, M. 2011. Mobile Platforms: The Clash of Ecosystems. Verkkodokumentti. VisionMobile Ltd. <www.visionmobile.com/rsc/researchreports/VisionMobile-Clash-of-Ecosystems_v1.pdf>. November 2011. Luettu 6.4.2012.

Gartner Says Worldwide Smartphone Sales Soared in Fourth Quarter of 2011 With 47 Percent Growth. 2012. Verkkodokumentti. Gartner, Inc. <www.gartner.com/it/page.jsp?id=1924314>. February 15, 2012. Luettu 6.4.2012.

Ghatol, Rohit & Patel, Yogesh. 2012. Beginning PhoneGap: Mobile Web Framework for JavaScript and HTML5. New York: Apress.

Gowel, Scott & McWherter, Jeff. 2012. Professional Mobile Application Development. Indianapolis: John Wiley & Sons.

Kosmaczewski, Adrian. 2012. Mobile JavaScript Application Development. Sebastopol: O'reilly Media.

MacFadyen, Jesse. 2009. PhoneGap for iPhone exposed. Verkkodokumentti. Adobe Systems, Inc. <www.phonegap.com/2009/11/04/phonegap-for-iphone-exposed>. 04 Nov 2009. Luettu 20.5.2012.

Trice, Andrew. 2012. Extending PhoneGap with native plugins for Android. Verkkodokumentti. Adobe Systems, Inc. <www.adobe.com/devnet/html5/articles/extending-phonegap-with-native-plugins-for-android.html>. 23 April 2012. Luettu 20.5.2012.

Wargo, John M. 2012. PhoneGap Essentials: Building Cross-Platform Mobile Apps. Crowfordsville: Pearson Education.