ANDROID-SOVELLUKSEN KÄÄNTÄMINEN PHONEGAP-MONIALUSTASOVELLUKSEKSI

HAMK
HÄMEEN AMMATTIKORKEAKOULU

Ammattikorkeakoulun opinnäytetyö

Tietojenkäsittelyn koulutusohjelma

Visamäki, kevät 2015

Saku Rekola

TIIVISTELMÄ

HAMK
HÄMEEN AMMATTIKORKEAKOULU

Visamäki
Tietojenkäsittelyn koulutusohjelma

Tekijä Saku Rekola **Vuosi** 2015

Työn nimi Android-sovelluksen kääntäminen PhoneGap-monialustasovellukseksi

TIIVISTELMÄ

Opinnäytetyön toimeksiantaja oli Ambientia Oy. Työn tavoitteena oli kääntää olemassa oleva LVI-Numero Oy:lle tehty Android-sovellus iOS- ja Windows Phone 8 -alustoille.

Kääntämiseen käytetään Apache Cordovaa, joka mahdollistaa mobiilisovelluksen luonnin Web-tekniikoita käyttäen natiivisovelluskehyksen sisään. Opinnäytetyön tavoite oli selvittää, onko hybridisovelluskehitys jo varteenotettava vaihtoehto perinteisille mobiilisovelluksille ja selvittää, millaisia haasteita useammalle alustalle samanaikaisesti kehitettävä sovellus aiheuttaa. Työssä käytettiin Cordovan kanssa AngularJS-sovelluskehystä.

Teoriaosuudessa esitellään JavaScriptin lisäksi yleisimpiä Node.js-pohjaisia front-end-kehitystyökaluja (Yeoman, Grunt ja Bower) joiden avulla projektinhallintaa helpotetaan. Varsinaisen ohjelmakoodin tuottamisen lisäksi kirjoitettiin vielä unit- eli yksikkötestit sekä end-to-end- eli skenaariotestit.

Lähteinä opinnäytetyössä käytettiin lähinnä verkkomateriaaleja, JavaScript- ja Node.js-osuuksissa myös kirjoja.

Opinnäytetyössä selvisi, että Android- ja iOS-sovellukset toteutuvat vaivattomasti Cordovan avulla, mutta Windows Phonen kanssa törmättiin ongelmiin, jotka jäivät osittain ratkaisematta. Sovelluksessa keskityttiin ulkonäön sijaan sovelluslogiikkaan, eikä sovellus ole valmis julkaistavaksi sovelluskaupoissa. Kohdealustojen lisäksi sovellus toimii myös Androidilla.

Avainsanat Apache Cordova, PhoneGap, monialustakehitys, iOS, Windows Phone 8

Sivut 27 s.

HAMK
UNIVERSITY OF APPLIED SCIENCES

Visamäki
Degree Programme in Business Information Technology

Author Saku Rekola **Year** 2015

Subject of Bachelor's thesis Porting an Android application to a cross-platform application with PhoneGap

ABSTRACT

The purpose of the thesis commissioned by Ambientia Oy was to port an existing Android application called LviDroid to a cross-platform application using Apache Cordova. The target platforms for this thesis were Windows Phone 8 and iOS. Apache Cordova allows a developer to use web techniques instead of platform specific code to build a mobile application.

The goal of this thesis was to find out if Cordova is a viable option to be a contender for native applications and to find out what challenges multi-platform development poses. AngularJS JavaScript framework was used together with Cordova in this project.

The theory part contains information about Apache Cordova and its relationship with PhoneGap. It also explains how AngularJS differs from traditional JavaScript and presents some of the most popular Node.js based tools (Yeoman, Grunt and Bower) that are used to manage a project workflow. In addition to the actual application code, unit and end-to-end tests were also written.

The source material comprised mainly of various Internet articles and guides since the used techniques are still evolving rapidly and no up-to-date literature was available. In the JavaScript and Node.js sections books were also used.

The result of the thesis shows that Cordova can be used to develop Android and iOS applications with relative ease. Problems arose with Windows Phone 8, some of which remain unsolved. The application focused on replicating all the functionality of the original Android app and is not ready to be released in app stores. The source code possessed by the commissioner could be changed to meet the different platform specific requirements with further development. The application also works well on Android.

Keywords Apache Cordova, PhoneGap, hybrid mobile app, cross-platform development, iOS, Windows Phone 8

Pages 27 p.

Sanasto:

AngularJS Googlen adoptoima JavaScript MVW -sovelluskehys. Sovelluskehys eli framework eroaa kirjastosta pakottamalla kehittämään ohjelman tietyn kaavan mukaisesti.

Apache Cordova kokoelma laitteistorajapintoja, jotka mahdollistavat mobiililaitteiden hardware-komponenttien käytön JavaScriptin avulla. Open source -versio PhoneGapista.

API (application programming interface) ohjelmointirajapinta. Määrittelee, miten eri ohjelmakomponenttien tulee olla vuorovaikutuksessa toisiinsa.

Async (asynchronous) synkronoimaton prosessointi mahdollistaa muun koodin suorittamisen samalla kun odotetaan pitkään operaatioon vastausta. Yleisin käyttökohde on ottaa async-yhteys serveriin ja prosessoida vastaus, kun se sattuu tulemaan takaisin.

BDD (Behaviour Driven Development) jatkaa TDD-mallia tuomalla mukaan käyttäjätarinat ja ohjeistuksen erilaiseen ajattelumalliin ja selkeyttämällä nimeämiskäytäntöjä.

Bower Noden NPM-paketinhallintaa täydentävä paketinhallintajärjestelmä. Hoitaa bower.json-tiedostossa määriteltyjen tiedostojen latauksen ja päivittämisen. Integroituu muihin työkaluihin, käytetään esimerkiksi Gruntin kanssa automatisoituun JavaScript-kirjastojen ja HTML-dokumenttien linkittämiseen.

Browserify mahdollistaa moduulien hallinnan selaimessa tarjoamalla Nodesta tutun require metodin. Katso myös RequireJS.

CLI (Command Line Interface) Komentoliittymä, komentorivillä käytettävä käyttöliittymä. Cordova ja Node tarjoavat omat komentotulkkia laajentavat syntaksinsa.

CSS (Cascading Style Sheets) tyyliohjeiden laji. CSS-tiedostoilla kerrotaan selaimelle, miten sivu tulee esittää.

DOM (Document Object Model) alusta- ja ohjelmointikieliriippumaton tapa esittää XML, HTML ja XHTML dokumenttien sisältöä. Mahdollistaa myös sisällön muokkauksen oman API:nsa kautta.

FIFO (First In, First Out) -lista on tiedon järjestys- ja manipulaatiotapa, jossa vanhin (ensimmäinen) lisäys prosessoidaan ensin. Tieto käsitellään siinä järjestyksessä, missä se saapuu jonoon. Pitkät operaatiot pidentävät jonoa ja saattavat aiheuttaa näkyvää viivettä ohjelman toiminnassa.

Front-end Web-sovelluksen käyttäjälle näkyvä osa, eli verkkosivu. Käyttöliittymän lisäksi sisältää yleensä myös jonkin verran toimintalogiikkaa sujuvan kokemuksen takaamiseksi. Vastakohtana backend, eli käytännössä serveri, jonka kanssa ohjelma kommunikoi.

Grunt ”The JavaScript Task Runner”. Automatisoi toistuvia tehtäviä. Ohjelmoitavissa JavaScriptillä, laajennettavissa plugineilla. Suosittu vaihtoehto Gruntille on Gulp.

IIFE (immediately-invoked function expression), itseään kutsuva funktio. Mahdollistaa funktion sisäisen koodin ajamisen erillään global scopesta ja hieman OOP-mallisen kehityksen.

iOS (entinen iPhone OS) Applen mobiilikäyttöjärjestelmä, jota käytetään ainoastaan iPhone- ja iPad-laitteissa.

JavaScript Engine Node.js yhteydessä törmäsin termiin, tarkoittaa JavaScript-tulkkia, joka kääntää koodin tietokoneen ymmärtämään muotoon. Verrattavissa Javan JVM:ään.

MVW (Model-View-Whatever) AngularJS julisti vuonna 2012 olevansa MVW-ohjelmistokehys, jossa ”Whatever” tarkoittaa ”whatever works for you”.

Node.js Chromen V8 JavaScript-tulkin päälle rakennettu alusta, joka mahdollistaa skaalautuvan serveripuolen toteutuksen JavaScriptillä. Nodea käytetään myös front-end kehityksessä mm. työkalujen asentamiseen NPM paketinhallintajärjestelmällä.

NPM (Node Package Manager) npm asentaa paketoidut node moduulit ja niiden riippuvuudet puolestasi. Esimerkiksi Cordova suosittelee käyttämään ”npm install”-komentoa manuaalisen latauksen sijaan.

PhoneGap on Adoben jakeluversio Apache Cordovasta. Suurin ero löytyy komentorivityökalun komennoista sekä Adoben tarjoamista osittain maksullisista lisäpalveluista.

RequireJS on JavaScript tiedostojen ja moduulien hallintaan kehitetty työkalu.

REST (Representational State Transfer) on HTTP-protokollaan perustuva arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen. Käytännössä mahdollistaa tietokantojen käytön HTTP-kutsuilla.

Scope JavaScriptissä on kaksi scopea: global ja local. Funktion ulkopuolella alustetut muuttujat menevät automaattisesti global scopeen ja näkyvät koko ohjelman laajuisesti. Funktion sisällä alustetut muuttujat taas elävät local scopessa eli ne luodaan ja tuhotaan joka kerta kun funktio kutsutaan, eivätkä ne näy funktion ulkopuolelle.

SPA (Single-page application) verkkosivu joka lataa kaiken HTML, CSS ja JavaScript koodin joko kerralla tai dynaamisesti käyttäjän toimien perusteella. Tällaisten verkkosivujen tarkoitus on tarjota sujuva, työpöytäohjelman tapainen käyttäjäkokemus.

TDD (Test Driven Development) ohjelmointitapa, jonka mukaan ohjelman kirjoitus aloitetaan kirjoittamalla ensin automatisoitu testi. Tarkoituksena helpottaa ohjelmakoodin ylläpitoa ja huomata kun jokin ominaisuus menee rikki.

Two-way data binding tarkoittaa yleisesti MVVM-arkkitehtuurissa sitä, että View reagoi Modelissa tapahtuviin muutoksiin ja Model vastaavasti Viewin muutoksiin.

UI (User Interface) käyttöliittymä.

Yeoman Työkalu, jolla voi generoida nopeasti erilaisia sivupohjia. Tutustuin tämän työkalun kautta Gruntiin ja Boweriin.

Porttauksella (porting) tarkoitetaan sovelluksen kääntämistä alustalta toiselle, eli eri ajoympäristöön kuin millä se alun perin suunniteltiin käytettäväksi.

Windows Phone 8 Microsoftin mobiilikäyttöjärjestelmän kolmas sukupolvi. Sisältää saman Windows NT kernelin kuin Windows 8. Windows Phone 8.1 päivityksen myötä tuli mahdolliseksi kehittää ”Universal App”-nimellä Windows Store -sovelluksia kaikille Windows-laitteille.

SISÄLLYS

1 JOHDANTO

Opinnäytetyön toimeksiantajana toimii Ambientia Oy. Työn tavoitteena on kääntää olemassa oleva LVI-Numero Oy:lle kehitetty Android-sovellus iOS- ja Windows Phone 8 -alustoille. Kääntämiseen käytetään Apache Cordovaa, joka mahdollistaa mobiilisovelluksen luonnin Web-tekniikoita käyttäen natiivitoteutuksen sijaan. Käytännössä Cordova mahdollistaa yhteisen koodipohjan käytön eri alustoilla ja laskee ylläpitokustannuksia, kun jatkokehitystä ei tarvitse tehdä jokaisella alustalla erikseen. Myös päivitykset saadaan julkaistua eri alustoille samaan aikaan.

Opinnäytetyössä käytetään Cordovan kanssa AngularJS- ja Bootstrap-frameworkeja. Bootstrap on suosittu CSS-framework, joka mahdollistaa responsiivisten verkkosivujen kehittämisen valmiiden komponenttien avulla [Bootstrap, 2015]. Bootstrapin sijaan opinnäytetyössä esitellään opinnäytetyön yhteydessä käyttöön otettuja yleisimpiä Node.js-pohjaisia front-end-kehitystyökaluja (Yeoman, Grunt ja Bower) joiden avulla projektin ja sen riippuvuuksien hallintaa pyritään helpottamaan. [Yeoman, 2015]

Varsinaisen ohjelmakoodin tuottamisen lisäksi kirjoitetaan vielä unit- eli yksikkötestit sekä end-to-end- eli skenaariotestit. Testien yhteydessä esitellään pinnallisesti Jasmine-testausframeworkia. [Jasmine framework, 2014]

Rakenteeltaan sovellus on suhteellisen yksinkertainen, sen tarvitsee vain käyttää olemassa olevaa REST-rajapintaa lvi-numeroiden hakemiseen sekä lukea viivakoodeja ja lähettää tuotetiedot haluttaessa sähköpostilla. Osoitteessa www.lvi-info.fi on nettisivut, joilla lvi-numeroiden haku onnistuu selaimella. Sovelluksen toteutukseen käytetystä Apache Cordovasta itselläni on kokemusta yhden aiemman projektin verran. Päätin kokeilla siipiäni ja yhdistää tähän sovellukseen mahdollisimman monia itseäni kiinnostavia työkaluja, joihin ei aiemmin ollut aikaa tutustua. AngularJS-framework valikoitui muiden suosittujen JavaScript MVC-frameworkien joukosta selkeimmän tutoriaalin ja koodin testauksen painottamisen takia. Opinnäytetyössä käytän Ionic Framework -tiimin kehittämää ngCordova nimistä AngularJS-moduulia, joka käärii tukun Cordova liitännäisiä AngularJS-direktiivien sisään.

Android-ohjelmoinnin osaaminen auttaa hahmottamaan mitä alkuperäinen sovellus tekee. Välttämätöntä Android-osaaminen tosin ei ole käännöstyön kannalta. Tässä opinnäytetyössä keskitytään JavaScript-ohjelmointikieleen ja Apache Cordova -natiivisovelluskehykseen, joka on PhoneGapin open source -ydin. Tämän lisäksi tutustutaan Node.js-alustaan sekä yleisimpiin Noden päälle rakennettuihin front-end-työkaluihin. JavaScript-osiossa tarkastellaan lähemmin AngularJS MVC-sovelluskehystä.

Opinnäytetyö vastaa kysymykseen, mitä eroa on hybridi- ja natiivisovelluskehityksellä, sekä tutkii, millaisia haasteita useammalle alustalle samanaikaisesti kehitettävä sovellus aiheuttaa. Lisäksi pyritään selvittämään, onko monialustasovelluskehitys Cordovan avulla jo varteenotettava vaihtoehto perinteisille natiivisovelluksille.

Opinnäytetyön lähdemateriaalina käytettiin pääasiassa verkkosivuja, sillä työkalujen ripeästä kehitystahdista johtuen ajan tasalla olevaa kirjallisuutta ei juurikaan löydy. Node.js-alustan kohdalla sääntö vahvisti poikkeuksen, ja JavaScript ohjelmointikielestä kirjoja löytyy runsaasti, joskin työssä käytetään vain yhtä.

Opinnäytetyö tarjoaa uutta tietoa monialustasovelluksen kehittämisprosessista ja erityisesti useammalle alustalle samanaikaisen kehittämisen aiheuttamista ongelmista. Suurin osa saatavilla olevasta PhoneGap-lähdemateriaalista keskittyy vain yhteen alustaan kerrallaan, eikä ota kantaa monialustakehityksen mukanaan tuomiin haasteisiin. Tässä työssä käsiteltävä sovellus ei valmistunut julkaisukuntoiseksi. Suurimmat haasteet kohdattiin Windows Phone 8 -alustalla, jonka kanssa tuntui esiintyvän ongelmia sekä Cordova- että AngularJS-sovelluskehysten käytön kanssa.

2 APACHE CORDOVA

Apache Cordova (http://cordova.apache.org/) on kokoelma laiterajapinnoista, joiden avulla mobiilikehittäjä pystyy käyttämään laitteen toiminnallisuuksia kuten kameraa tai kiihtyvyysmittaria suoraan JavaScript koodista ilman alustakohtaisen ohjelmointikielen opettelua [Cordova projekti, 2014]. Cordova projektin ylläpitämien virallisten rajapintojen lisäksi kehittäjä voi laajentaa sovelluskehyksen toimintaa kirjoittamalla itse alustakohtaisia liitännäisiä ja julkaista niitä muiden käyttöön. Kolmannen osapuolen liitännäisiä löytyy projektin viralliselta Cordova Plugin Registry -verkkosivulta kirjoitushetkellä 238 kappaletta, joten ennen oman liitännäisen kirjoittamista kannattaa tarkistaa löytyykö haluttu toiminto jo toteutettuna. Virallisen sivun lisäksi liitännäisiä löytyy ainakin Telerikin ylläpitämästä Verified Plugins Marketplace sekä PlugReg.com-sivustosta. Sivut saattavat sisältää osittain päällekkäisiä listauksia Cordova liitännäisistä. [Telerik Verified Plugins Marketplace, 2014; Cordova plugin registry, 2014]

Yhdistettynä johonkin UI-kirjastoon Cordova mahdollistaa natiiviohjelmaa muistuttavan älypuhelinsovelluksen kehittämisen pelkästään web-tekniikoita hyödyntäen. Cordovan JavaScript rajapinnat ovat yhtenevät jokaisella tuetulla laitealustalla, joten monialustakehityksen pitäisi onnistua yhdellä koodipohjalla. Käytännössä tämä on mahdollista vain, mikäli käyttää Cordova projektin ylläpitämiä liitännäisiä jotka on kirjoitettu jokaiselle alustalle yhteensopivaksi, kolmannen osapuolen liitännäisten kanssa usein joutuu etsimään jokaiselle alustalle oman toteutuksen. Natiivisovellusmaisen ulkoasun toteutus eri alustoille aiheuttaa omat ongelmansa sovelluskehitysprosessiin, mikäli Android- ja iOS-alustojen lisäksi haluaa julkaista myös Windows Phone -sovelluksen. Pelkän webview-ohjelmistokomponentin lisäksi Cordovan kanssa voidaan käyttää myös muita natiivikomponentteja. Esimerkiksi Appgyverin Steroids yhdistää natiiveja UI-elementtejä Cordova kehitykseen. [Appgyver, 2014; Schinsky, 2014]

Perinteisestä verkkosivusta poiketen Cordova-sovellus tallennetaan natiivisovelluskehyksen sisään itse laitteeseen, eikä sitä ladata verkosta. Ulkoiselta palvelimelta sovelluksen sisällön voi toki myös ladata, mutta se saattaa vaikuttaa ohjelman nopeuteen sekä aiheuttaa käyttökatkoja tilanteissa joissa mobiililaite ei ole yhteydessä verkkoon. Tietoturvan ja sovelluskaupoissa julkaisun kannalta sisällön haku ulkoiselta palvelimelta saattaa tosin aiheuttaa ongelmia.

Cordova-sovellukset paketoidaan natiiviohjelmien tapaan käyttäen alustakohtaisia kehitystyökaluja. Käytännössä tämä tarkoittaa, että Windows Phone -kehitykseen tarvitsee Windows 8 -käyttöjärjestelmän ja iOS-kehitykseen OS X:n. Lisäksi alustakohtaiset kehitystyökalut (SDK) pitää olla asennettuna paikallisesti. Microsoft on kuitenkin huomannut asiassa ongelman, ja tarjoaa virallisia ohjeita Windowsin virtualisointiin Macilla [Virtualisointi MSDN, 2014]. OS X:n lisenssiehdot taas kieltävät käyttöjärjestelmän virtualisoinnin muulla kuin Applen laitteilla, käytännössä pakottaen kehittäjän ostamaan Macin.

Cordovalla kehitetty sovellus voidaan julkaista sovelluskaupoissa natiiviohjelmien tapaan, mikäli se täyttää alustakohtaiset vaatimukset. Sovelluksen käyttäjän kannalta ei ole eroa, millä tekniikalla mobiilisovellus on kehitetty. Cordova tukee kaikkia yleisimpiä laitealustoja. Opinnäytetyön aikana Windows Phone 7 tuki loppui ja sen tilalle tuli tuki Windows 8 ja 8.1 store appseille sekä Windows universal appseille. Ajankohtainen listaus Cordovan tukemista alustoista löytyy osoitteesta http://cordova.apache.org/docs/en/edge/guide_support_index.md.html#Platform%20Support.

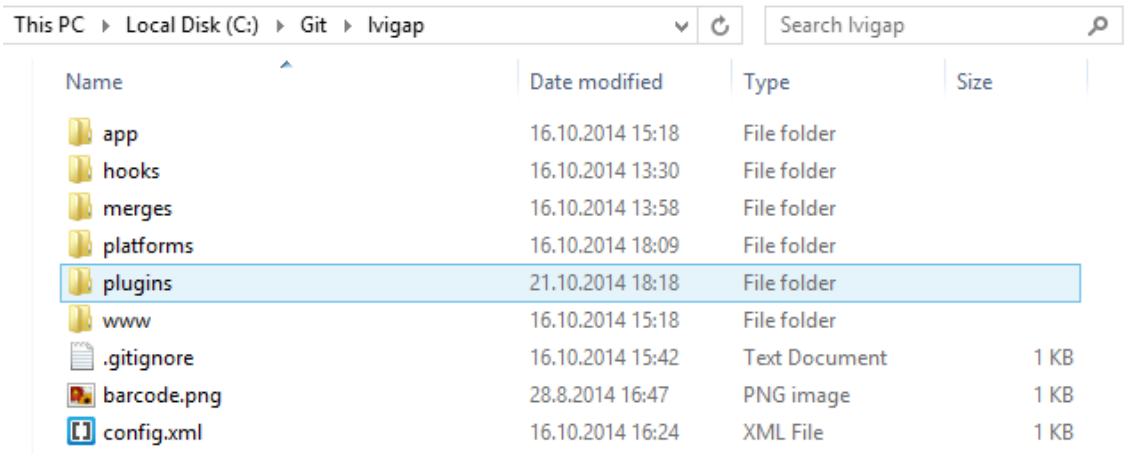
2.1 Historia

Vuoden 2008 lopussa Nitobi niminen startup-yritys julkisti open source projektin nimeltä PhoneGap. PhoneGap mahdollisti natiivin iPhone sovelluksen kehittämisen ilman Objective-C -kielen opettelua käyttämällä sisäänrakennettua web-selainta [PhoneGap blogi, 2008].

Vuonna 2011 Adobe osti Nitobin ja oikeudet PhoneGap brändiin. PhoneGap lähdekoodi annettiin Apache Software Foundationille [ASF, 2014], jossa se uudelleennimettiin aluksi Callbackiksi ja myöhemmin Cordovaksi, kadun mukaan jolla Nitobin toimisto sijaitsi Vancouverissa. [Chow, 2011]

Alusta lähtien PhoneGap-projektin tavoitteena on ollut mahdollistaa älypuhelinten ominaisuuksien käyttö web-applikaatioissa. Adoben mukaan Cordova projektin tavoite on tehdä itsestään tarpeeton, eli toimia välikappaleena kunnes mobiiliselaimet alkavat tukemaan sen tarjoamia ominaisuuksia. Cordova käyttääkin natiiviselaimen omaa toteutusta sellaisen ollessa tarjolla.

Version 3.0.0 myötä Cordovan ja PhoneGapin välille tuli ensimmäiset eroavaisuudet, kun niiden käytössä siirryttiin enemmän CLI-puolelle (komentorivipohjainen käyttöliittymä). Jotkin komennoista eroavat toisistaan, eikä Cordova luonnollisesti tarjoa PhoneGap Build -komentoja. Tämä päivitys muutti myös radikaalisti kansiorakennetta, jonka ”cordova create”-komento luo automaattisesti. Jokaiselle alustalle erikseen luodun, natiiviohjelmaa muistuttavan kansiorakenteen tilalle tuli yksi yhteinen /www/-kansio, jonka sisältö jaetaan jokaisen alustan kesken. Kappaleen lopussa oleva Kuva 1 näyttää uuden kansiorakenteen. Alustakohtaiset muutokset tehdään /merges/-kansioon, johon lisätään esimerkiksi /merges/android/index.html ja /merges/ios/index.html -tiedostot vastaamaan alustakohtaisia tarpeita. Ennen CLI:n käyttöönottoa projekti tuli natiivisovelluksen tavoin avata ja kääntää IDE:n avulla, mutta nykyään IDE:ä ei tarvita muuhun kuin alustakohtaisten virheilmoitusten lukemiseen. Hooks-kansioon on mahdollista lisätä omia skriptejä jotka ajetaan ohjelman kääntämisvaiheessa. Nämä skriptit voi kirjoittaa millä ohjelmointikielellä tahansa, mutta etenkin monialustasovellusta kehittäessä kannattaa käyttää Node.js:llä ajettavaa JavaScriptiä välttyäkseen alustariippuvuuksilta [Cordova Hooks, 2014].



Kuva 1. Cordova projektin rakenne

2.2 PhoneGap

PhoneGap (http://phonegap.com/) on osittain maksullinen Adoben järjestelmä, jonka suurin ero Cordovaan on mahdollisuus lisätä PhoneGap Build -pilvipalvelun avulla eri mobiilialustat projektiin ilman, että asentaa alustakohtaisia kehitystyökaluja paikallisesti. Myös sovellusten kääntäminen hoituu Build-palvelun avulla pilvessä. Tämä mahdollistaa esimerkiksi iOS-kehityksen Windows koneella. PhoneGap päivittyy hieman Cordovaa hitaammin, vaikka sen kuoren alla Cordova sykkiikin. Cordova projektia käytetään uusien ominaisuuksien testaamiseen, ennen kuin ne liitetään osaksi PhoneGap projektia. PhoneGapia on mahdollista käyttää myös täysin ilmaiseksi ilman Build palvelua, aivan Cordovan tavoin, komentoriviltä paikallisten työkalujen kanssa. [Williams, 2014; Lynch, 2014]

Opinnäytetyön aikana julkaistiin Android-, iOS- ja Windows Phone -alustojen sovelluskauppoihin ilmainen PhoneGap developer app. Se mahdollistaa työpöytäselaimissa yleistyneen ”live reload”-tyylisen kehityksen suoraan samassa WLAN-verkossa olevaan laitteeseen langattomasti ja ilman maksullisia kehittäjätilejä. Pikaisen kokeilun perusteella kyseessä on lupaava sovellus, joka madaltaa kynnystä kokeilla mobiilisovelluskehitystä ja PhoneGapia työkaluna. PhoneGap developer app löytyy osoitteesta http://app.phonegap.com/.

3 JAVASCRIPT

JavaScript on funktionaalinen ohjelmointikieli, jonka syntaksi polveutuu C:stä. JavaScript on callback-kieli, eli koodi kirjoitetaan ennakkoon odottamaan, että jokin kutsuu sitä. Selaimessa tämä koodia kutsuva jokin on nimeltään event loop, joka on yksisäikeinen (single-thread) FIFO-lista (first-in, first-out). Käytännössä tämä tarkoittaa, että jokainen event käsitellään yksi kerrallaan saapumisjärjestyksessä, eikä tässä jonossa ohitella. Mikäli ohjelmakoodi alkaa suorittaa pitkäkestoista operaatiota (kuten tiedon haku palvelimelta), loppuu muun koodin ajaminen kunnes operaatio on valmis. Käyttäjä huomaa tällaisen käytöksen yleensä siitä, että verkkosivu lopettaa käskyihin vastaamisen ja uudemmat selaimet ilmoittavat nopeasti, että sivuston skripti on jumissa. Yksisäikeisyyden asettamia rajoitteita pyritään kiertämään käyttämällä asynkronisia operaatioita. Event loopin jonomaisen luonteen vuoksi koodin tehokkuuden testaaminen on erityisen tärkeää, sillä esimerkiksi tuhannesta millisekunnin kestävästä operaatiosta tulee yksi sekunti lisää vasteaikaa sivustoon. [Resig, 2008]

JavaScriptistä on mahdotonta puhua mainitsematta funktioita, niin tärkeä niiden rooli kielessä on. Perinteisistä ohjelmointikielistä poiketen JavaScript kohtelee funktioita aivan kuten kaikkia muitakin olioita eli objekteja. Erona tavalliseen objektiin on, että function-objektia pystytään kutsumaan. Funktioita voidaan myös syöttää parametreina toisille funktioille.

JavaScriptissä scopet määrittyvät funktioiden mukaan, eivät {}-blokkien perusteella kuten suurimmassa osassa C-syntaksista polveutuvissa kielissä. Selainympäristössä funktio pitää sitoa window-objektiin, niin sanottuun global scopeen, jotta sitä ylipäätään voidaan kutsua. Toinen tapa on kirjoittaa koodi itseään kutsuvan anonyymin funktion sisään (sanasto, IIFE) esimerkin 1 mukaisesti. Tällaiset funktiot sidotaan yleensä johonkin sivun latauksen valmistumisen yhteydessä laukeavaan eventtiin.

Esimerkki 1:

```
(function(){/* code */ }()); // itseään kutsuva funktio
```

Ehkäpä suurin osoitus funktioiden tärkeydestä on, että kaikki JavaScript koodi sijaitsee niiden sisällä. Ainoa poikkeus sääntöön on HTML-merkinnän seassa olevien script-tagien sisään kirjoitettu koodi. Script-tagien sisällä oleva koodi ajetaan samanaikaisesti HTML-merkinnän evaluoinnin kanssa. Suurin osa JavaScript-kirjastoista käyttää ”document ready”-eventtiä koodin ajamisen aloituspisteenä.

Nykyinen JavaScript koostuu erilaisista kirjastoista ja frameworkeista, jotka pyrkivät paikkaamaan natiivin JavaScript kielen puutteita ja selainten välisiä toteutuseroja. Esimerkiksi moduuleita ei niin sanottu vanilla JavaScript kirjoitushetkellä tunne, tuleva ECMAScript 6 -standardi tulee paikkaamaan tämän ja monia muita puutteita. Nykyinen JavaScript pohjautuu ECMAScript 5 -standardiin, ja päivitystahti on ollut toistaiseksi hidasta. Van-

hentuneen standardin takia monia muissa ohjelmointikielissä olevia ominaisuuksia (kuten luokat) joudutaan nykyhetkellä vielä emuloimaan erilaisten koodirakenteiden avulla. Kenties myös kirjastoista päästään hiljalleen eroon selainvalmistajien mahdollistaessa standardien mukaisen koodin ajamisen omien toteutustensa sijaan. [EcmaScript 6 specifications, 2014]

JavaScript osaamiseen lukeutuu selainriippumattoman (cross-browser) koodin kirjoittamisen lisäksi sen testaus, tehokkuuden mittaus (performance analysis) ja debuggaus taidot.

Koodin testaamista pidetään yleisesti ”best practicena”, sillä oletusarvoisesti JavaScript koodia ajetaan useilla eri selaimilla ja sen tulee toimia jokaisessa täysin samoin. Koodin tehokkuuden mittaus on myös tärkeää, sillä JavaScriptin yksisäikeisyys (single-threaded execution model) saattaa aiheuttaa varsinkin vanhemmilla JavaScript moottoreilla melkoisia ongelmia vasteaikojen kasvaessa. [Lähde: Resig & Bibeault, Secrets of the JavaScript Ninja, 2013, s. 39-40]

Koodin tehokkuutta mitataan yleensä ajamalla sitä useita (satoja)tuhansia kertoja luotettavan mittaustuloksen takaamiseksi. Mittayksikkönä käytetään aikaa, yleensä millisekunteja. Koodin tehokkuuden analysointia varten on olemassa useita Web-palveluita joilla voi helposti vertailla esimerkiksi vanilla JavaScriptin ja jonkin JS-kirjaston tarjoaman toteutuksen eroja. Yksi tällainen sivusto löytyy osoitteesta http://jsperf.com/.

3.1 AngularJS

AngularJS (https://angularjs.org/) on Googlen ylläpitämä JavaScript sovelluskehys joka valikoitui projektiin lähinnä oman kiinnostuksen takia, sillä näin yksinkertaisen sovelluksen toteutus olisi onnistunut helpommin esimerkiksi perinteisen jQuery-kirjaston avulla. Halusin kuitenkin tutustua paremmin MVC-malliin, sillä entuudestaan se ei itselleni ollut tuttu. Angular lupasi myös opettaa Unit- ja E2E-testauksen salat kuin itsestään. Verkkosivuilta löytyvän tutoriaalin jälkeen hetken aikaa kuluikin hurmiossa, kun kaikki tuntui loksahtaneen kohdalleen. Käytäntö kuitenkin osoittautui luultua hankalammaksi. Yhtenä syypäänä oli ”Yeoman workflown” ottaminen mukaan projektiin, se kun toi mukanaan lisää entuudestaan tuntemattomia työkaluja kuten Gruntin ja Bowerin. Näitä esitellään tarkemmin otsikosta 4.2 lähtien.

AngularJS tarjoaa modulointisysteemin, joka JavaScriptistä itsestään edelleen puuttuu. Pääasiassa dependency injectionin, josta käytetään jatkossa lyhennettä DI, avulla varmistetaan koko ohjelman latautuminen ennen sen ajamista. DI helpottaa myös ohjelman rakentamista, sillä applikaatio-moduulin alustamisen jälkeen ei ole juurikaan väliä missä järjestyksessä loput komponentit alustaa, toisin kuin JavaScriptissä yleensä. DI:n hienoin piirre on sen yksinkertaisuus, ohjelmamoduuleille annetaan (injektoidaan) riippuvuudet sen sijaan, että ne määrittäisivät ne itse. Tämä mahdollistaa helpon riippuvuuksien hallinnan, jonka ansiosta esimerkiksi testejä varten voidaan käyttää erillisiä testaukseen tarkoitettuja mock-versioita riippuvuuksista.

Angularin riippuvuuksien hallinta ei kuitenkaan hallinnoi tiedostojen lataamisjärjestystä, joka suuremmissa projekteissa tulee ottaa huomioon. Suuremman projektin parissa työskennellessä kannattaakin tutustua esimerkiksi RequireJS tai Browserify työkaluihin. [Sankar, 2014]

Aiemmin hankitusta JavaScript-osaamisesta ei Angularin kanssa päässyt juuri nauttimaan, sillä se tuntui enemmän omalta ohjelmointikieleltään kuin perinteiseltä JavaScriptiltä. Myös Angularin oma sanasto aiheutti aluksi päänvaivaa. Perinteistä poiketen AngularJS-koodi on sidottu window-objektin sijaan $scope-direktiiveihin. Tämä eristää koodin tehokkaasti myös selaimen kehitystyökaluilta ja Angulariin on muiden raskaampien sovelluskehysten tapaan tarjolla omat kehitystyökalunsa. Angular tiimin julkaisema Batarang-niminen Chrome-selaimen lisäosa tuntui toimivan ng-inspectoria paremmin, joten käytin sitä. Työskentely uuden frameworkin kanssa helpottui huomattavasti, kun tajusi asentaa sille tarkoitetut kehitystyökalut.

AngularJS ottaa vahvasti kantaa koodin rakentamismallin suhteen ja käytännössä pakottaa kehittäjän kirjoittamaan koodin oikein. Angularin tapoihin sopeutuminen kestää hetken, varsinkin jos on tottunut perinteisen JavaScriptin parissa työskentelyyn. AngularJS on juuri kantaaottavuutensa vuoksi melko aloittelijaystävällinen sovelluskehys, sillä se ei anna yhtä paljoa mahdollisuuksia virheellisten rakenteiden luomiseen kuin perinteinen JavaScript.

Suuren suosion ansiosta Angulariin on saatavilla valmiita moduuleita melkein jokaiseen käyttötarkoitukseen, eli pyörää ei tarvitse lähteä keksimään itse uudelleen, vaan sen voi ladata projektiinsa vaikkapa Bowerin avulla GitHubista. Esimerkiksi XHR-kutsun aikana näytettävän latausikonin käyttö onnistui lataamalla ”angular-loading-bar”-moduuli ja yhdistämällä se omaan CSS-tiedostoon jossa määritettiin näytettävä animaatio.

Opinnäytetyön kirjoittamisen aikana suureen suosioon nousi Ionic Framework, joka on rakennettu AngularJS:llä. Ionic pyrkii helpottamaan Android- ja iOS-sovellusten kehittämistä tarjoamalla kehittäjille valmiit natiivikäyttöliittymää muistuttavat CSS-komponentit sekä AngularJS-direktiivit Cordova liitännäisten käyttöä varten. Opinnäytetyössä käytetäänkin Ionic tiimin ”ngCordova”-nimellä julkaisemaa direktiivipakettia, joka oli kirjoitushetkellä hyvin aikaisessa alpha-vaiheessa. [Ionic Framework, 2014]

3.2 Koodin testaus

Koodin testaaminen tuntui aluksi pelottavalta salatieteeltä ja se olikin opinnäytetyön aikaa vievin osuus. Sen verran testauksesta etukäteen tiesin, että vain omaa koodia tulee testata. Julkaistujen kirjastojen ja frameworkien koodin tulisi olla valmiiksi testattua, eikä niiden testaamisesta näinollen tarvitse itse huolehtia.

Koodia testaamalla pyritään todistamaan sen toimintavarmuus. Ennen kaikkea testien tarkoituksena on ilmoittaa kehittäjälle virhetilanteista niiden syntyessä ja vähentää virheiden paikantamiseen kuluvaa aikaa. Testit myös

kertovat toisille kehittäjille miten koodi toimii ja niitä käytetäänkin usein pikaoppaan tavoin uuden projektin tai kirjaston pariin siirryttäessä. Testaamiseen tutustumisen tärkeyttä korostaa myös se, etteivät useimmat open source -projektit nykyisin edes vastaanota testaamatonta koodia.

Lähtökohtaisesti JavaScript testien kirjoitus vaatii hieman erilaisen ajattelutavan kuin normaalin koodin kirjoittaminen, esimerkiksi suuri tabu, global scopen käyttö testattavien osien tallentamiseen on testiolosuhteissa täysin hyväksyttävää. Myös ilman selainta työskentely tuntuu aluksi oudolta. AngularJS-tutoriaali teki tällä osa-alueella loistavaa työtä tarjotessaan helposti omaksuttavan lähtökohdan testauksen maailmaan. Suurin osa verkkomateriaalista sivuuttaa testaamisen yleensä joko vaikeana, liian tapauskohtaisena tai turhaan aikaa vievänä.

Opinnäytetyössä perehdyin Jasmine-testausframeworkin saloihin AngularJS tiimin kehittämien Karma ja Protractor työkalujen kanssa. Jasmine on BDD (Behaviour Driven Development) painotteinen testausframework jota voi käyttää ilman selainta, sillä se ei tarvitse DOM puuta toimiakseen. [Jasmine framework, 2014]

Karma-työkalu on tarkoitettu unit- eli yksikkötestien ajoon. Se on suunniteltu toimimaan kaikkien testausframeworkien kanssa. Mikäli Jasmine, Mocha tai QUnit eivät miellytä, voi suosikki frameworkilleen kirjoittaa itse adapterin. Karmalla voi ajaa testejä myös mobiililaitteissa, joskaan tätä ominaisuutta ei opinnäytetyön puitteissa kokeiltu. [Karma, 2014]

Protractor-työkalulla taas ajetaan end-to-end eli E2E-testejä selaimessa. Se on AngularJS:ää varten kehitetty työkalu ja vaatii toimiakseen Google Chrome -selaimen sekä Selenium WebDriverin asentamisen. WebDriverin avulla pystytään simuloimaan käyttäjän toimia verkkoselaimessa, esimerkiksi kirjoittamaan syötteitä eri kenttiin ja klikkailemaan sivun elementtejä. Protractorin käyttö on miellyttävää, sillä sen ajaessa testejä näkee oman verkkosivunsa läpiajon pikakelauksena! [Protractor, 2014]

3.2.1 Unit testaus

Unit eli yksikkötestit ovat lyhyitä ja muusta koodista mahdollisimman eristettyjä testejä, yksinkertaisimmillaan voidaan esimerkiksi verrata yksittäisen funktion palauttamaa arvoa ohjelmoijan ennalta määrittämään oletusarvoon. Unit testien tulisi olla riittävän riippumattomia muusta ohjelmakoodista, jotta niitä pystytään ajamaan esimerkiksi ilman selainta. Päämääränä on yleensä automatisoida testien ajaminen esimerkiksi jokaisen kerran ohjelmakoodin päivittyessä.

Hyvä unit testi on yksinkertainen, itsenäinen (eristetty muusta testikoodista) sekä toistettavissa oleva. Testitulosten tulee pysyä muuttumattomina testejä useaan kertaan ajettaessa, muutoin testitulosta ei voida pitää luotettavana. Testin tulee testata vain yhtä asiaa. Testiä kirjoittaessa tulee poistaa mahdollisimman paljon ylimääräistä (HTML, CSS ja JavaScript) jotta testitulokseen vaikuttaa vain testattava koodi. Testit eivät myöskään saa olla riippuvaisia niitä edeltävien testien tuloksista. Mitä pienempiin osiin testit saa

pilkottua, sitä helpompaa koodin debuggaus eli bugien paikannus virhetilanteiden syntyessä on. On myös tärkeää varmistaa testin toimivuus kirjoittamalla se ensin niin, että se epäonnistuu. [Lähde: Resig & Bibeault, Secrets of the JavaScript Ninja, 2013, s. 48]

TDD-periaatteen vastaisesti kirjoitin testit vasta viimeisenä, kun olin saanut suurimman osan ohjelman toiminnoista kuntoon. Angularin vakaiden julkaisuversioiden päivityksen ei pitäisi rikkoa mitään, mutta testien kanssa voi asiasta olla varma.

Varsinkin Angular direktiivien testaaminen tuntui erittäin vieraalta, kun selain ei renderöi HTML-rakennetta valmiiksi. Perusperiaate niiden testaamiseen tosin on melko yksinkertainen, $compile-funktio kutsutaan parsimaan tietty DOM-rakenne ja varmistetaan, että se on oletuksen mukainen. Angular tarjoaa erikseen testaukseen tarkoitettuja mock-luokkia. Näitä kutsutaan testikoodissa ympäröimällä alaviivoin haluttu luokka esimerkin 2 mukaisesti. Http-backendin mockaus oli vaikeaa, kunnes ymmärsi, ettei sen tarvitse palauttaa oikeaa dataa vaan ideana on lähinnä varmistaa, että API-kutsu lähtee oikeaan osoitteeseen.

Esimerkki 2:n koodissa testataan hakutulossivun toiminta. Aivan ensimmäisenä määritellään testattava Controller-komponentti Jasminen describe-lausekkeella, joka määrittelee millä nimellä testi tulostaa virheilmoitukset komentokehotteeseen. Ennen varsinaisia ”it”-operaattorilla alkavia testejä alustetaan testausympäristö. Yksikkötestissä ei simuloida koodin ajamisen sovelluksessa laukaisevaa napin painallusta. Kolmannen rivin beforeEach-lausekkeella määritellään Angularin DI:in avulla Controllerin vaatimat riippuvuudet, tässä tapauksessa käytetään mock-versiota $httpBackend-luokasta. Viidennellä rivillä $httpBackend.expectGET saa oikeasta API:sta tallennetun JSON-vastauksen paikalliselta koneelta pelkän ok-koodin sijaan. Tämä on testin kannalta hieman epäolennaista. Testi olisi voinut olla yksinkertaisempi ja varmistaa vain koodin kutsuvan oikeaa API-osoitetta. Toisaalta JSON-vastauksen käyttö mahdollisti viimeisen testin, joka varmistaa koodin näyttävän automaattisesti edellisen hakutuloksen käyttäjälle tämän palatessa tulossivulle, kirjoittamisen ja auttoi hahmottamaan Jasminen käyttämää syntaksia.

Esimerkki 2: ProductListCtrl nimisen controllerin testauskoodi

```
describe('Controller: ProductListCtrl', function() {
var scope, ctrl, $httpBackend;

beforeEach(inject(function(_$httpBackend_, $rootScope, $con-
troller) {

$httpBackend = _$httpBackend_;
$httpBackend.expectGET('http://url-here/r/prod-
uct/list').respond(responseJSON);

scope = $rootScope.$new();
ctrl = $controller('ProductListCtrl', {$scope: scope});
}));

it ('$scope.queryNumber should be a function', function() {
expect(typeof (scope.queryNumber)).toBe('function');
```

```
});

it ('should not automatically query', function() {
expect(scope.query).toEqual(undefined);
expect(scope.products).toEqual(undefined);
});

it('should query "from cache" as default action', function()
{
scope.queryNumber('01800');
expect(scope.products).toEqualData([]);
$httpBackend.flush();
expect(scope.products.length).toEqualData(10);
});
});
```

Karma-työkalun testit kirjoitetaan oletuksena karma.conf.js tiedostoon, joka sijaitsee eri kansiossa kuin itse sovelluskoodi.

3.2.2 E2E-testaus

E2E eli end-to-end testauksen tarkoitus on testata ohjelman toimintaa kokonaisuutena ja täydentää yksikkötestejä. Suurimpana erona yksikkötesteihin nähden on, ettei backendiä ole tarkoitus simuloida, vaan testit ajetaan oikeaa backendiä vastaan. Erillistä testibackendiä vasten ajo on myös mahdollista, mikäli sellainen vain on tarjolla.

Testit kirjoitetaan skenaarioina, joissa testataan yleensä yksi kerrallaan sivun tai näkymän (view) kaikki toiminnot ja samalla varmistetaan ohjelman osien yhteensopivuus. E2E-testeillä matkitaan käyttäjän toimia, kuten tekstin syöttöä ja painikkeiden klikkaamista. Ne ajetaan selaimessa. Opinnäytetyössä toteutetussa sovelluksessa testattiin, että Angular tarjoilee sivut oikeassa järjestyksessä, input valuet pysyvät muuttumattomina, napit kutsuvat mitä pitää, cachen toiminta ja hakutulosten suodatus. Skenaariotestit ovat yksikkötestejä huomattavasti laajempia, joten esimerkki 3 sisältää vain hakunapin painamisen sekä vastaanotettujen hakutulosten suodatuksen sovelluksen aloitussivulta. Suodatusta testataan lisäämällä haetun numeron perään lisää nollia ja vertaamalla ”expect”-lausekkeella osumien määrää API:lta saatuun JSON-tiedostoon.

Esimerkki 3: Jasminen E2E-testisyntaksin esittely skenaariotestillä.

```
describe('LviGap App', function() {

it('should redirect to product list view', function () {
  browser.get('http://localhost:9000');
  browser.getLocationAbsUrl().then(function (url) {
    expect(url.split('#')[1]).toBe('/products');
  });
});

/* Main viewin alla ajettavat testit */
describe('Product list view', function () {

beforeEach(function () {
  browser.get('localhost:9000/#/products');
```

```
});

it('should query when search is pressed', function () {
  var productList = element.all(by.repeater('product in prod-
ucts'));
 var query = element(by.model('query'));
  query.sendKeys('01800');
 element(by.css('#queryButton')).click();
  expect(productList.count()).toBe(97);
});

it('should filter query results as user types into search-
box', function() {
  var productList = element.all(by.repeater('product in prod-
ucts'));
 var query = element(by.model('query'));
  expect(productList.count()).toBe(97);

 query.sendKeys('0');
  expect(productList.count()).toBe(10);

 query.sendKeys('0');
  expect(productList.count()).toBe(1);
});
```

Cordova-koodin lisäämisen jälkeen en enää ajanut testejä, sillä Cordova luokat olisi joko pitänyt mockata tai ajaa testit suoraan laitteissa, mutta tähän aika ei riittänyt. Myöhemmin ngCordova tiimi julkaisi valmiit luokat Cordova-ominaisuuksien mockaukseen, mutta testien päivittäminen ajan tasalle olisi silti vienyt paljon aikaa. Protractorin käyttämästä Selenium WebDriverista on saatavilla versiot myös iOS- ja Android-laitteille.

4 NODE.JS

Node.js on Googlen V8 JavaScript moottorin päälle rakennettu alusta, joka mahdollistaa JavaScriptillä kirjoitettujen sovellusten ajamisen OS X, Linux ja Windows alustoilla. Alun perin se kehitettiin ”server push”-ominaisuutta varten sekä mahdollistamaan JavaScriptin käytön serveripuolella.

Itse Node.js on kirjoitettu C:llä. Se on siis C-ohjelma joka ajaa JavaScriptiä. Käytännössä tämä mahdollistaa monipuolisten ohjelmien kirjoittamisen JavaScriptillä. JavaScript itsessään ei ole tarkoitettu käyttöjärjestelmätason operaatioiden ohjaamiseen, mutta sillä pystytään lähettämään niitä ohjaavalle C-koodille käskyjä oikein hyvin. Suurin hyöty kehittäjän kannalta on, ettei tarvitse opetella C-kieltä. [Lähde: What is Node.js?, Brett McLaughlin, 2011]

Perinteisesti web-serverit luovat jokaiselle yhteydelle oman säikeensä. Tavallisesta web-serveristä poiketen Node.js käyttää yhtä säiettä yhteyksien tarkkailuun. Tämä mahdollistaa kymmenientuhansien rinnakkaisten yhteyksien ylläpidon edullisemmin, sillä jokainen säie kuluttaa keskusmuistia. [Wikipedia, 2014]

Node.js mahdollistaa myös perinteisten työpöytäsovellusten kirjoittamisen JavaScriptillä. Tällöin mukaan tosin pitää paketoida Chromium-selainmoottori, joka kasvattaa yksinkertaisten sovellusten kokoa.

4.1 NPM

NPM-paketinhallintajärjestelmää käytetään yleensä Node.js-moduulien hallintaan. Riippuvuudet kirjataan package.json-tiedostoon, josta niitä on mahdollista hallita käsin tai komentorivin kautta. Jotkin IDE:t ja tekstieditorit tarjoavat myös graafisen käyttöliittymän NPM-pakettien hallintaan. NPM tarjoaa kaksi vaihtoehtoa paketin asennukseen, joko käyttöjärjestelmälaajuisen global tai kansiokohtaisen local asennuksen. Osa NPM-paketeista vaatii globaalin asennuksen (esimerkiksi grunt-cli) lisäksi vielä paikallisen paketin asentamisen projektikohtaisesti.

Serveripuolella NPM on oikein pätevä vaihtoehto, sillä se tukee sisäkkäisiä riippuvuuksia. Käytännössä tämä tarkoittaa sitä, että mikäli käytössä on esimerkiksi jQuery-kirjaston eri versioista riippuvaisia komponentteja, voidaan niille jokaiselle tarjota oma versionsa käytettäväksi ja välttää täten mahdolliset konfliktit.

Näin ei haluta toimia front-end puolella, jossa sivustojen latausaikoja pyritään pienentämään kaikin mahdollisin keinoin käyttäjäkokemuksen parantamiseksi. Saman kirjaston useampaan kertaan lataaminen verkon yli vie paitsi kaistaa, myös aikaa.

NPM-paketinhallinnan käyttö front-end puolella on kuitenkin täysin mahdollista. Dokumentaatiosta löytyvän ”npm dedupe”-komennon käyttö on yksi vaihtoehto samojen tiedostojen lataamisen vähentämiseksi. Komento tutkii paikallisen riippuvuuspuun sisällön ja pyrkii yksinkertaistamaan sen

rakennetta siirtämällä yhteneviä riippuvuuksia ylöspäin puussa, mahdollistaen niiden jakamisen usean komponentin kesken. [npm-dedupe, 2014]

Toinen vaihtoehto on käyttää Browserify-moduulia. Se mahdollistaa Nodesta tutun require-metodin käytön selaimessa. [Browserify, 2014]

4.2 Node.js front-end kehityksessä

Front-end kehityksessä Node on nykyään vahvasti mukana. Melkein kaiken tarvittavan pystyy lataamaan NPM-paketinhallintajärjestelmän kautta, JavaScript ja CSS kirjastot sekä erinäiset kehitystyökalut (PhoneGap, Cordova, Grunt jne.) mukaan lukien. NPM:n avulla hoidetaan myös työkalujen päivittäminen ja ajoittainen versioiden palauttelu.

4.2.1 Grunt

Kuva 2. Grunt logo (http://gruntjs.com/img/grunt-logo.svg)

Grunt (http://gruntjs.com/) on JavaScript task runner. Sen avulla voi automatisoida monia työtehtäviä kuten koodin kutistamisen, JSHintin ajon ja CoffeeScript-tiedostojen käännön JavaScriptiksi. Melkein kaikkiin tehtäviin löytyy olemassa oleva plugin, jonka voi helposti asentaa NPM:n kautta.

Projektikansion juuresta löytyvä GruntFile.js sisältää eri grunt taskien määritykset. Yeomanilla luodussa projektissa GruntFile.js-tiedosto oli valmiiksi melkoisen sekava, sillä siinä oli useassa eri kansiossa sijaitsevia taskeja require-viittauksien takana. Yleensä Gruntia tarvitsi käyttää vain kutsumaan valmiita plugineita suhteellisen pienin muokkauksin. Opinnäytetyössä Gruntia käytettiin koodin kehitysvaiheessa ajamaan JSHintiä, web-serveriä

ja kutistamaan koodi Uglify-moduulin avulla yhteen tiedostoon. Myös testien ajo hoidettiin automatisoidusti. Suosittu vaihtoehto Gruntille on Gulp.

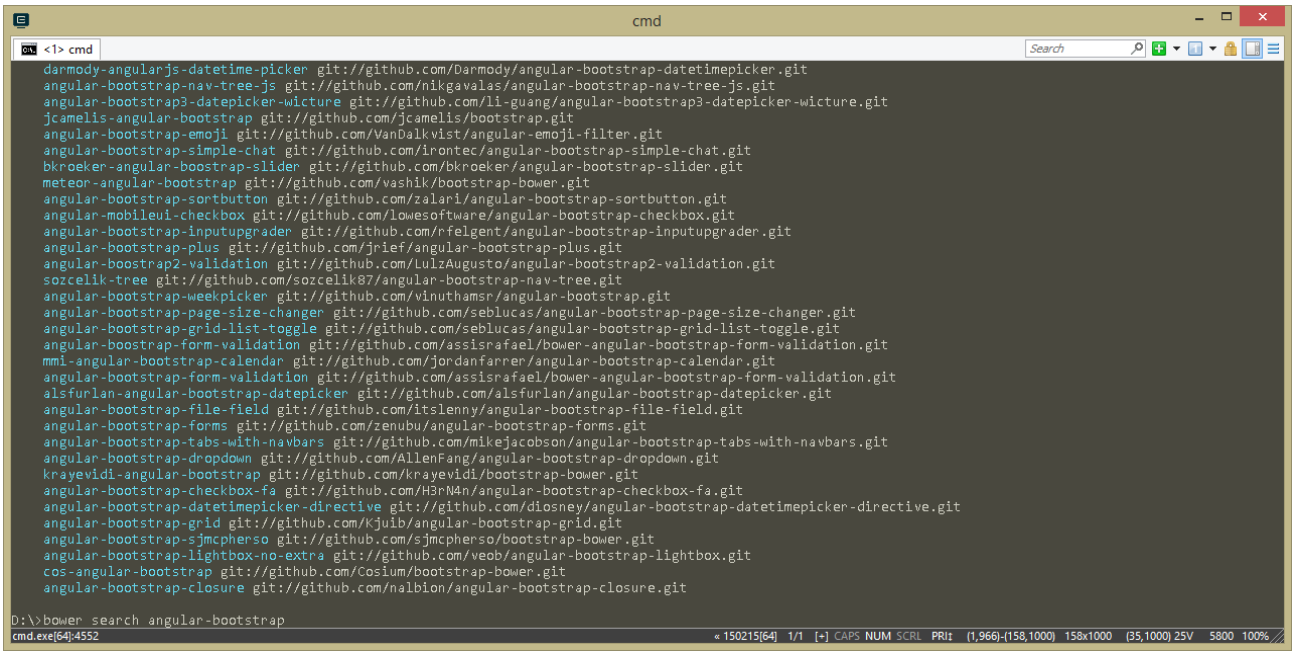
Web-kehityksen lisäksi Gruntilla voidaan automatisoida mitä tahansa muitakin tehtäviä, kuten varmuuskopioiden siirtämisen levyltä toiselle. Kyse ei ole siis pelkästä web-kehitystyökalusta.

4.2.2 Bower

Kuva 3. Bower logo (http://bower.io/img/bower-logo.svg)

Bower on paketinhallintajärjestelmä, joka on optimisoitu front-end-riippuvuuksien hallintaan. NPM paketinhallinnasta poiketen Bower ei tue sisäkkäisiä riippuvuuksia ja vähentää täten sivuston latausaikaa tarjoamalla vain yhden version paketista, josta muut komponentit ovat riippuvaisia. [Bower.io, 2014] Monissa projekteissa käytetään Bowerin rinnalla NPM-paketinhallintaa kehitystyökalujen kuten Yeomanin ja Gruntin asennukseen, ja ”bower install”-komento voidaan liittää osaksi ”npm install”-komentoa.

Bowerin suurin ero muihin markkinoilla oleviin paketinhallintatyökaluihin on, että sen käyttö ei rajoitu ainoastaan JavaScript-kirjastojen hallintaan. Sillä voidaan hallita mitä tahansa paketteja, esimerkiksi HTML- tai CSS-tiedostoja tai kuvia. Bower ei tarjoa mitään lisäominaisuuksia paketinhallinnan ohella, sen avulla ei kutisteta koodia eikä se tue AMD:n kaltaista modulointijärjestelmää. Bower on tarkoitettu vain pakettien hallintaan ja sen se tekee hyvin. Esimerkiksi ”bower search”-komennon avulla voidaan etsiä paketteja komentoriviltä ilman selaimeen koskemista. Ilman hakusanaa komento tulostaa listan kaikista bower paketeista. Bowerin etsi toiminnon käyttö paljastaa, että se on oikeasti oikotie GitHubiin. (Kuva 4) Tämän lisäksi ”bower install”-komennolla voidaan asentaa yksittäisiä tiedostoja URL-osoitteista.



Kuva 4. Bower search -komennon käyttö komentokehoitteessa.

4.2.3 Yeoman

Yeoman (http://yeoman.io) on komentoriviltä ajettava työkalu, joka tarjoaa lukuisia erilaisia generaattoreita, joiden avulla saa polkaistua projektin pikaisesti käyntiin. Opinnäytetyössä käytettiin Angular generaattoria (npm install -g generator-angular), joka generoi hieman yllättäen AngularJS-tutoriaalista poikkeavan sovelluksen. Eroja oli muun muassa AngularJS-tutoriaalin suosittelemien kansioiden nimeämiskäytännöissä (views-kansiota kutsuttiin partials-kansioksi) sekä tutoriaalin käyttämä versio. Yeoman generaattorin asentama stable-versio 1.2.5 erosi jonkin verran tutoriaalin käyttämästä uusimmasta beta-versiosta (1.30-build.3027).

Yeoman mahdollistaa projektin hallinnan komentorivin avulla projektin luonnin jälkeen, joskaan opinnäytetyössä ominaisuutta ei käytetty. Angularin parissa työskennelleelle saattaa perusosien automaattinen luonti säästää hieman aikaa ja vaivaa, mutta alussa se aiheuttaa lähinnä sekaannusta. Yeomanin mukana tulivat tutuksi myös Grunt- ja Bower-työkalut. [Yeoman generator-angular, 2014]

5 OHJELMAN TEKO

SPA (single page app) -kehitys on sovellusarkkitehtuurin puolesta lähempänä natiivia mobiilisovelluskehitystä kuin perinteistä Web-kehitystä. Varsinkin Gruntin mukanaan tuoma kääntämisprosessi muistuttaa enemmän perinteistä sovelluskehitystä kuin web-sivun tekoa. Sovellusta kehittäessä ei ollut vielä mahdollista määrittää Gruntia käyttämään valmiiksi kutistettuja versiota kirjastoista, joten ne oli pakko jokaisella ajokerralla kutistaa uudelleen build-komentoa käyttäessä.

Perinteisissä Web-sovelluksissa Model-luokka on suoraan yhteydessä tietokantaan ja se sijaitsee serveripäässä. Single page appien tapauksessa Model on yleensä yhteydessä rajapintaan (API), joka puolestaan hoitaa tietokantayhteydet. Tässä opinnäytetyössä käsiteltävän ohjelman tapauksessa API oli jo valmis, joten ainoaksi huoleksi jäi kutsujen lähetys sekä datan vastaanotto ja käsittely sovelluksen päässä. Yksi Angularin erikoisuus on, ettei sovelluksen Model-luokalla viitata ainoastaan pysyväisdataan (persistent data), vaan se voi olla käytännössä mikä tahansa objekti jonka muutoksiin halutaan reagoida.

Alkuperäinen Ambientian kehittämä Android-sovellus oli suunnattu vain tablet-laitteille, mutta iOS- ja Windows Phone -markkinoita haluttiin lähestyä kehittämällä sovellus puhelimille. Sovellukseen alun perin toteutetut grafiikat päätettiin käyttää uudelleen niiltä osin kuin se olisi mahdollista, mikä käytännössä poissulki myös sovelluskaupoissa julkaisun vaatimuslistalta.

Toteutustavaksi valitsin hybridisovelluksen, sillä Cordova tuki molempia haluttuja alustoja ja sillä kehitetty sovellus olisi mahdollista myöhemmin julkaista muillekin tuetuille alustoille. Sovelluksen toteutus Cordovalla mahdollistaa yhden koodipohjan jakamisen kaikkien alustojen kesken. Yhteinen koodipohja helpottaa koodin ylläpitoa ja mahdollistaa ideaalitilanteessa ominaisuuksien samanaikaisen lisäämisen ja päivittämisen eri alustoilla. Hybridisovelluksessa laitteiden välisistä eroista ei myöskään tarvitse välittää yhtä paljoa kuin natiivisovellusta kehittäessä. Esimerkiksi jokaista näyttöresoluutiota varten ei tarvitse tehdä erillistä käyttöliittymää, vaan web-sivut saadaan helposti responsiivisiksi. Responsiivisuudella tarkoitetaan, että verkkosivun komponentit skaalautuvat resoluution mukaan ja esimerkiksi puhelimille voidaan tarjota erilailla jäsennelty sivu kuin suuremman näytön omaavalle tablet-laitteelle. Työssä käytetty Bootstrap-kirjasto tarjosi valmiit määritykset (breakpoint) yleisimpien näyttöresoluutioiden mukaan, joskin Windows Phonen Internet Explorer -selain ei ilmoittanut laitteen näyttöresoluutiota samalla tavalla kuin Android- ja iOS-selaimet. Android- ja iOS-puhelimet ilmoittavat selaimelle viewport-metatagin kautta näyttöresoluutioksi 320x480 pikseliä, kun taas Windows Phone ilmoittaa laitteen aidon näyttöresoluution. Testilaitteessa se oli 480x800 pikseliä.

Sovellusta testattiin kehitysvaiheessa pääosin Android- sekä Windows Phone 8.1 -laitteilla. Aluksi käytössä oli myös iPad-laite, mutta luovuin siitä, sillä iOS-emulaattori riitti tarpeisiini. Ilman Apple-kehittäjätiliä piti

sovellusta ajaa iOS-laitteessa PhoneGap Developer App -sovelluksella. Sen kanssa kolmannen osapuolen liitännäisiä (plugin) ei voinut testata, joten ulkoasun hiomisen jälkeen sille ei enää ollut juurikaan käyttöä. Se kuitenkin mahdollisti alkuvaiheessa iOS-kehitystyön Windows-tietokoneella, ennen kuin toimeksiantaja sai järjestettyä meille kehittäjätilit ja Macit.

Loppumetreillä tuli ajankohtaiseksi testata myös itse koodin siirreltävyys tietokoneelta toiselle. Varsinkin Windowsista OS X:lle koodia siirrettäessä huomasi, että HTML-tiedostot ja JavaScript-koodit sisältävien kansioiden symbolinen linkitys ei ollut kovin hyvä vaihtoehto. Kehitysvaiheessa jonkin verran helpotusta siitä oli, sillä sovelluksen kansiorakenne pysyi selkeänä ilman automaatiotyökaluja, jotka olivat eri kansiossa kuin itse Cordova-sovellus. Jatkokehityksen kannalta tuntui kuitenkin tarpeelliselta eriyttää uudet hienot työkalut omaan kehityshaaraansa ja käyttää perinteistä Cordova-projektin rakennetta mahdollisimman helpon käyttöönoton takaamiseksi.

5.1 Kääntäminen muille alustoille

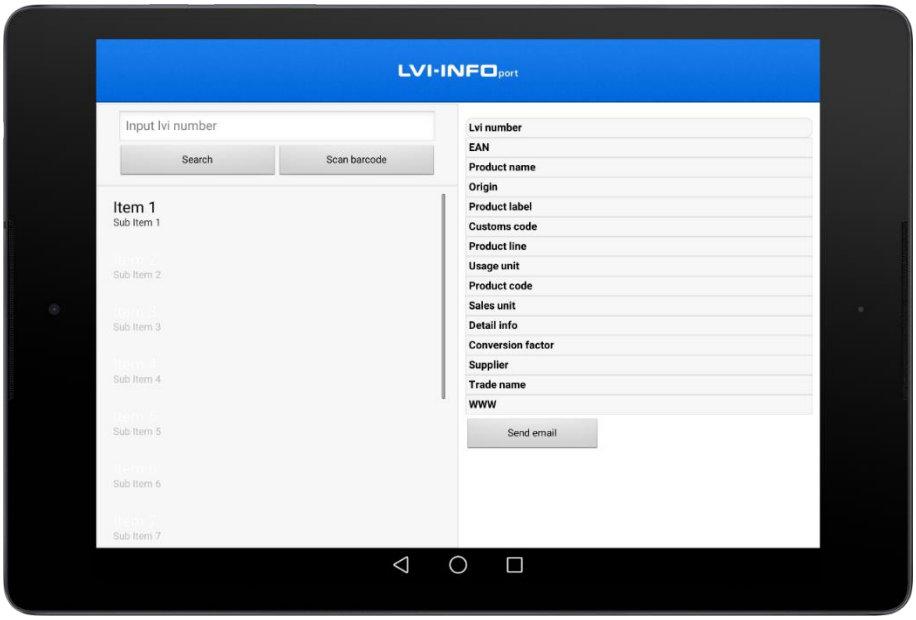
Sovelluksen porttaus eli kääntäminen eri alustalle tarkoitti Cordovan kanssa lähinnä sitä, että katsoin miten alkuperäinen ohjelma toimii ja toteutin samankaltaiset ominaisuudet uudelleen JavaScriptillä. Käyttöliittymä piti totta kai suunnitella uusiksi, sillä tablet-näkymä piti puhelimessa jakaa pienempiin osiin.

Sähköpostin lähetykseen käytettiin ngCordova-kirjaston $cordovaSocialSharing-liitännäistä. Sähköpostitus osoittautui odotettua hankalammaksi, sillä API-vastauksen käyttäminen suoraan lisäsi sähköpostiin toisinaan tyhjiä kenttiä. Ratkaisuna sähköpostin tiedot parsittiin DOM-puusta, jossa tyhjät kentät jätettiin lisäämättä. DOM-puusta poistetut kentät aiheuttivat tosin sähköpostimuotoilun hajoamisen. Onneksi säännölliset lausekkeet (regular expression) auttoivat muotoilun kanssa, ja sain ensikosketuksen niiden käyttöön. Esimerkin 4 koodissa poistetaan ensin ylimääräiset rivinvaihdot ja sen jälkeen korvataan tyhjät merkit rivivaihdolla. Ennen HTML-koodin kutistusta tämä ratkaisu toimi hyvin, mutta kutistamisen jälkeen lausekkeista tuli hyödyttömiä. Sivu jätettiin toistaiseksi kutistamatta, sillä siitä aiheutui enemmän ongelmia kuin hyötyjä.

Esimerkki 4: Sähköpostin lähetys, regex selkeyden vuoksi kahdella rivillä:

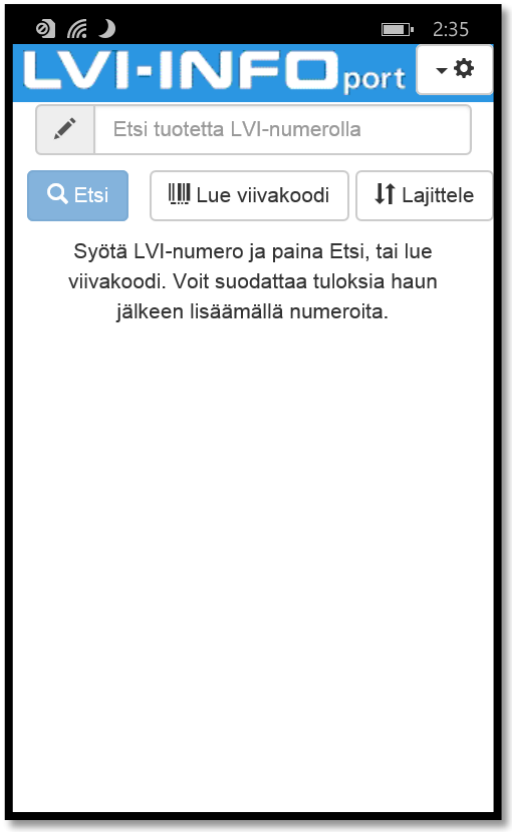
```
var message = angular.element('.results').text()
.replace(/\n\s*\n/g, '\n') //double line breakin poisto
.replace(/\s{2,10}/gm, '\n'); //tyhjien merkkien poisto
```

Alla on ensimmäisessä kuvassa alkuperäinen sovellus (Kuva 5), jonka jälkeen tulevat kuvat ovat näkemykseni puhelinversiosta. Esimerkkikuvat otettiin Windows Phone -laitteella. Käyttöliittymä on jokaisella alustalla toistaiseksi samanlainen, sillä siihen ei tarvinnut kiinnittää toimeksiantajan mukaan liikaa huomiota. Käyttöliittymä skaalautuu suuremmaksi riippuen näyttökoosta, mutta sisältö pysyy samankaltaisena, eli tablet-laitteille ei toteutettu alkuperäisen sovelluksen kaltaista näkymää joka mahdollistaa tuotetiedon avaamisen ja tarkastelun ilman sivunvaihtoa.



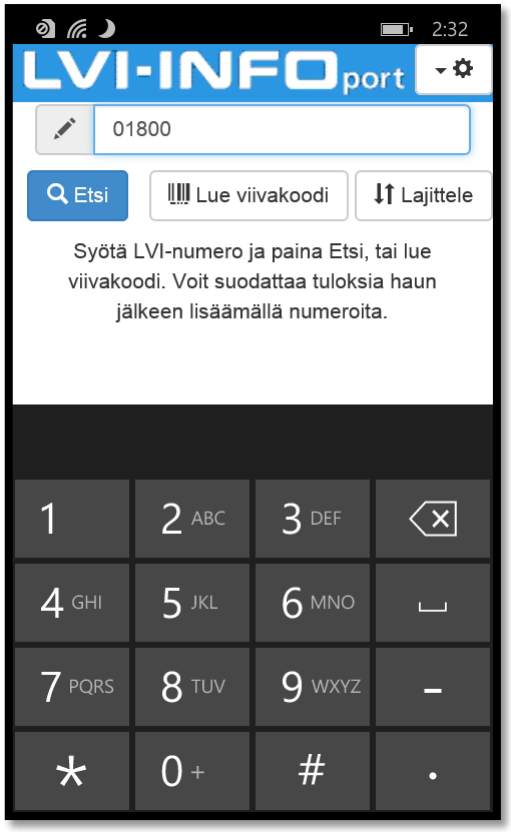
Kuva 5. Alkuperäinen Android-sovellus emulaattorissa.

Sovelluksen käynnistyessä käyttäjälle näytetään alla olevan kuvan mukainen näkymä. Alkuperäisestä sovelluksesta poiketen lisäsin lajittele-painikkeen, joka kääntää hakutulokset päinvastaiseen järjestykseen sekä oikeaan yläkulmaan hammasrattaan kuvalla varustetun pudotusvalikon, josta voi vaihtaa sovelluksen kieltä. Lvi-infon API ei toistaiseksi kuitenkaan sisällä kuin suomenkielistä sisältöä, mutta käännöksille oli jo olemassa tyhjät kentät. (Kuva 6)



Kuva 6. Puhelinsovelluksen alkunäkymä.

Angularin kehittäjäystävällisyys näkyy alla olevassa kuvassa, jossa päädyin käyttämään input arvona tavallisen numeron sijaan puhelinnumeroa, sillä Angular muuntaa numerona syötetyt luvut automaattisesti kokonaisluvuiksi (int), eli käytännössä poistaa kaikki muita numeroita edeltävät nollat. Tämä vaikuttaa sovelluksen käyttäjälle tarjoamaan näppäimistöön, jossa ei ole enter-näppäintä jolla aloittaa haku. Käyttäjälle piti siis tarjota erillinen etsi-painike sovelluksen käyttöliittymässä, sillä helppoa tapaa joko tyyppimuunnoksen estämiseen tai erilaisen näppäimistön esittämiseen ei ollut. (Kuva 7)



Kuva 7. Puhelinsovelluksen hakukenttä, jossa päädyin käyttämään puhelinnumeroa hakukentän input arvona.

Etsi-painikkeen painamisen jälkeen luetaan käyttäjän syöttämä LVI-numero ja sovellus listaa kaikki hakutulokset allekkain. Käyttäjä voi tämän jälkeen lisätä ja poistaa numeroita rajatakseen hakua. Tämä on AngularJS:n sisäänrakennettu ominaisuus, jota ei erikseen tarvinnut toteuttaa. (Kuva 8)

Viivakoodinlukijan skannaama numero laukaisee automaattisen haun, joka palauttaa käyttäjän hakunäkymään yleensä yhden tuotteen kanssa (Kuva 8). Tarkoitus oli toteuttaa viivakoodista haku suoraan Kuva 9 ”tarkemmat tiedot”-näkymään, mutta Windows Phonen viivakoodinlukijaliitännäisen hajoaminen aiheutti ristiriidan kehitysprioriteetteihin ja ominaisuus jäi toteuttamatta.

Kuva 8. Puhelinsovelluksen tulosnäkymä.

Hakutulosta painamalla sovellus siirtää käyttäjän tuotteen tarkempiin tietoihin. Näytettävistä tiedoista poistetaan tyhjät kentät, joita API:lta tulee. Esimerkiksi alkuperämaa puuttuu suuresta osasta tuotteita. Tuotteen nimen käyttö otsikkona johtuu siitä, että kaikista tuotteista ei ole kuvaa ja näkymä oli pelkkänä listana erittäin sekava.

Tarkemmat tiedot sivulla on vasemmassa yläkulmassa joka alustalla ”Palaa”-painike. Windows Phonen fyysisen takaisinpaluunäppäimen käyttö aiheutti toisinaan erilaisia ongelmia selainhistorian kanssa, varsinkin kun kyseessä oli skannattu viivakoodi. Suoraan tähän näkymään viivakoodinlukijasta siirryttäessä kävi välillä niin, että sovellus kutsui liitännäistä uudelleen ilman tarvittavia parametreja ja avasi kameranäkymän, josta ei päässyt pois kuin sammuttamalla puhelimen. Android-laitteella testatessa näin ei käynyt kertaakaan. Windows Phonen selainhistoria siis toimi erilailla kuin webkit-selaimissa, mutta siihen ei ehditty paneutua tarkemmin viivakoodinlukijan lakattua palauttamasta lainkaan arvoja. (Kuva 9)

Kuva 9. Puhelinsovelluksen tarkemmat tiedot -näkymä.

5.2 iOS

Web-sovellusten kehitysalustoina iOS ja Android eivät juuri poikkea toisistaan, sillä molemmissa on Webkit-pohjainen selain. Ainoa vastaan tullut ongelma iOS-toteutuksessa oli, että yläpalkki jossa kello ja ilmoitukset sijaitsevat piirtyi ohjelman päälle. Huomioon otettavaa iOS-sovelluksen kehityksessä on, ettei Applen laitteissa ole erillistä näppäintä takaisinnavigointia varten, vaan se tulee toteuttaa sovelluksen käyttöliittymässä Kuva 9 tavoin. IOS-versiopäivitysten myötä käyttöjärjestelmän navigointipainikkeiden ulkoasu vaihtelee, joten natiivisovellusmaiseen ulkoasuun tähtäävän web-sovelluksen pitäisi sisältää jokaiselle eri iOS versiolle omat CSS-määritykset.

Android- ja iOS-sovellusten kanssa on mahdollista käyttää myös työpöytäselaimen (Chrome ja Safari) kehitystyökaluja virheiden paikantamiseen etäyhteydellä (remote debugging). Vieläkin helpompi vaihtoehto näille alustoille löytyy GapDebug-nimisen sovelluksen muodossa. Ohjelman kotisivut löytyvät osoitteesta https://www.genuitec.com/products/gapdebug/. Suurimman osan iOS-kehityksestä pystyi hoitamaan Windows tietokoneen Chrome-selaimella. Kun Chromessa saatiin ulkoasu toteutettua, testattiin sitä vielä PhoneGap developer appin avulla laitteessa. Cordovan ominaisuuksia tulee aina testata ajamalla sovellus fyysiseen laitteeseen, sillä emulaattorit eivät anna luotettavaa kuvaa sovelluksen toiminnasta.

5.3 Windows Phone 8

Windows Phone -alusta esitti työn suurimmat haasteet. PhoneGap-käännöskilpailu Windows Phonelle pidettiin onneksi hieman ennen opinnäytetyön aloitusta, joten monia asioita oli ehditty jo murehtia. Sen sijaan, että olisi alettu keksiä ratkaisuja alustan ongelmiin, pystyttiin tutkimaan erilaisia valmiita ratkaisuja. Silti osa ongelmista jäi korjaamatta, sillä osaa virheistä ei saatu luotettavasti toistettua. Windows Phonen kanssa työskennellessä virheiden paikannus tuntuu olevan hyvin työlästä. Edes Visual Studio ei antanut JavaScript virheistä ilmoitusta. Lopulta koko sovellus hajosi Cordovan versiopäivityksen yhteydessä, eikä enää edes aiemmin toiminutta versiota saatu ajettua puhelimeen täysin toimivana. Viivakoodinlukija (virallinen plugin) ei suostunut käynnistymään jostain syystä oikein. Sovellus avasi kameranäkymän, muttei tehnyt sen jälkeen mitään.

Ohjelman kehitys jatkui paikallisen koneen selaimen jälkeen pääosin Lumia 820 -puhelimella. Tässä vaiheessa ongelmia alkoi ilmetä, ensimmäisenä alustan web-selaimelle ominainen efekti, sivun ylä- tai alareunaan kelatessa aiheutuva pomppiminen (rubber banding), jota natiivisovelluksissa ei ole. Mitään helppoa keinoa tämän efektin poistamiseen ei löytynyt. Vaihtoehtona olisi ollut poistaa kaikki kosketuseleet käytöstä. Esimerkin 5 mukainen CSS-määritys, joka poistaa kaikki oletusarvoiset kosketuseleet käytöstä on hyvä, mikäli web-sivun sisältöä ei tarvitse vierittää. Tämän ohjelman tapauksessa listattuja tuotteita tuli hausta riippuen niin paljon, ettei tätä ratkaisua ollut järkevää käyttää, sillä tulos sivulle olisi joutunut toteuttamaan vierityksen uudelleen itse JavaScriptillä (Kuva 8).

Esimerkki 5:

```
CSS-määritys:
 -ms-touch-action:none;
```

Myös Windows Phonen tapa käsitellä viewport-ominaisuutta poikkesi muiden puhelinalustojen selaimista. Windows Phone -sovellus ilmoittaa viewport-kokona näytön natiiviresoluution, joka on minimissään 480x800 pikseliä, kun taas Android ja iPhone esittävät viewportin avulla selaimelle olevansa 320x480 pikseliä. Tähän onneksi auttoi esimerkin 6 mukainen yksinkertainen media query. Se kertoo Internet Explorer -selaimelle, että renderöi sisällön 320 pikseliä leveänä puhelimen ollessa pystyasennossa. Huomionarvoista on, että sama CSS-määrite on käytössä myös Internet Explorerin työpöytäversiossa. Mikäli media queryn jättää määrittämättä, kuten tämän sovelluksen tapauksessa tehtiin Windows Phone -puhelinten resoluutioiden monipuolistumisen takia, ei samaa sivua parane tarjoilla työpöytäkäyttäjille.

Esimerkki 6:

```
@media (max-width: 480px){
 @-ms-viewport {
           width: 320px;}
 }}
```

Chromen remote debuggaus -työkaluja tuli ikävä, sillä Windows Phonen kanssa toimi ainoastaan Weinre (Web inspector remote), jota taas ei suositella käytettävän yhdessä Angularin kanssa. Tässä sovelluksessa Weinre hajotti ainakin internet yhteydet, mutta toisaalta sen avulla varmistuttiin, että Cordova ja AngularJS molemmat latautuivat oikein. Visual Studioon ei ohjelma jostain syystä suostunut lähettämään virheilmoituksia kuin muutaman kerran, joten ohjelmaa piti tehdä JavaScriptin osalta käytännössä sokkona. Angular virheiden paikannus hoitui Android-puhelimen kanssa kivuttomammin, joten käytin sitä hyödyksi Cordovan liitännäisten kanssa. [Bloch, 2013]

Windows Phone 8.1 preview -päivitys oli aluksi epäilty syypää siihen, että Angularin sivujen vaihto (routing) hajosi yhtäkkiä kokonaan. Sovellus tarjosi vain linkkiä sovelluskauppaan, kun yritti vaihtaa näkymää. Tutkimusten jälkeen kuitenkin paljastui, että Windows Store -ohjelmat lisäävät URL-osoitteiden alkuun joko ms-appx (Windows 8) tai x-wmapp0 (Windows Phone 8) alkuliitteen. Angularin $compileProvider-direktiivi piti siis muokata hyväksymään molemmat esimerkin 7 mukaisesti.

Esimerkki 7:

```
function ($compileProvider)
{
$compileProvider.aHrefSanitizationWhitelist
(/^\s*(https?|ftp|mailto|file|ghttps?|ms-appx|x-wmapp0):/);
};
```

Ensimmäisen kappaleen lopussa mainittu sovelluksen hajoaminen johtui XHRHelper.cs-nimisen luokan muutoksista. Se sijaitsee */platforms/wp8/cordovalib/*-kansiossa ja on osa Cordovan luomaa natiivisovelluskehystä. Ongelma oli, että sovellus lakkasi yhtäkkiä lataamasta aloitussivua 3.4.0 versiosta päivittämisen jälkeen. Opinnäytetyön puitteissa en ehtinyt tutkia miten ”hookin”, eli kääntämisvaiheessa ajettavan skriptin, olisi saanut korvaamaan Cordovan luoman luokan, jonka tyydyin nyt vain päivittämään tiedostoa käsin toimivaan versioon. Todennäköisesti tämä virhe poistuu tulevan versiopäivityksen myötä.

6 YHTEENVETO

Opinnäytetyön tavoitteena oli kääntää toimeksiantajan, Ambientia Oy:n, kehittämä alkuperäinen Android-sovellus PhoneGapin avulla sekä iOS- että Windows Phone 8 -alustoille yhteensopivaksi. Tavoitteeseen ei päästy täydellisesti kummallakaan alustalla. Kehitykseen olisi tarvinnut varata huomattavasti enemmän aikaa. iOS-puolella sovelluksen ulkoasua pitää vielä muokata tablet-laitteille sopivaksi, muuten sovellukseen saatiin toteutettua halutut ominaisuudet.

Windows Phone 8.1 -alusta aiheutti enemmän ongelmia. Cordova 3.4.0 -version päivittämisen jälkeen sovellusta ei saanut kääntymään toimivaksi erään päivityksen yhteydessä ilmaantuneen bugin takia. Windows Phonella aiemmin toiminut viivakoodinlukija lakkasi myös toimimasta. Viivakoodinlukua ei saatu toimimaan enää edes aiemmin käännetyllä sovelluksella, jolla se toimi ennen Cordovan päivitystä. Syy käytökseen ei selvinnyt, sillä Visual Studion debuggeri pysyi mykkänä.

Kysymys hybridisovelluksen toteuttamisesta natiivisovelluksen sijaan ei ole yksiselitteinen. Mikäli voidaan hyödyntää jo olemassa olevia Web-kehitystaitoja, voi PhoneGap-sovellus olla oikea valinta. Vaikka JavaScript-sovellukset eivät pystykään natiivikoodin tehokkuutta vastaamaan, ei käyttäjän kannalta tällä ole juurikaan merkitystä, mikäli sovellus vain tuntuu reagoivan käyttäjän toimiin nopeasti. Kehittämistä helpottaa yhden koodipohjan (code base) ylläpito, joka mahdollistaa myös jokaiselle alustalle samaan aikaan tapahtuvat muutokset. Tämän lisäksi erityisesti Android-puolella villisti vaihtelevat näyttöresoluutiot saadaan viewport-metatagin avulla toteutettua yksittäisellä käyttöliittymällä. Hieman enemmän aikaa kehitysvaiheessa kuluu eri alustojen testaamiseen.

Monialustakehitystä tukevia sovelluskehyksiä, jotka tarjoavat natiiviohjelmaa muistuttavan käyttöliittymän, alkaa löytyä jo useampia. Suurin osa niistä tosin ei tue Windows Phonea. Android- ja iOS-ohjelmien toimintalogiikka tuntuu olevan menossa enemmän samaan suuntaan yksinkertaistuvan ulkonäön ja poistuvien painikkeiden ansiosta. Jopa käyttöjärjestelmien pyyhkäisyeleet muistuttavat toisiaan. Android- ja iOS-sovellusten toteutus onnistuukin PhoneGapilla helposti. Windows Phone toteutukseen pitää kuitenkin nykytilanteessa varata enemmän aikaa, sillä sen tuki tuntuu olevan osittain puutteellista, eikä sen käyttöliittymää saa toteutettua nopeasti valmiilla sovelluskehyksellä.

Opinnäyteyön alussa en tuntenut kumpaakaan käännöksen kohteena olevaa käyttöjärjestelmää. Aiempaa kokemusta minulla oli Android-ohjelmoinnin ja Cordova-kehityksen perusteista. Halusin lisätä mahdollisimman monia itseäni kiinnostavia työkaluja mukaan projektiin, sillä toimeksiantajalla ei ollut erityisvaatimuksia käytettyjen tekniikoiden suhteen. Opinnäytetyö kartutti kokemusta monialustakehityksen lisäksi MVC-ohjelmointimallista ja AngularJS-ohjelmoinnista, joka poikkeaa huomattavasti perinteisestä JavaScriptistä. Lisäksi sain mahdollisuuden tutustua Node.js-pohjaisiin työkaluihin, joita useat open source projektit nykyään käyttävät.

Cordova tuntemuksen lisäännyttyä on pakko ihmetellä, miksi se asennetaan käyttöjärjestelmätasolla eikä projektikohtaisesti. Versioiden edestakaisin päivittelyltä vältyttäisiin helposti, mikäli jokainen projekti käyttäisi omaa, ehkä jopa alustakohtaista versiotaan. Windows Phone -sovelluksen hajoamiselta olisi saatettu välttyä, mikäli iOS-alustalta löytyneen haavoittuvuuden paikkaaminen ei olisi päivittänyt sitäkin alustaa. Myöskään vakiintunutta käytäntöä Cordova liitännäisten hallintaan ei vielä ole, kokeiluasteella oleva save/restore -toiminto ei itselläni ainakaan toiminut. Tiimityöskentelyä hankaloittaa liitännäisten hallinta, sillä valmiiksi käännetyt sovellukset sotkevat versiohallinnan nopeasti. Gitignore-komennon käyttö platforms- ja plugins-kansioille onkin vakiintunut käytäntö. Parhaita vaihtoehtoja nykyhetkellä on package.json-tiedostoon oman ”cordova plugins add”-komennon lisäys ”npm install”-komennon yhteyteen.

Opinnäytetyön tuloksena syntynyt sovellus ei ole julkaisukelpoinen sovelluskaupoissa, sillä se ei huomioi eri laitealustojen eroja, eikä sitä saatu luotettavasti toimimaan Windows Phone alustalla. Esimerkiksi Bootstrap-kirjasto oli mielestäni huono valinta tähän projektiin ja sen kanssa työskentely oli ajoittain tuskallista. Bootstrapin käyttöönottoa helpompaa olisi ollut toteuttaa käyttöliittymä käsin. Kehityksessä apuna käytettyjä työkaluja olisi kannattanut ottaa käyttöön yksi kerrallaan, sillä niiden yhtäaikainen tutkiminen ja käytön opettelu vei opinnäytetyössä liian paljon aikaa. Automaation ymmärtäminen on huomattavasti helpompaa, mikäli on ensin käsin tehnyt automatisoitavat vaiheet.

Jatkokehityksen helpottamista ajatellen erotin sovelluskoodin ylimääräisistä kehitystyökaluista tavalliseksi Cordova-projektiksi. Sovelluskaupoissa julkaisua varten tulee vielä testata näkymien toimivuus tablet-laitteilla, huomioida erikseen Windows Phonen fyysinen palaa-näppäin sekä saada viivakoodinlukija liitännäinen jälleen toimimaan. Lisäksi Bootstrapin vaihtaminen esimerkiksi kevyempään Topcoat-kirjastoon on harkitsemisen arvoista. Topcoatilla toteutettu ensimmäinen versio käyttöliittymästä tuntui helpolta ja luonnolliselta verrattuna Bootstrapin oppimiskäyrään. Responsiivisuutta Topcoat-kirjasto ei suoraan tarjoa, joten mikäli Bootstrapin gridmalli on toivottu osa käyttöliittymää, pitää sellainen toteuttaa erikseen joko valmiilla kirjastolla tai käsin. Media queryjen avulla grid-mallin voisi jättää kokonaan pois sovelluksesta ja toteuttaa resoluution mukaan skaalautuvan käyttöliittymän. Mikäli Windows Phonen kanssa muu ei auta, voisi kokeilla yhdistää olemassa olevaa koodia natiiviin Windows Phone HTML-sovellukseen. Tällöin tulisi tutustua tarkemmin WinJS-kirjastoon ja miten sen voi yhdistää Angularin kanssa.

LÄHTEET

Resig, John & Bibeault, Bear. 2013. Secrets of the JavaScript Ninja. Manning Publications Co.

McLaughlin, Brett. 2011. What is Node.js?. O'Reilly Media.

Appgyver, 2014: AppGyver homepage. Viitattu 28.10.2014, http://www.appgyver.com/

ASF, 2014: Project Details for Apache Cordova. Viitattu 4.7.2014, http://projects.apache.org/projects/cordova.html

Bloch, O. 2013: Debug your mobile HTML5 page remotely with weinre (WEb INspector REmote), Viitattu 2.12.2014, http://msopentech.com/blog/2013/05/31/now-on-ie-and-firefox-debug-your-mobile-html5-page-remotely-with-weinre-web-inspector-remote/

Bootstrap, 2015: Bootstrap · The world's most popular mobile-first and responsive front-end framework. Viitattu 13.4.2015, http://getbootstrap.com/

Bower.io, 2014: A package manager for the web. Viitattu 9.7.2014, http://bower.io

Browserify, 2014: Browserify lets you require('modules') in the browser by bundling up all of your dependencies. Viitattu 9.7.2014, http://browserify.org/

Chow, C. 2011: Day 1: Nitobi joins Adobe. Viitattu 4.7.2014, http://phonegap.com/2011/10/27/day-1-nitobi-joins-adobe/

Cordova developer mailing list archive, 2014. Viitattu 28.10.2014, http://mail-archives.apache.org/mod_mbox/cordova-dev/

Cordova Hooks, 2014. Viitattu 18.11.2014, https://github.com/apache/cordova-app-hello-world/blob/master/hooks/README.md

Cordova plugin registry, 2014: Discover plugins for your Apache Cordova project. Viitattu 26.5.2014, http://plugins.cordova.io/

Cordova projekti, 2014: Apache Cordova. Viitattu 4.7.2014, http://cordova.apache.org/

EcmaScript 6 specifications, 2014: ECMAScript Language Specification ECMA-262 6th Edition – DRAFT. Viitattu 6.8.2014, http://people.mozilla.org/~jorendorff/es6-draft.html

Ionic Framework, 2014: Ionic: Advanced HTML5 Hybrid Mobile App Framework. Viitattu 28.11.2014, http://ionicframework.com/

Jasmine framework, 2014: Jasmine: Behavior-Driven JavaScript. Viitattu 13.7.2014, http://jasmine.github.io/

Karma, 2014: Karma - Spectacular Test Runner for Javascript. Viitattu 2.7.2014, http://karma-runner.github.io/

Lynch, M. 2014: The Last Word on Cordova and PhoneGap. Viitattu 4.7.2014, http://ionicframework.com/blog/what-is-cordova-phonegap/

npm-dedupe, 2014: dedupe | npm Documentation. Viitattu 7.7.2014 https://www.npmjs.org/doc/cli/npm-dedupe.html

PhoneGap blogi, 2008: Unofficial announcement of PhoneGap. Viitattu 4.7.2014, http://phonegap.com/2008/08/07/unofficial-announcement-of-phonegap/

Protractor, 2014: E2E test framework for Angular apps. Viitattu 2.7.2014, https://github.com/angular/protractor

Resig, J. 2008: John Resig - How JavaScript Timers Work. Viitattu 14.8.2014, http://ejohn.org/blog/how-javascript-timers-work/

Sankar, A., 2014: AngularJS Dependency Injection Demystified. Viitattu 14.1.2015, http://anandmanisankar.com/posts/angularjs-dependency-injection-demystified/

Schinsky, H. 2014, Mixing Cordova/PhoneGap Components with Native iOS – Part 1. Viitattu 22.7.2014, http://devgirl.org/2014/07/22/mixing-cordova-phonegap-components-with-nativ/

Telerik Verified Plugins Marketplace, 2014: Verified Plugins Marketplace | Cordova/PhoneGap Plugins. Viitattu 16.4.2015, http://plugins.telerik.com/

Virtualisointi MSDN, 2014: Installing Windows and the dev tools on your Mac. Viitattu 4.7.2014, http://msdn.microsoft.com/en-us/library/windows/apps/jj945492.aspx

Williams, T. 2014: Cordova vs. PhoneGap: An update. Viitattu 4.7.2014, http://blog.devgeeks.org/post/73789983750/cordova-vs-phonegap-an-update

Yeoman generator-angular, 2014: Yeoman generator for AngularJS - lets you quickly set up a project with sensible defaults and best practices. Viitattu 4.7.2014, https://github.com/yeoman/generator-angular

Yeoman, 2015: The web's scaffolding tool for modern webapps | Yeoman. Viitattu 13.4.2015, http://yeoman.io/