LAMK Lahden ammattikorkeakoulu
Lahti University of Applied Sciences

Uudelleenkäytettävä mobiilisovellus

Mobiilisovelluksen kehittäminen

LAHDEN
AMMATTIKORKEAKOULU
Tekniikan ala
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka
Opinnäytetyö
Kevät 2016
Timo Salola

Lahden ammattikorkeakoulu
Tietotekniikan koulutusohjelma

SALOLA,TIMO:

Uudelleenkäytettävä mobiilisovellus
Mobiilisovelluksen kehittäminen

Ohjelmistotekniikan opinnäytetyö, 47 sivua

Kevät 2016

TIIVISTELMÄ

Opinnäytetyössä tutkittiin mobiilisovelluksen kehittämistä sekä toteutettiin mobiilisovellus. Mobiilisovelluksen kehittämisessä tulee huomioida kohdeyleisö, koska on olemassa useita eri mobiililaitteita sekä -käyttäjärjestelmiä, jotka eroavat toisistaan. Tämä vaikeuttaa kehittäjien työtä, koska ei ole yhtä oikeaa ratkaisua tuottaa sovellus jokaisen käyttäjän laitteelle.

Natiivisovelluksilla pystytään toteuttamaan laitekohtaisia ratkaisuja, joiden avulla pystytään toteuttamaan monipuolisia sovelluksia. Web-sovellukset tarjoavat mahdollisuuden käyttää sovellusta usealla eri laitteella, mutta ne eivät pysty korvaamaan natiivisovelluksen tuomaa tukea laitteelle. Näiden kahden tyypin välille on syntynyt mobiilisovellusmaailmaan hybridisovellus, jonka avulla pystytään tekemään natiivisovelluksia usealle eri alustalle.

Opinnäytetyössä suunniteltiin ja toteutettiin mobiilisovellusalusta, jonka avulla pystytään toteuttamaan asiakkaan toiminnanohjaukseen liittyviä projekteja. Tavoitteena oli luoda mahdollisimman monia mobiililaitteita tukeva alusta, jolloin sovellusta pystytään käyttämään jo asiakkaan hankkimilla laitteilla.

Työ tehtiin kotimaiselle JL-Soft Oy:lle. Yritys on erikoistunut ohjelmistojen suunnitteluun sekä tuotantoon. Yritys kehittää sekä tarjoaa omaansa ModulErp toiminnanohjausjärjestelmää asiakkailleen. Yritys myös tarjoaa ohjelmistojen suunnittelua sekä toteuttamista asiakkailleen.

Asiasanat: mobiilisovelluksen kehittäminen, hybridisovellus, JavaScript, Cordova, Angular, Bootstrap

Lahti University of Applied Sciences
Degree Programme in Information Technology

SALOLA, TIMO:

Reusable mobile software
Mobilesoftware development

Bachelor's Thesis in software engineering, 47 pages

Spring 2016

ABSTRACT

The objective of this thesis was to investigate mobile software development. In mobilesoftware development one must take into account the target audience since there are multiple different mobile devices and platforms available which differ from each other. This makes it hard to develop mobile software because there is not common solution to make an application for every user's device.

With native applications developers can make device specific solutions, that enable the development of versatile applications. Web applications can be used with many devices but they cannot support devices to the same extent as native applications. Between these two there has come a new solution, hybrid application, which tries to give it best of both worlds.

In this thesis, a mobile software platform was planned and developed. In this mobile software platform a company can do customer-based projects that are related to enterprise resource planning. The purpose of the development of the mobile software platform was to create a platform that can support many mobile devices and platforms.

The work was accomplished for JL-Soft Oy. The company is specialized in software development and production. The company develops and sells their own ModulERP enterprise resource planning software for their customers.

Key words: mobile software development, hybrid application, JavaScript, Cordova, Angular, Bootstrap

SISÄLLYS

1 JOHDANTO

JL-Soft Oy on vuonna 1990 perustettu ohjelmistotuotekehitykseen erikoistunut yritys. Yrityksen tuotteita on toimitettu jo yli 25 maahan. JL-Soft Oy:n päätuote on yrityksen kehittämänsä toiminnanohjausjärjestelmä, joka hallitsee toiminnanohjauksen kaikki osa-alueet. Sovellusta on kehitetty jatkuvasti viimeiset 20 vuotta ja sitä on pyritty päivittämään vastaamaan tämän päivän yritysten tarpeita. Sovelluksen käyttöliittymästä on olemassa työpöytä, web sekä mobiiliversiot. (JL-Soft 2015.)

Tämänhetkinen toiminnanohjausjärjestelmä on rakennettu modulaariseksi, jolloin asiakas voi valita tarvitsemansa osa-alueet yrityksensä käyttöönsä. Järjestelmästä on tarjolla valmiita paketteja yrityksille aina täysin muokattaviin versioihin, jotka räätälöidään aina asiakkaan tarpeiden mukaan. (JL-Soft 2015.)

Toiminnanohjauksesta halutaan saada mahdollisimman reaaliaikaista tietoa, jolloin järjestelmän pitää kulkea käyttäjien mukana. Tämä mahdollistaa käyttäjille suoraan työtehtävien lähettämisen tai uusien myyntitilausten lähettämisen järjestelmään ilman, että myyjän tarvitsee palata toimistolle. Mobiililaitteiden yleistymisen takia tämä on mahdollista toteuttaa.

Tämänhetkinen mobiilitoteutus on hankittu kolmannelta osapuolelta, jonka kanssa JL-Soft Oy tekee aktiivisesti yhteistyötä kehittääkseen sovellusta eteenpäin. Mobiilisovelluksella on oma rajanpinta, joka on integroitu toiminnanohjauksen www-sovelluspalveluun. Toiminnanohjauksen www-sovelluspalvelun vastuulla on yrityslogiikka ja mobiilisovelluksen rajapinnan on tarkoitus toimia mobiilisovelluksen sekä toiminnanohjauksen www-sovelluspalveun välillä. Tämä monimutkaistaa sekä hidastaa tämänhetkistä toteutusta, koska on olemassa useampi rajapinta pääsovelluksen sekä käyttäjän välissä.

Työn tavoitteena on tutkia eri tekniikoita, joilla pystytään toteuttamaan mobiilisovelluksia. Tutkimuksen perusteella rakennetaan mobiilisovelluspohja, jonka avulla JL-Soft Oy pystyy toteuttamaan

asiakaslähtöisiä projekteja. Mobiilisovelluspohjan tulee olla mahdollisimman monipuolinen ja mukautua asiakkaan sekä sovelluskehittäjän tarpeisiin toteuttaa projekti.

2 MOBIILI

Mobiililaitteiden myynti on ollut kasvussa ja siten myös mobiilisovellusten määrä on kasvanut huomattavasti. Vuoden 2014 lopussa sovelluksia oli saatavilla Googlen Play -palvelusta yli 1,4 miljoonaa sekä Applen App Storesta yli 1,2 miljoonaa sovellusta (Ariel 2015). On ennustettu, että sovellusmarkkinat tuottavat vuonna 2017 yhteensä 77 miljardia dollaria. Sovelluksia on ennustettu ladattavan yli 268 miljardia. (Clifford 2015.)

2.1 Käyttöjärjestelmät

Mobiilisovelluksen kehityksen suurin haaste on saada sovellus tarjottua mahdollisimman suurelle yleisölle. Käytetyimmät mobiilikäyttöjärjestelmät ovat Googlen kehittämä Android 76,6 %:n markkinaosuudella, Applen kehittämä iOS 19,7 %:n osuudella sekä Microsoftin kehittämä Windows Phone 2,8 %:n osuudella. Muiden käyttöjärjestelmien markkinaosuus on alle prosentin. (IDC 2015.)

2.1.1 Android

Android-käyttöjärjestelmän ensimmäinen versio julkaistiin syyskuussa 2008 versio numerolla 1.0 (Morrill 2008). Ensimmäinen myytävä Android-käyttöjärjestelmällä oleva puhelin oli HTC Dream. Puhelinta myytiin yli miljoona kappaletta Yhdysvalloissa alle vuodessa (T-Mobile 2015). Versio sisälsi kaikki tutut sovellukset nykyisestä versiosta, kuten Android Marketin, web-selaimen sekä suurimman osan Googlen tuottamista palveluista, kuten kalenterin, kartat sekä kontaktit. Tämän jälkeen Google on julkaissut käyttöjärjestelmästään useita versioita, joista viimeisin on julkaistu maaliskuussa 2015 versionumerolla 5.1 (Eason 2015).

Google tarjoaa kehittäjilleen Android Studio -sovelluskehitystyökalun, jonka avulla pystytään kehittämään sovelluksia käyttöjärjestelmälle. Android-käyttöjärjestelmän ohjelmontikieli on Java, mutta sille on myös mahdollista kehittää sovelluksia C- sekä C++-kielillä NDK:n (native development kit) kautta. Kehittämisestä Android-käyttöjärjetelmälle Google

ei peri erillistä maksua, mutta käyttäjän pitää reksiteröityä kehittäjäksi saadakseen kaikki mahdolliset Googlen tarjoamat sovellustoiminnot käyttöönsä.

Android-sovellukset julkaistaan Google Play -kaupassa, jonka nimi oli ennen Google Market. Sovelluksen julkaisemiseen täytyy rekisteröityä julkaisijaksi, joka maksaa 25 dollaria jokaiselta julkaisijalta, mikä on kertamaksu eikä sitä tarvitse uusia. Google ei tarkista itse Play-kauppaan lähetettyjä sovelluksia vaan jättää testauksen vastuun kehittäjälle. (Google 2015e.)

2.1.2 iOS

iOS on Applen kehittämä kosketusnäytöllisiin laitteisiin suunnattu käyttöjärjestelmä. iOS nimi oli alun perin iPhone OS, mutta se muutettiin nykyiseen muotoon maaliskuussa 2010 (Chartier 2015). iPhone OS 1.0 julkaistiin kesäkuussa 2007 (Staff 2015). Ensimmäinen versio oli hyvin rajoitettu, ja siitä puuttui paljon tarvittavia ominaisuuksia, kuten tuki kolmannen osapuolen sovelluksille. Se kuitenkin lisättiin seuraavassa kokonaisessa versionumerossa (Staff 2015). Viimeisin käyttöjärjestelmästä julkaistu versio on iOS 9, joka julkaistiin syksyllä 2015 (Swider 2015).

iOS-sovellukset ohjelmoidaan Applen kehittämällä Objective-C-kielellä. Objective-C:n kehitys aloitettiin jo 1980-luvun alkupuolella, ja sitä kehitetään jatkuvasti eteenpäin. (Singh 2003.)

iOS-sovellukset julkaistaan Applen ylläpitämässä App Storessa, mikä vaatii kehittäjiltä vuosittaisen maksun lukuunottamatta korkeakouluja (Apple 2015a). Jokainen julkaisuun menevä sovellus täytyy hyväksyttää Applella. Apple tarkistaa sovelluksen toimintavarmuuden sekä muita tärkeitä ominaisuuksia. Tällä pyritään takaamaan App Storessa tarjottavan sovellusten laatu. Ei-hyväksytyissä sovelluksista lähetetään tiedote kehittäjälle, missä kerrotaan syy hylkäykselle. Korjausten jälkeen kehittäjä voi lähettää sovelluksen uudestaan hyväksyttäväksi sovelluskauppaan

(Apple 2015a). Hyväksyttämisprosessi kestää viikosta kahteen viikkoon (Shiny Development 2015).

2.1.3 Windows Phone

Windows Phone on verrattain uusi mobiilikäyttöjärjestelmä, jonka on kehittänyt Microsoft. Sen ensimmäinen versio julkaistiin lokakuussa 2010, ja se tunnettin nimellä Windows Phone 7 (Hollister 2015). Tämän jälkeen Microsoft julkaisi seuraavan kokonaisen versionumeron vuonna 2012, Windows Phone 8:n, joka ei ollut yhteensopiva vanhojen laitteiden kanssa laitteistovaatimusten takia (Miles 2015). Viimeisin käytössä oleva versio Windows Phone 8.1 on julkaistu huhtikuussa (Warren 2015). Microsoft on kuitenkin julkistanut tammikuussa 2015 tulevan Windows 10 version.

Windows Phone -sovellukset kehitetään Microsoftin kehittämällä .NET Frameworkillä, jonka ensimmäinen versio julkaistiin vuonna 2002 (Wikipedia 2015). .NET Frameworkin viimeisin versio 4.5.2 on julkaistu toukokuussa 2014 (Microsoft 2014). Sovellusten kehittäminen käyttöjärjestelmälle on ilmaista.

Windows Phone -sovellukset julkaistaan Microsoftin Windows Phone Storessa, johon täytyy rekisteröityä kehittäjäksi. Rekisteröityminen maksaa yksityisille henkilöille noin 19 dollaria ja yrityksille noin 99 dollaria (Microsoft 2015a). Sovellukset tarkistetaan Microsoftin toimesta (Microsoft 2015c).

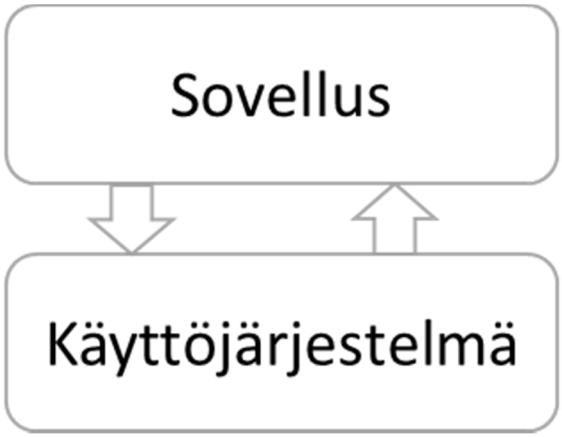
2.2 Mobiilisovellustyypit

Mobiilisovelluksista on olemassa kolmea eri päätyyppiä: natiivi, web ja hybrid. Ennen sovelluskehityksen aloitusta täytyy miettiä kohdeyleisön laajuutta, budjettia, aikataulua. Jokaisella eri päätyypillä on omat hyvät sekä huonot puolensa, jotka vaikuttavat projektin onnistumiseen.

2.2.1 Natiivisovellukset

Natiivisovellukset ovat perinteinen tapa toteuttaa mobiilisovelluksia. Niissä käytetään käyttöjärjestelmän tukemaa ohjelmointikieltä, jolloin kaikki käyttöjärjestelmän sekä laitteiston tarjoamat toiminnallisuudet ovat sovelluksen käytössä.

Natiivisovelluksilla pystytään tekemään hyvinkin monimutkaisia sekä laitekohtaisia ratkaisuja, koska ohjelma keskustelee suoraan laitteen kanssa kuvion 1 tapaan. Kehitettäessä natiivisovellusta täytyy jokaiselle tuetulle käyttöjärjestelmälle tehdä oma sovellus, mikä lisää kehittäjien työtä. Jokainen sovelluksen toiminnallisuus täytyy suunnitella huolella ja toteuttaa erikseen käyttöjärjestelmille. Natiiviohjelmilla saadaan yleensä tuotettua parempi kokemus loppukäyttäjälle, koska ohjelma on käyttöjärjestelmän standardien mukainen. Natiivisovellusten suorituskyky on muita sovellustyyppejä parempi, koska ne on suunniteltu tietylle käyttöjärjestelmälle. (Oksman 2015.)

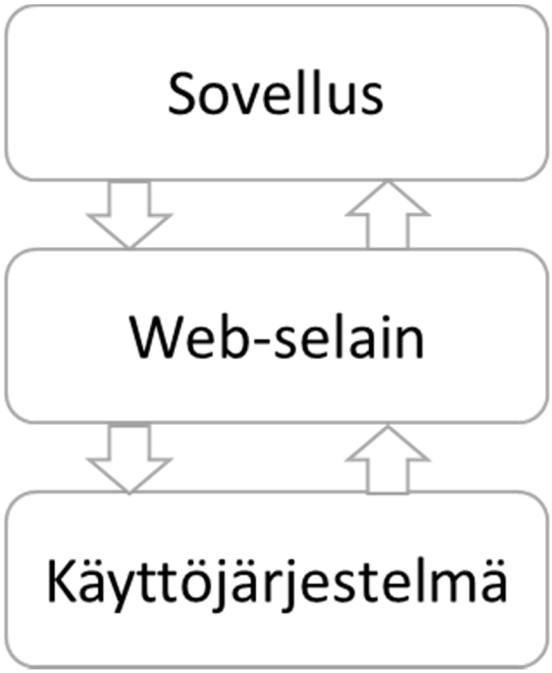


KUVIO 1. Natiivisovellus arkkitehtuuri

Sovellusten kehittäminen tehdään yleensä käyttöjärjestelmän valmistajan tarjoamalla sovelluskehitysalustalla, mikä antaa hyvät työkalut sovelluksen testaamiseen oikeassa ympäristössä. Tämä kuitenkin vaatii kehittäjältä erikoisosaamista kyseisestä käyttöjärjestelmästä, koska toteutukset ovat käyttöjärjestelmä kohtaisia eikä niitä välttämättä pysty soveltamaan toisissa järjestelmissä.

2.2.2 Web-sovellus

Web-sovellukset ovat yleisesti toteutettu käyttäen HTML-, CSS- sekä JavaScript-teknologioita, joten ne eivät poikkea tavallisesta web-sivusta. Web-sovellukset hyödyntävät käyttöjärjestelmän selaimen ominaisuuksia kuvion 2 tapaan. Selainten tekniikat on standardoitu kolmannen osapuolen kautta, mikä mahdollistaa sovelluksen käyttämisen mahdollisimman monessa eri käyttöjärjestelmässä. (Oksman 2015.)



KUVIO 2. Web-sovelluksen arkkitehtuuri

Käyttöjärjestelmien selaimien toteutuksissa on kuitenkin eroja, mitkä pitää ottaa sovelluskehityksessä huomioon. Toisella käyttöjärjestelmällä tai samalla käyttöjärjestelmän eri laittaiden välillä saattaa olla huomattaviakin eroja siinä, kuinka hyvin standardeja on tulkittu ja toteutettu. (Oksman 2015.)

Web-sovelluksille on tarjolla paljon erilaisia käyttöliittymä- ja toimintakirjastoja, kuten jQuery, AngularJS, Twitter Bootstrap. Käyttöliittymäkirjaston käyttö helpottaa kehittäjän työtä tekemään toimivampia sovelluksia, koska kirjastot toteuttavat samat toiminnot riippumatta käyttöjärjestelmän selaimesta. Yleisesti näille kirjastoille on

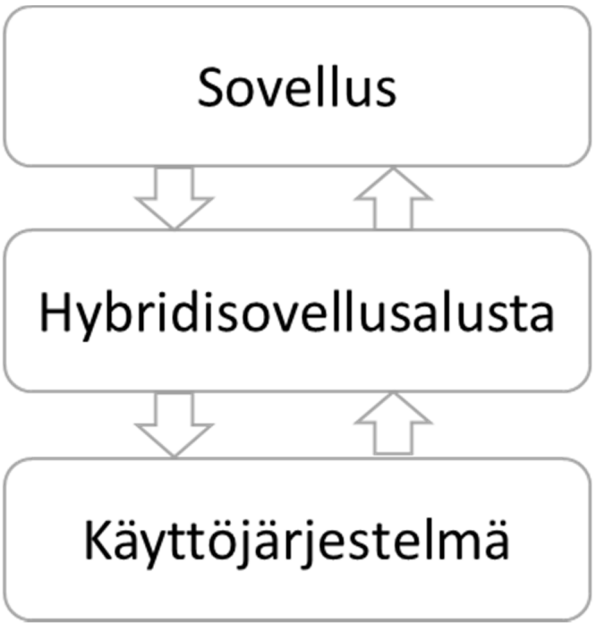
saatavilla hyvä dokumentaatio sekä yhteisön tuki eri keskustelupalstojen kautta. (Oksman 2015.)

Web-sovelluksia pystytään kehittämään millä tahansa teksinmuokaimella, mutta yleisesti käytetään jotain ohjelmistoympäristöä, kuten Netbeansia tai Eclipseä. Ohjelmointiympäristöistä löytyy samanlaisia testaus- ja kehitys-aputoimintoja kuin natiivisovelluksien ohjelmointiympäristöissä.

2.2.3 Hybridisovellukset

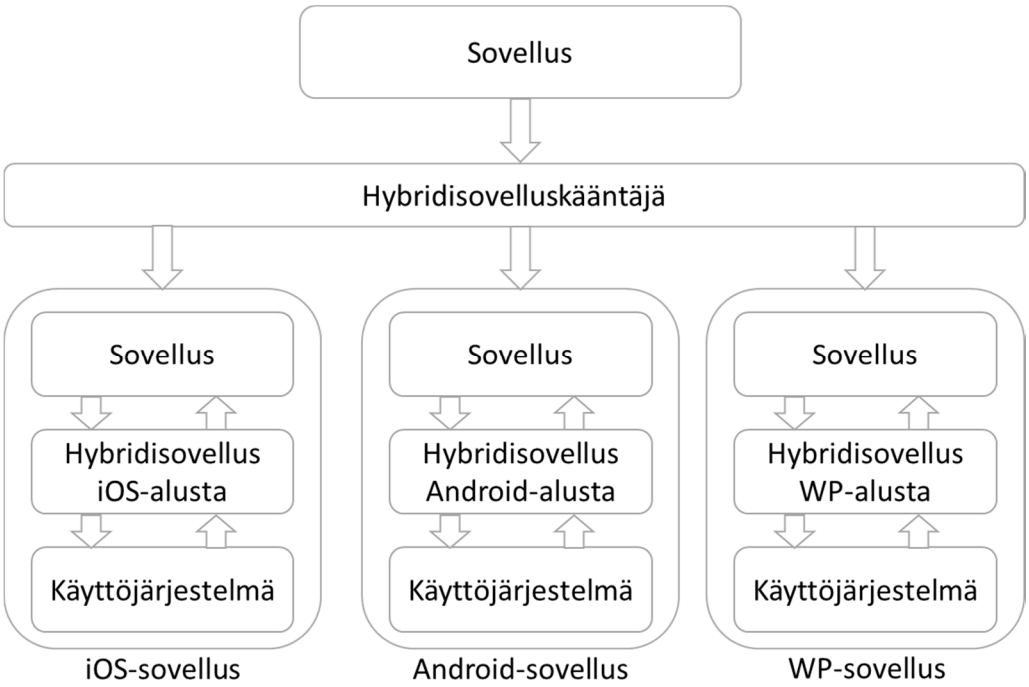
Hybridisovellustoteutuksia on olemassa erilaisia toteutuksia, mutta niiden periaate on aina sama: saada sovellus pienellä vaivalla usealle eri käyttöjärjestelmälle. Tällöin saadaan sovellus nopeammin käytettäväksi suuremmalle käyttäjäkunnalle. Yksi suosituimmista tavoista tehdä hybridisovelluksia on web-pohjainen hybridisovellus. Siinä laajennetaan web-sovellusta tarjoamalla toimintoja, joita natiivisovelluksella on saatavilla (Salesforce 2015). On olemassa muitakin tapoja toteuttaa hybridisovellus, esimerkiksi Xamarin tarjoaa mahdollisuuden toteuttaa sovelluksen C#-kielellä (Xamarin 2015).

Hybridisovelluksessa sovellusta ajetaan hybridisovellusalustan päällä kuvion 3 tapaan. Alusta tarjoaa universaalin rajapinnan käyttäjärjestelmän tarjoamiin palveluihin. Tällöin samaa sovellusta pystytään ajamaan eri käyttöjärjestelmillä ilman muutoksia itse sovellukseen. Sovelluksen käännösvaiheessa siihen liitetään käyttöjärjestelmäkohtainen alusta.



KUVIO 3. Hybridisovelluksen toimintaperiaate

Hybridisovellukset mahdollistavat sovellusten julkaisemisen usealle eri käyttöjärjestelmälle mahdollisimman pienellä vaivalla. Sovellukset kääritään eri säiliöihin riippuen julkaistavasta alustasta kuvion 4 tapaan. Tämä mahdollistaa sovelluksen kutsujen välittämisen käyttöjärjestelmälle ilman että jokaiselle käyttöjärjestelmälle luotaisiin oma natiivisovellus. Esimerkiksi sovelluksen on mahdollista pyytää telemetrisiä tietoja käyttöjärjestelmältä yhdellä kutsulla jokaisessa käyttöjärjestelmässä. (Telerik 2015.)



KUVIO. 4 Hybridisovelluksen kääntäminen eri käyttöjärestelmille

Web-sovelluspohjaiset hybridisovellukset käynnistetään riisutussa kokoruudun selaimessa, jossa selaimen osoiterivi sekä sivuhistorian selaaminen ovat piilotettuna. Tällöin jää vain HTML- ja CSS-koodia prosessoiva kontrolli näkyville, jolloin saadaan web-sovellus naamioitua näyttämään natiivilta sovellukselta eikä loppukäyttäjä huomaa eroa toteutuksessa. (Telerik 2015.)

3 MOBIILISOVELLUKSET

3.1 Sovelluksen ja palvelimen välinen kommunikointi

Sovelluksen ja palvelimen kommunikoinnin toteuttamiseen on monta eri tapaa. Yleisesti mobiilisovelluksisa on kaksi eri toteutusta tiedon välittämiseen: sovellus hakee tiedon palvelimelta tai palvelin lähettää sovellukselle tietoa.

3.2 Palvelimelta tiedon hakeminen

Palvelimelta tiedon hakemiseen yleisesti mobiilisovelluksissa käytetään HTTP-protokollaa hyödyntäviä ratkaisuja, koska ne toimivat järjestelmästä riippumatta. Yleisimpiä toteutuksia on REST-rajapinnat sekä SOAP-etäkutsut.

3.2.1 SOAP

SOAP eli Simple Object Access Protocol mahdollistaa RPC-etäkutsujen tekemisen palvelimelle useiden protokollien yli. Sitä kuitenkin yleisimmin käytetään HTTP-protokollan kanssa toteutettaessa internetpalveluja. SOAP pohjautuu yksinkertaiseen XML-RPC-protokollaan ja toimii sen jatkeena.

SOAP-kutsut on muotoiltu XML-merkintäkielellä, ja ne ovat luettavissa selkokielisinä ilman erillistä ohjelmaa. Viestien rakenne on sama kutsussa sekä vastauksessa. Viesti on jaoteltu neljään eri osaan: kuoreen, otsikkoon, runkoon ja virheeseen. Näistä pakollisia jokaisessa kutsussa on kuori, joka käärii koko viestin, sekä runko, joka sisältää suoritettavan komennon. Ylätunnisteeseen kirjataan mahdolliset kirjautumistiedot sekä muu yleinen tieto kutsun suorittamiseen. Virheosioon voidaan kirjata virheen sattuessa siihen liittyvät tiedot. Kuvioissa 5 ja 6 on esimerkki asiakassovelluksen sekä palvelinsovelluksen välisestä kommunikaatiosta minimissään.

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

KUVIO 5. Esimerkki SOAP kutsu HTTP-protokollalla (W3Schools 2015)

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

KUVIO 6. Esimerkki SOAP vastaus HTTP-protokollalla (W3Schools 2015)

SOAP kehitettiin alun perin Microsoftille vuonna 1998 (Ferguson 2004), jonka jälkeen se siirrettiin World Wide Web Consortiumin ylläpitämäksi vuoteen 2009 asti. Nykyisin alkuperäistä SOAP:aa ei enää kehitetä sellaisenaan (W3C 2009). Sen jatkeeksi on rakennettu lisäosia, kuten SwA (SOAP with Attachments). SwA lisää mahdollisuuden lähettää liitteitä SOAP-viestin perässä.

3.2.2 Palvelimen tiedon lähettäminen

Android, iOS ja Windows Phone tukevat Push-viestien lähettämistä suoraan puhelimeen eri pilvipalveluiden avulla. Tämän tekniikan

käyttäminen tekee sovelluksen toiminnasta tehokkaampaa. Ilman Push-viestejä sovellus joutuisi ajoittain lähettämään kyselyitä palvelimelle tarkistaakseen uuden tiedon, mikä kuluttaa paljon puhelimen resursseja ja johtaa akun tyhjenemiseen nopeasti. (Google 2015d.)

```
{
  "registration_id" : "APA91bHun4MxP5egoKMwt2KZFBaFUH-1RYqx...",
  "data" : {
    "Nick" : "Mario",
    "Text" : "great match!",
    "Room" : "PortugalVSDenmark",
  },
}
```

KUVIO 7. Esimerkki Google Cloud Messaging -viestistä

Esimerkiksi Google on toteuttanut Android-käyttöjärjestelmissä Push-viestit Google Cloud Messaging -palvelun avulla. Palveluun voidaan lähettää viestit joko käyttäen HTTP-rajapintaa tai XMPP-yhteyttä. Viestit ovat rakenteltaan kuvion 7 mukaisesti jäsenneltyjä JSON-merkkijonoja, jotka sisältävät vastaanottajan ja lähetettävän datan. Sovelluksen puolella täytyy rekisteröityä vastaanottamaan viestejä, minkä jälkeen sovellus voi vastaanottaa viestejä yksilöllisellä tunnisteella. Google tukee maksimissaan 4 kilobitin kokoisia viestejä, mikä rajoittaa viestit vain informatiivisiin viesteihin. Kaikki tätä suuremmat viestit täytyy hakea palvelimelta toisella protokollalla. (Google 2015d.)

On olemassa myös kolmannen osapuolen tekemiä sovelluksia, joiden avulla kehittäjän ei tarvitse tehdä Push-viesti-integraatiota jokaiselle käyttöjärjestelmälle erikseen. Riittää kun palvelinsovellus lähettää viestinsä palveluun, niin palvelu toimittaa viestin perille mobiililaitteelle. (Pushwoosh 2015.)

3.3 Tiedon tallentaminen

Kuten kaikissa sovelluksissa, on tiedon esittäminen tärkeä osa niiden toiminnallisuutta. Mobiilisovelluksisa tietoa voidaan tallentaa kahteen eri paikkaan, joko suoraan puhelimen tiedostojärjestelmään tai puhelimen tietokantoihin. Tiedon tallentamista hyödynnetään silloin, kun tieto pitää

olla heti saatavilla, vaikka verkkoyhteyttä ei olisi käytössä. Tieto on saatavilla sovelluksen uudelleenkäynnistyksen jälkeenkin, koska se tallennetaan puhelimen sisälle. Tiedon tallentamista myös kannattaa tehdä niissä tapauksissa, jos tietoa tarvitaan uudelleen. Tällöin pystytään vähentämään sovelluksen käyttämää tiedonsiirtomäärää. (Wikipedia 2015.)

3.3.1 SQL-tietokannat puhelimessa

Markkinoiden suurimmat käyttöjärjestelmät tarjoavat natiivi- ja hybridisovelluksiin mahdollisuuden tallentaa tietoa tietokantoihin. Niiden toteutukset poikkeavat toisistaan, mutta toteuttavat kuvion 8 mukaista SQL-lauseita, mikä helpottaa sovelluskehitystä. Uusimmissa selaimissa on myös saatavilla WebSQL-tietokanta, mikä mahdollistaa web-sovellusten tietokantakyselyt. WebSQL:n kehitys on kuitenkin lopetettu. (W3 2010.)

```
SELECT _id, name FROM sellers;
```

KUVIO 8. Esimerkki SQL-kysely

Tietokantoihin tallentaminen kannattaa silloin, kun tietoa pitää jäsennellä sekä sovellukseen halutaan nopeutta. Niillä pystytään optimoimaan sovelluksen suorituskykyä analysoimalla tallennettua tietoa, mikä vaatii tiedon jäsentelyä ennen tietokannan perustamista. Jos tieto ei ole jäsenneltävissä, se kannattaa tallentaa puhelimen levylle. (Burns 2015.)

3.3.2 Puhelimen tiedostojärjestelmät

Jos sovelluksessa tarvitaan suurempia tiedostoja, ne kannattaa tallentaa puhelimen tiedostojärjestelmään. Näitä ovat esimerkiksi suuret liitetiedostot, kuten kuvat.

Natiivisovelluksissa tiedon tallentaminen on mahdollista suoraan puhelimen muistiin, mutta web-sovelluksissa käytetään selaimen tarjoamaa paikallista tallennustilaa tiedon pidempiaikaiseen säilyttämiseen (Whitney 2012). Nykyisin sovelluksen tallentama tietomäärä ei ole

ongelma, koska puhelimien muistiin pystytään tallentamaan useita gigatavuja tietoa (Voo 2015).

3.4 Käyttöliittymäsuunnittelu

Suurimmassa osassa nykyisissä mobiililaitteista on käytössä kosketusnäyttö (CDS 2015), joka pitää ottaa sovelluksen käyttöliittymää suunniteltaessa huomioon. Kun käytössä ei ole tavallista hiirtä, niin sovelluksen käyttö ei ole yhtä tarkkaa kuin tietokoneilla käytettäessä. Käyttöliittymän kontrolleista täytyy tehdä käyttäjän sormelle sopivan kokoisia (Anthony 2015). Käyttöliittymäsuunnittelussa voidaan myös hyödyntää laitteen kiihtyvysantureita, jolloin pystytään esimerkiksi suorittamaan toiminto laitteen heilautuksen yhteydessä.

3.4.1 Kontrollien koko

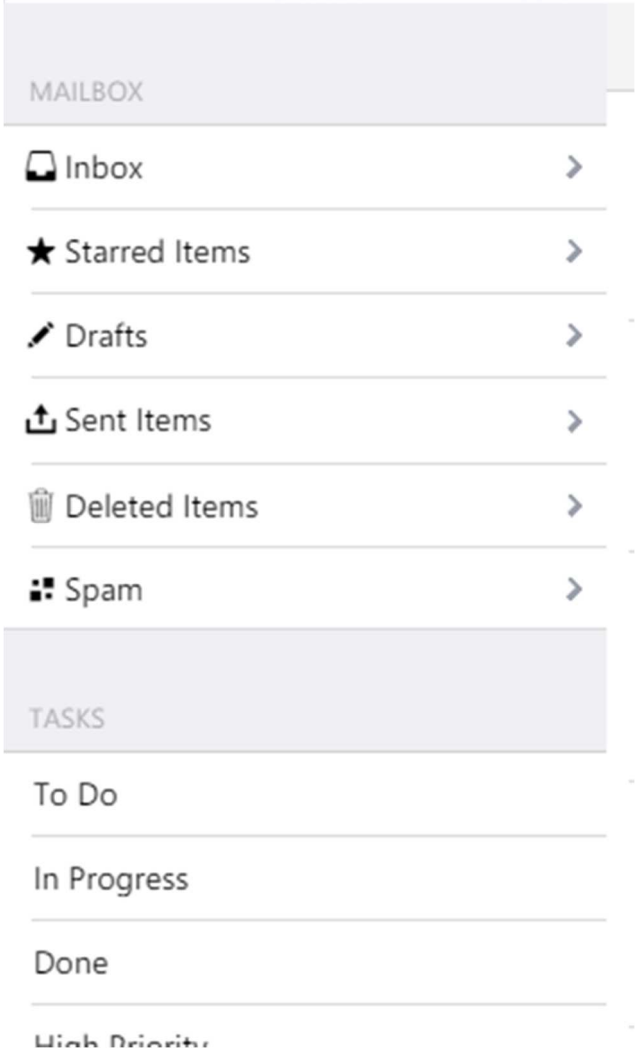
Sovellusten kontrollit eivät voi olla liian pieniä eivätkä liian lähellä toisiansa, koska se voi aiheuttaa virhepainalluksia käyttäjältä. Eri käyttöjärjestelmien sovellussuosituksissa on eroja kontrollien ko’oille, esimerkiksi Apple suosittelee kontrollin minimikorkeudeksi ja –leveydeksi 44 pikseliä kun taas Windows Phone suosittelee 26 pikseliä. Pikselikoko ei ole kuitenkaan verranollinen todellisuuden kanssa, koska näytöt voivat olla erikokoisia ja niiden pikselitiheydet vaihdella. (Anthony 2015.)

3.4.2 Toiminnot

Kiinteiden toimintopainikkeiden määrä ja niiden toiminallisuus vaihtelevat käyttöjärjestelmien kesken, jolloin puuttuvat toiminnot täytyy huomioida käyttöliittymissä. Käyttöjärjestelmien käyttöliittymäsuosituksissa on kuitenkin määritelty, mitkä toiminnot käyttöliittymistä pitäisi löytyä. Esimerkiksi iOS-käyttöjärjestelmässä on etunäytössä vain kotinäppäin, jonka avulla käyttäjä voi siirtyä käyttöjärjestelmän kotinäyttöön. Taaksepäin siirtyäkseen täytyy iOS-käyttöliittymään lisätä erillinen taakse-toimintopainike (Apple 2015b), kun taas kyseinen painike on Windows

Phone-käyttöjärjestelmässä pakollinen (Microsoft 2015b) ja Android-käyttöjärjestelmässä vaihtoehtoinen, mutta yleisesti käytetty (Google 2015a).

Laitteen näyttö voi myös tunnistaa eleitä, kuten pitkiä painalluksia sekä useampia sormia kerralla. Ruudun hipaisulla sivusta voidaan tuoda esimerkiksi kuvion 9 tapainen vetovalikko, jonka taakse voidaan lisätä sovelluksessa tarvittavia toimintoja. Uusimmissa laitteissa voidaan tunnistaa käyttäjän painalluksen voimakkuus, jolloin pystytään esimerkiksi näyttämään lisätietoja painetusta tiedosta.



KUVIO 9. Kendon vetovalikko

Koska ruudun koko rajoittaa käyttöliittymän suunnittelua sekä käyttöliittymän kontrollien koko täytyy olla tavallista suuremmat, eleitä tulisi

hyödyntää käyttöliittymän suunnittelussa mahdollisimman paljon. Tällöin saadaan käyttäjälle näkyville oleellinen tieto ja tieto, jota käyttäjä ei välttämättä tarvitse juuri sillä hetkellä, piilotettua. (Creative Workline 2015.)

4 HYBRIDISOVELLUKSEN OSAT

Hybridisovellus muodostuu yhdestä tai useammasta komponentista. Osa hybridisovellusalustoista tarjoaa kaiken tarpeellisen sovelluksen rakentamiseen, mutta näiden heikkous sekä vahvuus on rajallisuus. Monipuolisimpia sovelluksia pystytään rakentamaan, kun käytetään mahdollisimman suosittuja tekniikoita, joita ovat esimerkiksi HTML-, CSS- sekä JavaScript-tekniikat. Nämä tekniikat ovat tällä hetkellä suosituimmat sekä helpoin tapa rakentaa sovelluksia internetiin laajan tuen takia. Jotta näitä tekniikoita voidaan hyödyntää niin tarvitaan vähintään tekniikoita tukeva alusta. Tarvittaessa voidaan käyttää käyttöliittymä- sekä toiminnallisuus-kirjastoja. Näitä ovat esimerkiksi AngularJS sekä Twitter Bootstrap, joiden avulla pystytään helpottamaan sovelluksen rakennetta.

4.1 Cordova

Cordova on Apache Software Foundationin ylläpitämä avoimen lähdekoodin hybridisovellusalusta, joka tunnetaan myös vanhalla nimellään PhoneGap. Cordova tukee useita eri alustoja mobiililaitteista aina tavallisiin tietokoneisiin. Sovelluskehityksessä käytetään HTML5-, CSS- sekä JavaScript-tekniikoita. (Cordova 2015.)

Cordovan toimintaperiaate perustuu riisuttuun selaimen näkymään. Cordova luo koko ruudun peittävän WebView-näkymän, jonka avulla se pystyy esittämään nattiivisovellusta, vaikka kysessä on tavanomainen HTML-sivu. Sivua pystyttäisiin selaamaan myös käyttöjärjestelmän omalla web-selaimella, mutta koska se on käännetty Cordova sovellukseksi, niin sillä on pääsy käyttöjärjestelmän tarjoamiin palveluihin, kuten esimerkiksi kameraan. (Cowart 2013.)

Cordova ei sisällä ohjelmointiympäristöä vaan sille voidaan kehittää sovelluksia käyttämällä samoja työkaluja kuin web-sovelluskehityksessä. Sovellukset käännetään käyttöjärjestelmille yksinkertaisen komentorivi-käyttöliittymän kautta, joka on saataville Windows-, Linux- sekä OS X -käyttöjärjestelmille. Sovelluskehitykseen käytettävä käyttöjärjestelmä

kuitenkin rajoittaa sovelluksen kääntämistä mobiilikäyttöjärjestelmille. Esimerkiksi Windows-käyttöjärjestelmällä ei pystytä kääntämään sovelluksia iOS-käyttöjärjestelmälle. Adobe tarjoaa tähän ratkaisun kääntämällä sovelluksen pilvipalvelussa, jolloin kehittäjän ei tarvitse erikseen hankkia kääntämistä varten erillistä laitteistoa. (Cordova 2015.)

4.1.1 Lisäosat

Cordovan toiminta perustuu perusalustaan ja sen ympärille rakennettuihin lisäosiin. Se tarjoaa yleisesti käytettyihin toimintoihin lisäosia, kuten kameran, tallennustilan ja verkkotilatietojen käyttöön. Cordovaan on kuitenkin mahdollista kehittää kolmannen osapuolen lisäosia sen standardoidun lisäosa-tekniikan myötä. Jokainen lisäosa täytyy toteuttaa erikseen tuetulle käyttäjärjestelmille. (Cordova 2015.)

Lisäosat toteuttavat Cordovan Plugman rajapinnan, jonka avulla pystytään määrittämään tarvittavat tiedostot jokaiselle käyttöjärjestelmälle erikseen. Tämä mahdollistaa lisäosien yksinkertaisen lisäämisen suoraan pilvipalveluista(repository) osaksi projektia. Cordova ylläpitää tietokantaa lisäosista, joita pystyy lisäämään projektiin. (Cordova 2015.)

Kun lisäosa on lisätty projektiin Cordova lisää käännösvaiheessa tarvittavat JavaScript-funktiot, joiden avulla sovellus pystyy tekemään kutsuja käyttöjärjestelmälle. Kutsut toteutaan kuvion 10 mukaisesti takaisinkutsuina, jolloin täytyy toteuttaa erilliset funktiot onnistuneelle ja epäonnistuneelle kutsulle. (Cordova 2015.)

```
navigator.camera.getPicture(function(imageURI) {
    // Onnistunut käyttöjärjestelmä kutsu
  }, function(err) {
    // Epäonnistunut käyttöjärjestelmä kutsu
  }, cameraOptions);
```

KUVIO 10. Cordova-sovelluksen tapa kutsua käyttöjärjestelmää

4.2 AngularJS

AngularJS on avoimeen lähdekoodiin perustuva sekä Googlen ylläpitämä JavaScript-kirjasto. AngularJS ei ole alun perin Googlen projekti vaan sen ensimmäinen versio julkaistiin vuonna 2009, jonka jälkeen Google otti projektin ylläpidettäväksi. (Google 2015b.)

AngularJS on tarkoitettu käytettävän web-selaimissa suoritettavissa yksisivuissa sovelluksissa. Se tarjoaa kehittäjälle MVC- (Model-View-Controller) sekä MVVM (Model–view–viewmodel) -arkkitehtuurit, mutta sillä ei voida kehittää palvelinpuolen ratkaisuja. AngularJS pyrkii helpottamaan sovelluksen kehitystä sekä testausta tarjoamalla selkeät laajennukset sekä mahdollisuuden matkia laajennuksia testauksessa. (Google 2015b.)

4.2.1 Direktiivit

AngularJS pyrkii laajentamaan jo olemassa olevaa HTML-syntaksia lisäämällä siihen merkintöjä. AngularJS lukee koko HTML-sivun läpi käynnistyessään ja tekee ennalta määrätyt muutokset tarvittaviin merkintöihin HTML-koodissa sekä piirtää sivun uudestaan uusilla tiedoilla. Näitä merkintöjä kutsutaan direktiiveiksi. (Google 2015c.)

AngularJS etsii direktiivejä HTML-elementeistä, attribuuteista, CSS-luokista sekä kommenteista. Se tarjoaa jo suuren määrän valmiita direktiivejä, jotka erotetaan ng-etuliitteellä. Sovelluskehittäjät voivat kehittää uusia direktiivejä laajentamaan sovelluksia. Tyypillisesti AngularJS-direktiivejä käytetään laajentamaan kontrolleja sekä toistuvia asioita sivulla. (Google 2015c.)

AngularJS-direktiivi määritellään AngularJS-moduuliin. Direktiivin alustuksessa kerrotaan direktiivin nimi, johon viitataan HTML-koodissa. Direktiivin nimi on kirjoitettu camelCase-tyylillä, eli sen ensimmäinen kirjain on pieni ja seuraavat sanat on isoilla kirjaimilla. Koska HTML:n attribuuteissa ei erotella isoa tai pientä kirjainta, direktiivin nimi täytyy

HTML-koodissa kirjoittaa ison kirjaimen kohdalta väliviivalla. HTML-koodissa kuvion 11 direktiiviin viitataan attribuutilla ”hello-world”.

```
angular.module('helloWorld', []).directive('helloWorld', function() {
    return {
        restrict: 'E',
        template: 'Hello world!'
    }
})
```

KUVIO 11. Esimerkki AngularJS-direktiivistä

4.2.2 Näkymät

Näkymällä AngularJS-sovelluksessa tarkoitetaan sovelluksen sivun osaa, jota ohjataan direktiiveillä, kontrollereilla sekä moduuleilla. Näkymät ovat HTML-sivun osia, jotka sisältävät AngularJS-sovellukselle tyypillisä elementtejä. (Google 2015g.)

Yhdellä HTML-sivulla on mahdollista olla useampi näkymä käyttämällä ng-view-direktiiviä. Kuvion 12 tapaan sovelluksesta tehdään yksivuinen määrittelemällä päänäkymälle JavaScript- sekä CSS-tiedostot, jolloin niitä ei tarvitse ladata kun näkymä vaihtuu. Tämän jälkeen voidaan märittää reititys-moduulilla näkyvä näkymä, jolloin sovellus vaikuttaa monisivuiselta ohjelmalta käyttäjällä, vaikka todellisuudessa se on yksisivuinen.

```
<html ng-app>
    <head>
        <title>Sample app</title>
    </head>
    <body>
        <div ng-view>

        </div>
    <script type="text/javascript" src="js/angular.min.js"></script>
    </body>
</html>
```

KUVIO 12. Esimerkki yksisivuisen AngularJS-sovelluksen päänäkymästä

4.2.3 Kontrollerit

AngularJS-näkymän ydintoiminnot rakennetaan kontrolleihin. Yhdellä sivulla voi olla useampia eri kontrolleja sekä kontrollit voivat sijaita sisäkkäin, jolloin ne näkevät toisensa ja pystyvät vaihtamaan tietoa sekä lähettämään tapahtumia. Tämä helpottaa sovellusten rakennetta, koska sovellus pystytään jakamaan sivulla pienempiin osiin. (Google 2015h.)

Jokainen kontrolleri sisältää scope-muuttujan. Angular mahdollistaa kontrollerin tiedon sitomisen näkymään scope-muuttujan avulla. Tätä kutsutaan kaksisuuntaiseksi sitomiseksi (engl. Two-way-binding). Jos kontrolleri muuttaa scope-muuttujassa tietoa niin tieto siirtyy automaattisesti sivulle näkyviin. Tämä helpottaa kehittäjän työtä, koska jokaisen muutoksen yhteydessä ei tarvitse manipuloida HTML-sivua. (Google 2015h.)

Kontrollerille ei ole tarkoitus kasata sovelluksen ydintoimintoja. Kontrollerille tulisi laittaa vain yhdelle sivulle tarkoitettu liiketoiminalle tärkeä logiikka. Kontrollerit eivät säilytä tietoa sivunvaihdon yhteydessä, vaan ne tuhotaan sovelluksen muistista, kun ne poistuvat sivulta käytettävistä. Jos kontrollerin tulee säilyttää tietoa, niin se täytyy tehdä palveluiden avulla.

4.2.4 Palvelut

Palveluilla pystytään tarjoamaan yhdelle tai useammalle kontrollerille, direktiiville tai toiselle palvelulle palveluita. Ne sisältävät metodeja, joita pystytään uudelleen käyttämään, mikä selkeyttää sovelluksen rakennetta. Esimerkiksi tyypillisesti kommunikaatio palvelimen ja sovelluksen välille rakennetaan palveluna, koska sitä käytetään useammalla kontrollerilla. Palvelut ovat singletoneja, eli niistä on olemassa vain yksi instanssi koko sovelluksessa. Kuviossa 13 on esimerkki palvelun määrittelystä, jonka avulla voidaan tehdä yhteenlasku. (Google 2015f.)

```
angular.module('helloWorld', []).service('helloWorld', [function() {
    this.calculate = function($a, $b) {
        return $a + $b;
    };
}])
```

KUVIO 13. Esimerkki AngularJS-palvelusta

Palvelut injektoidaan kontrollerin käyttöön sen alustuksessa käyttämällä riippuvuus-injektointia, eli kontrolleri pyytää käyttöönsä tarvittavat palvelut. Eri kontrolleissa käytettävä sama palvelu on todellakin sama. Palvelun kautta kontrollerit pystyvät keskustelemaan keskenään vaikka se ei olisi muuten mahdollista. (Google 2015f.)

4.2.5 Moduulit

AngularJS-sovellukset koostuvat yhdestä tai useammasta moduulista. Moduuli voi sisältää direktiivejä, kontrollereita, palveluita sekä muita ominaisuuksia. Moduulin ei ole pakko olla kokonainen sovellus, vaan tyypillisesti eri kokonaisuudet sijoitetaan eri moduuleihin, jotka kasataan sitten sovelluksen päämoduulissa. Tämä helpottaa sovelluksen rakennetta sekä testaamista eri tilanteissa. (Google 2015i.)

AngularJS-sovelluksessa HTML-sivulla viitataan moduuliin ng-app -direktiivillä kuvion 14 tapaan, mikä aloittaa sovelluksen alustamisen (Google 2015i).

```
<html ng-app>
    <head>
        <title>Sample app</title>
    </head>
    <body>
        <input type="text" ng-model="name"
            placeholder="Insert name...">
        <p>Hello {{ name }}</p>
    <script type="text/javascript" src="js/angular.min.js"></script>
    </body>
</html>
```

KUVIO 14. Esimerkki AngularJS-sovelluksesta

4.3 Twitter Bootstrap

Twitter Bootstrap kehitys on aloitettu Twitterin puolesta vuonna 2010, ja se on kasvanut yhdeksi suosituimmaksi avoimen lähdekoodin käyttöliittymäkirjastoksi. Kirjasto pyrkii mukautumaan aina ruudun koon mukaan, jolloin sen avulla pystytään rakentamaan sivustoja niin mobiililaitteille kuin tietokoneille. (Bootstrap 2015.)

4.3.1 Ruudukkojärjestelmä (Grid system)

Ruudukkojärjestelmällä on mahdollista sovittaa sama sivu usealle eri näyttölle. Ruudukkojärjestelmä koostuu taulun tapaan riveistä ja soluista, mikä helpottaa sivun organisointia. Yksi rivi koostuu 12 solusta. Solun sisälle on mahdollista muodostaa uusi rivi kuvion 15 mukaisesti, jolloin solua saadaan jaettua vielä pienempiin osiin. Rivejä sekä soluja yhdistelemällä pystytään tekemään laajasti mukautuva sivusto eri näyttölaitteille.

```
1  <html>
2      <head>
3          <title>Bootstrap Grid</title>
4      </head>
5      <body>
6          <div class="row">
7              <div class="col-sm-6">
8                  <div class="row">
9                      <div class="col-sm-6">
10                         <p>25% of page content</p>
11                     </div>
12                     <div class="col-sm-6">
13                         <p>25% of page content</p>
14                     </div>
15                 </div>
16             </div>
17             <div class="col-sm-6">
18                 <p>Half page content</p>
19             </div>
20         </div>
21     </body>
22 </html>
```

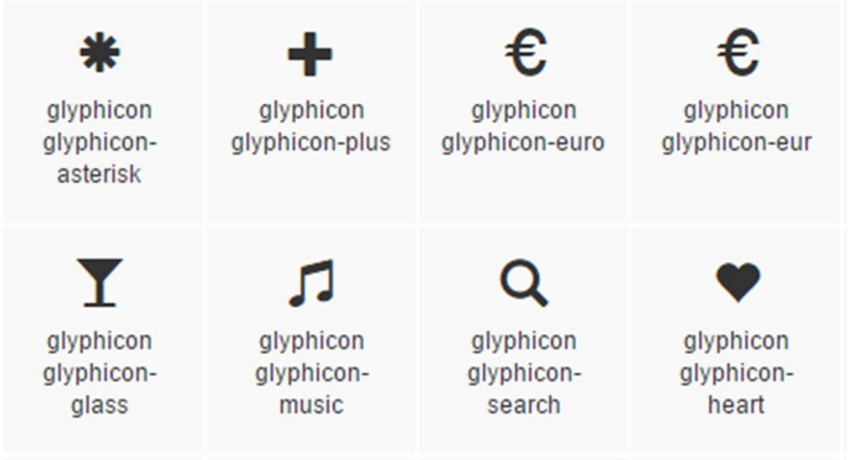
KUVIO 15. Esimerkki Bootstrapin ruudukkojärjestelmästä

Bootstrapissä on määritetty neljä eri näyttökokoa: todella pieni, pieni, keskikokoinen sekä suuri. Todella pienillä tai pienillä näytöillä tarkoitetaan mobiililaitteita, kun taas keskikoisia sekä suuria näyttöjä käytetään pöytätietokoneissa. Tämä mahdollistaa tiedon esittämisen aina näytölle sopivassa formaatissa. Käytännössä, jos näytön koko on pienempi kuin solulle suunniteltu leveys, niin sen sijaan, että solut olisivat vierekkäin, ne siirtyvät allekkain. Soluille on myös mahdollista määrittää useampi eri näyttökoko, jolloin niiden leveys kasvaa ja pienenee näyttökoon mukaan.

4.3.2 Komponentit

Bootstrap sisältää suuren määrän valmiita komponentteja käytettäväksi sekä useat yhteisön jäsenet ovat kehittäneet laajennuksia sekä lisäyksiä näihin komponentteihin. Komponentit mukailevat Bootstrapin syntaksia sekä tyyliä (Bootstrap, 2015). Komponenttien tarkoitus on tarjota valmiita kontrolleja sivulle käytettäväksi, jotka on testattu toimivaksi eri selaimilla.

Bootstrap tarjoaa myös suuren määrän kuvion 16 mukaisia symboleita. Symboolit eivät ole Bootstrapin kehittämiä vaan glyphicons.com -sivusto tarjoaa ne vapaasti käytettäväksi Bootstrapin yhdeydessä. Symboleita voi käyttää Boostrap-sivustolla määrittelemällä elementille luokaksi halutun symbolin css-luokat kuvion 17 esimerkin mukaisesti. (Bootstrap 2015.)



KUVIO 16. Bootstrapin mukana tulevia symboleja

```
1   <span class="glyphicon glyphicon-search" aria-hidden="true"></span>
```

KUVIO 17. Esimerkki Boostrapin symbolin käytöstä

5 MOBIILISOVELLUKSEN TOTEUTUS

Mobiilisovellus toteutettiin JL-Soft Oy:lle eri asiakasporjekteja varten. Työn tarkoituksena oli luoda mobiilisovellusalusta, jonka avulla pysytään toteuttamaan asiakkaille erinäisiä mobiiliaplikaatioita toiminnanohjaukseen liittyen. Alustalla pystytään kehittämään kaikille yleisimmille mobiililaitteille sovelluksia.

Asiakasprojektista riippuen voidaan käyttää jo sovellukseen tehtyjä ominaisuuksia. Ominaisuuksia voidaam myös lisätä tai soveltaa tarvittaessa. Asiakasprojekteissa pyritään aina pääsemään muokattuun järjestelmään, joka vastaa täysin asiakkaan käyttövaatimuksia.

5.1 Sovelluskehityksen työkalut

Sovelluksen kehittämiseen otettiin käyttöön erilaisia, sovelluskehitystä helpottavia työkaluja. Näiden avulla pystytään hallitsemaan sovelluksen kehityksen kulkua ja siihen tarvittavia työkaluja.

Sovelluksen kehittämiseen otettiin käyttöön kaksi eri pakentinhallintasovellusta, npm ja bower, sekä usein suoritettavia tehtäviä varten grunt. Sovelluksen versiohistoriaa hallitaan git-versionhallintatyökalulla sekä gitlab web -portaalilla, jonka kautta hallitaan sovelluskehitykseen liittyviä tehtäviä.

Npm:n kautta voidaan asentaa itse sovelluksenkehitykseen sekä testaamiseen tarvittavat sovellukset, kuten bower sekä grunt. Tämä helpottaa kehittäjän työtä, koska sovelluksen kehittämiseen tarvittavat työkalut kulkevat sen mukana.

Sovelluksen JavaScript- sekä CSS-kirjastoja hallinnoidaan bower-paketinhallinnan avulla, jolloin kehittäjillä on aina käytettävissä viimeisimmät versiot sovelluksessa käytettävistä kirjastoista. Bower myös hallitsee kirjastojen riippuvuudet, minkä ansiosta uuden kirjaston asentamisesta tulee suoraviivaisempaa.

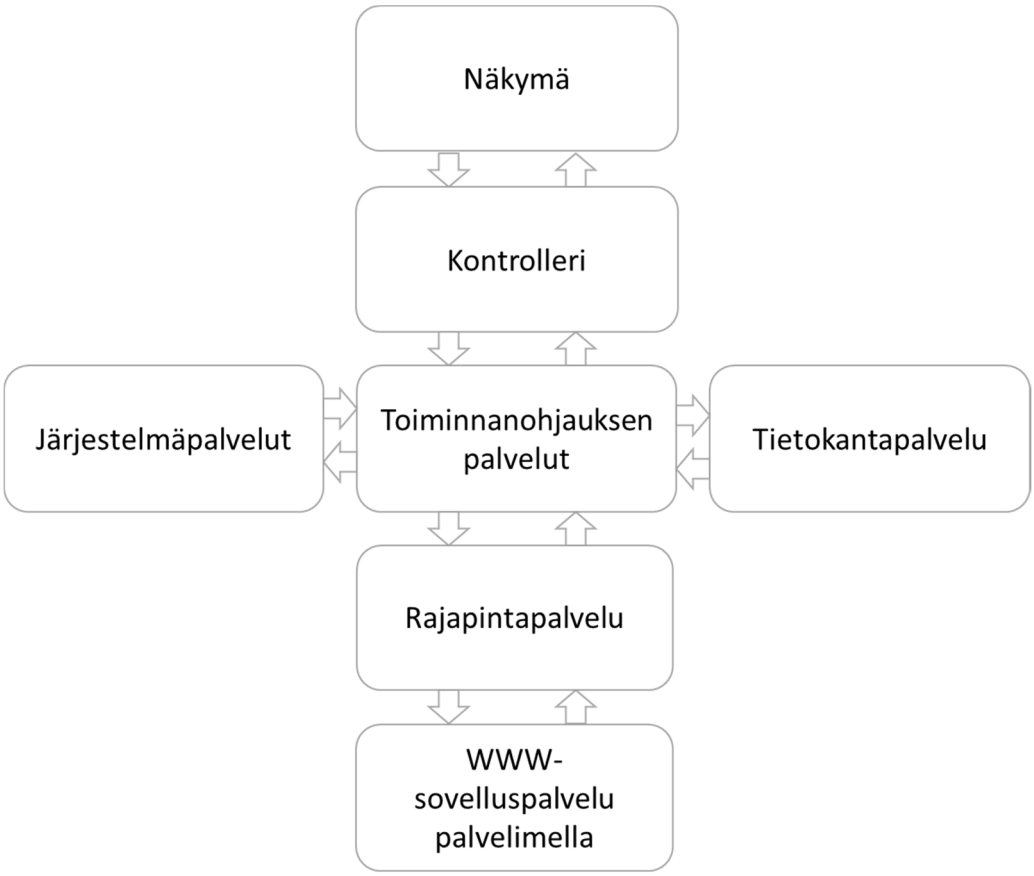
Gruntin avulla helpotetaan useasti suoritettavien tehtävien tekemistä, kuten sovelluksen kokoamista. Gruntin lisäosan avulla pystytään lisäämään sovelluksen päätiedostoon tarvittavat JavaScript-tiedostot oikeassa riippuvuusjärjestyksessä.

Git-versionhallinnan avulla pystytään kehittämään useampaa eri toimintoa samanaikaisesti eri kehittäjien kesken. Git:n avulla pystytään myös luomaan paikallisesti eri sovellushaaroja, jolloin kehittäjä voi helposti tallentaa tehdyt muutokset ja aloittaa uuden haaran kehittämisen ilman, että sovelluksessa olisi keskeneräisiä toimintoja. Git myös varmistaa sovelluksen varmuuskopioinnin toimimalla hajautettuna versionhallintana, koska jokaisella kehittäjällä on kopio sovellukseen tehdyistä muutoksista.

Git-versionhallintaa tukemaan otettiin käyttöön Gitlab-webportaali, joka tarjoaa visuaalisen näkymän versionhallinnasta. Gitlabin avulla pystyään myös hallitsemaan sovelluksen kehitystä kehityskohtien avulla. Gitlab myös mahdollistaa Git:n muutoksien linkittämisen kehityskohtiin, mikä helpottaa sovelluskehityksen seuraamista.

5.2 Arkkitehtuuri

Sovellus rakennettiin hyödyntämällä AngularJS:n palveluita, joiden avulla pystyttiin luomaan sovelluksesta modulaarinen kuvion 18 mukaisesti, jossa toiminnanohjauksenpalvelut esittävät eri toiminnanohjaukseen tarvittavaa osa-aluetta. Tämä tarkoittaa sitä, että tarvittaessa pystytään vaihtamaan kokonainen palvelu, ilman että sovelluksen muuhun rakenteeseen tarvitsee tehdä muutoksia. Asiakasprojekteihin yleensä joudutaan luomaan jokaista eri toimintoa varten uusi näkymä sekä kontrolleri, koska ne sisältävät aina toiminnolle tyypillisiä tehtäviä.



KUVIO 18. Sovelluksen arkkitehtuuri

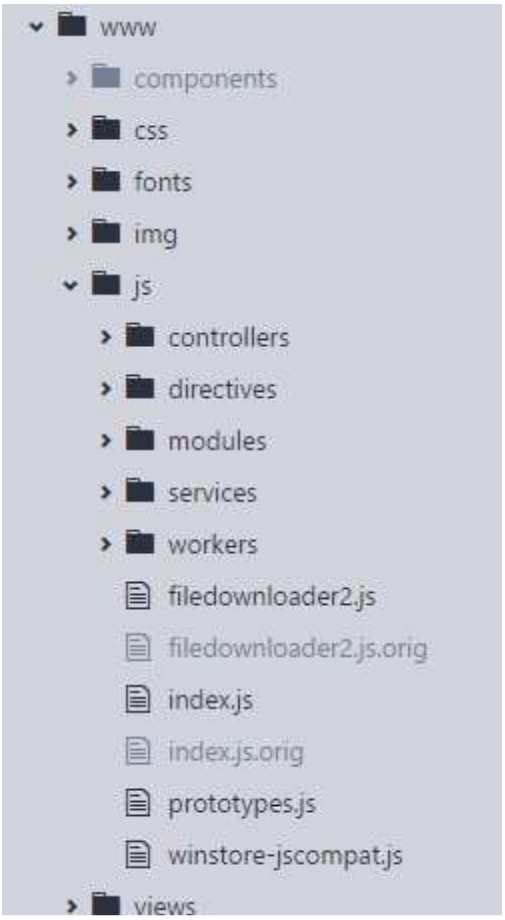
Toiminnanohjauksen-palvelu pyrkii aina hakemaan tietoa www-sovelluspalvelulta, jolloin sovelluksessa oleva tieto on ajantasaista. Tämä ei ole aina kuitenkaan mahdollista, minkä takia palvelu pyrkii tallentamaan tietokantapalvelun kautta haetun tiedon tietokantatauluihin, jolloin jos www-sovelluspalveluun ei ole yhteyttä, niin sovellusta pystytään käyttämään yhteydettömässä tilassa.

Toiminnanohjauspalvelu myös käyttää hyväkseen eri järjestelmäpalveluita, joiden avulla sillä on pääsy käyttöjärjestelmän toimintoihin sekä muihin yleiskäyttöisiin toimintoihin, kuten ilmoitusten luomiseen sekä otsikkorivin päivitykseen.

5.3 Hakemistorakenne

Sovelluksen hakemistorakenne rakennettiin niin, että se lajittelee sovelluksen eri osat omiin kansioihinsa. Sovelluksessa käytettävien kolmannen osapuolen kirjastot asennetaan bower-

paketinhallintasovelluksen avulla components-kansioon. Kaikki sovellukseen tehdyt css-kuvaustiedostot tallennetaan css-kansioon sekä sovelluksen näkymät tallennetaan views-kansioon. Sovellukseen kehitetetyt JavaScript-tiedostot tallennetaan js-kansioon, joka on jaettu useampiin alakansioihin. Eri alakansioihin tallennetaan AngularJS:n eri sovellusalueet, esimerkiksi kaikki sovelluksessa käytettävät kontrollerit tallennetaan controllers-kansioon sekä sovelluksen palvelut tallennetaan services-kansioon kuvion 19 tapaan.



KUVIO 19. Hakemistorakenne

5.4 Reititys

Reitityksellä tarkoitetaan sovelluksen sivujen linkittämistä toisiinsa osoitteilla. Sovelluksessa reititystä vastaa AngularJS:n kehittämä ngRoute-moduuli. Reitityksissä määritellään jokaiselle reitille oma näkymä sekä

kontrolleri, jotka injektoidaan sivulle elementtiin, jolla on määritetty ng-view direktiivi.

Sovelluksessa hyödynnettiin mahdollisuutta ladata näkymä dynaamisesti osoiterivillä annettavalla parametrilla. Tämä mahdollistaa kontrollerin uudelleenkäytön eri näkymillä ilman, että sovelluksen reitityksiä tarvitsee muuttaa. Jokainen reitti alkaa reittiä kuvaavalla nimellä, kuten kontrollerin nimellä, minkä jälkeen kerrotaan käytettävä näkymä.

Näkymän jälkeen reitille voidaan antaa vielä ylimääräisiä parametreja, joita voidaan käyttää kontrollerilla. Tämä mahdollistaa yksinkertaisen datan siirtämisen seuraavalle kontrollerille. Esimerkiksi varastoja selatessa annetaan varaston yksilöivä ID toisena parametrina, jolloin kontrolleri saa suoraan tiedon siitä, minkä varaston tietoja pitää esittää. Parametrit ovat saatavissa kontrollerilla routeParams-palvelulta ja annettavat parametrit määritetään reitiyksen yhteydessä.

Angular on luonut reititysten käyttöä varten $location-palvelun. Palvelu käyttää JavaScript window.location-rajapintaa ja antaa mahdollisuuden lukea sekä asettaa URL-osoitteita. Palvelun avulla voidaan sovelluksessa siirtyä sivulta toiselle joko niin, että edellinen sivu jää sovelluksen historiaan tai se poistetaan historiasta.

5.5 Tiedonsiirto

Sovelluksen tiedonsiirtoa varten rakennettiin ws-palvelu, joka tarjoaa sovelluksen eri osille yhtenäisen rajapinnan tiedonsiirtoa varten. Palvelulla on kaksi metodia, POST sekä GET, joiden avulla voidaan lähettää ja vastaanottaa tietoa www-sovelluspalvelulta. Tiedonsiirto on toteutettu sovelluksen ja www-sovelluspalvelun välillä SOAP-viesteillä. Ws-palvelu on rakennettu niin, että se ottaa sovellukselta vastaan JSON-formaatissa tietoa sekä palauttaa www-sovelluspalvelulta saadun tiedon JSON-formaatissa.

Yhteydetöntä tilaa varten sovelluspalvelun POST-metodille on mahdollista antaa parametreina tieto siitä tallennetaanko lähetetty tieto, jos tietoa ei

ole saatu siirrettyä. Tieto tallennetaan käyttämällä JavaScriptissä sekä Cordovassa tuettua localstoragea, joka pystyy säilyttämää tietoa koko sovelluksen elinkaaren yli.

5.6 Tiedon tallentaminen

Sovelluksesssa tietoa voidaan tallentaan puhelimen tietokantaan, kuten tiedot eri varastoista sekä niiden sisältämistä nimikkeistä. Tietokanta toimii SQL-kyselyillä. Tietokannan käyttämistä varten sovellukseen rakennettiin database-palvelu, jonka avulla pystytään hakemaan tietoa turvallisesti, koska palvelu pyrkii estämään SQL-injektiot tallentamalla syötetyn tiedon suodattimen kautta. Palvelun kautta pystytään tallentamaan suuria määriä tietoa samaan tauluun lähettämällä tieto taulukkomuodossa. Tämä mahdollistaa yhdellä transaktiolla useamman lisäyslauseen toteuttamisen, mikä nopeuttaa tiedon tallentamista huomattavasti. Database-palvelun avulla pystytään myös poistamaan tietoa tauluista.

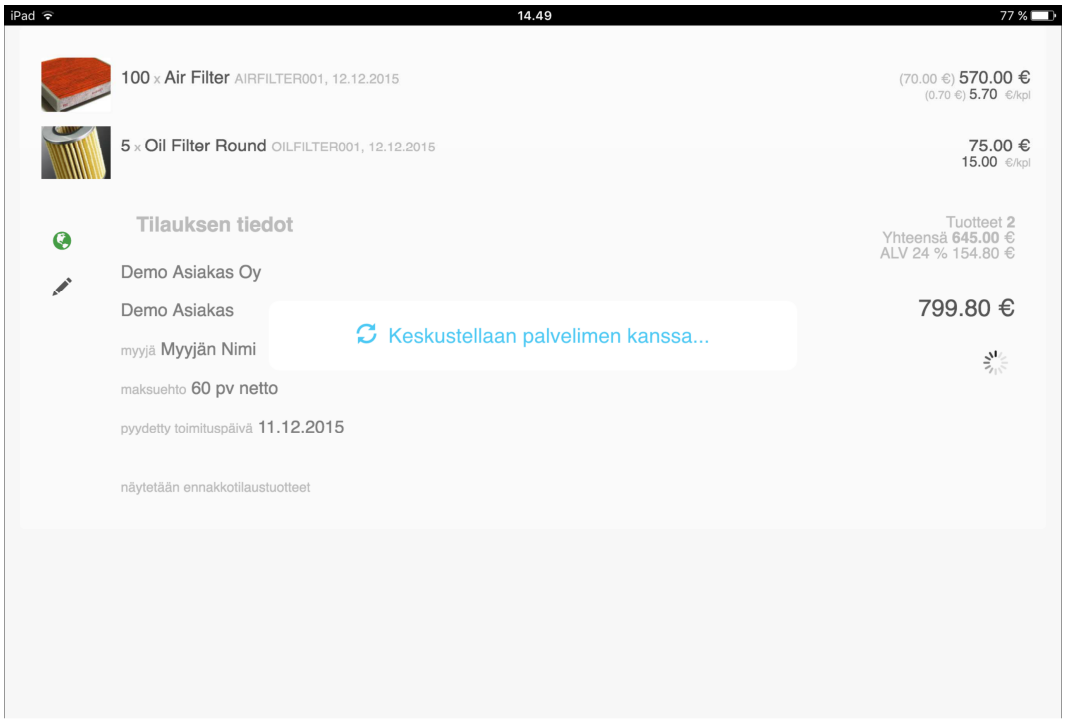
Laitteiden tietokannoilla on kuitenkin kokorajoituksia, mikä tarkoittaa sitä, että suuria tietomääriä, kuten kuvia, sinne ei voida tallentaa. Tätä varten luotiin erillinen tiedosto-palvelu, jonka tarkoitus on tallentaa tieto tiedostoihin laitteen tiedostojärjestelmään sekä tarjota tieto samassa muodossa takaisin sovelluspalvelulle. Palvelua hyödynnetään esimerkiksi kun sovelluksella otetaan kuvia. Kuvat tallennetaan tiedostostojärjestelmään base64-merkkijonona, koska ne lähetetään siinä muodossa myös www-sovelluspalvelulle.

5.7 Ulkoasu

Sovelluksen ulkoasu toteutettiin Twitter Bootstrapillä, koska sillä pystytään helposti toteuttamaan näkymiä pienille sekä suurille ruuduille. Isommilla näytöillä on mahdollista näyttää toisiinsa liittyvää tietoa vierekkäin, mikä vähentää käyttäjän tarvetta selata sivua alemmaksi. Pienillä näytöillä näytön leveys luo rajoituksia ruudulla näytettävälle tiedolle. Isolla näytöllä vierekkäin olevat kentät siirtyvät allekkaisiksi ja levenevät kokonäytön levyisiksi riveiksi.

Ulkoasu pyrkii olemaan neutraali väreiltään sekä korostamaan käyttäjälle olennaista tietoa. Tämä helpottaa sovelluksen käyttämistä, kun käyttäjän ei tarvitse etsiä tietoa näkymältä. Tärkeysluokassa alempana oleva tieto esittään näkymällä häivyttämällä se taustaan, jolloin se ei korostu kuvion 20 tapaan.

Sovelluksessa pystytään näyttämään käyttäjälle kuvion 20 tapaisia informatiivisia viestejä popuppina keskellä ruutua. Viestin tarkoitus on informoida käyttäjää tapahtumasta, mutta ei kuitenkaan estää käyttäjää käyttämästä sovellusta. Tähän päädyttiin, koska eri käyttöjärjestelmillä käyttöjärjestelmän tukemat popup-viestit ovat erillaisia sekä erinäköisiä. Tällöin saatiin yhtenevä ulkoasu näille viesteille.

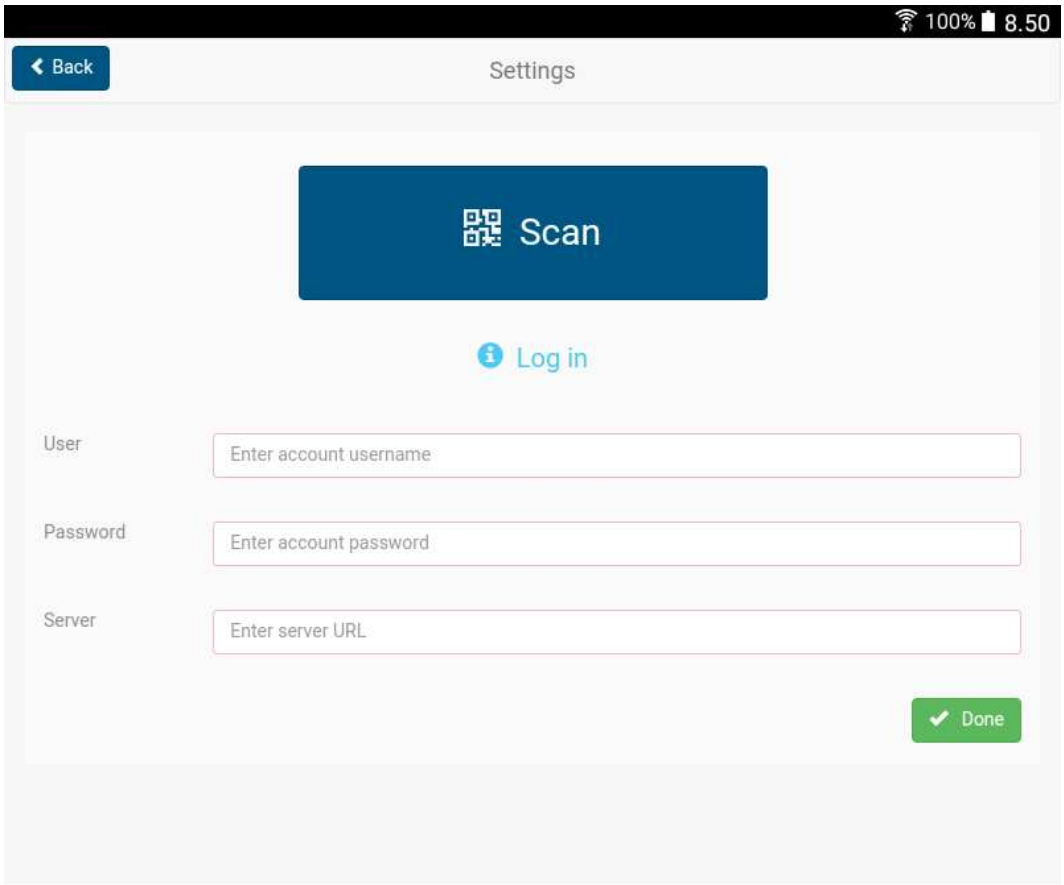


KUVIO 20. Esimerkki informatiivisesta viestistä, jota voidaan näyttää käyttäjälle

Viestit on toteutettu käyttämällä palvelua, jolle voidaan antaa parametreina viestin tyyppi, kesto sekä itse viesti. Viestityyppejä on olemassa neljä: varoitus, virhe, onnistuminen sekä informaatio. Viesti piilotetaan käyttäjältä, jos asetetaan uusi viesti, viesti, on näkynyt tarpeeksi kauan tai viesti poistetaan näkymästä.

5.8 Sovelluksen käyttöönotto

Sovelluksen käyttöönotto on tehty helpoksi asennuksen jälkeen. Ensimmäisellä käynnistyskerralla käyttäjä ohjataan suoraan asetussivulle (kuvio 21). Sivulla on mahdollista täyttää käyttäjänimi, salasana sekä www-sovelluspalveun osoite, jotka varmennetaan, kun kaikki asetukset on syötetty. Jos jokin asetus ei ole oikein tai yhteyttä www-sovelluspalveluun ei pystytä muodostamaan, niin asetus, josta virhe havaittiin, korostetaan käyttäjälle.



KUVIO 21. Sovelluksen asetukset

Asetukset on myös mahdollista asettaa skannaamalla QR-koodilla. Koodi joko lähetetään käyttäjälle tai se on saatavilla toiminnanohjausjärjestelmästä suoraan. QR-koodi sisältää vähintään sovelluksen asettamiseen tarvittavat asetukset.

5.9 Myyjämiesliittymä

Ensimmäisenä asiakasprojektina alustalle toteuttettiin myyjämiesliittymä. Liittymä on tarkoitettu käytettävän varastossa tapahtuvassa myynnissä, jossa asiakas tulee tutustumaan tuotteisiin myyjän vastolle, sekä etämyynnissä, jossa myyjä matkustaa asiakkaan tiloihin. Liittymän kautta myyjä pystyy tekemään tilauksen suoraan toiminnanohjausjärjestelmään, josta tilaus käsitellään eteenpäin.

Ennen tilauksen tekemistä myyjä syöttää asiakkaan nimen, yhteyshenkilön, myyjän, asiakkaan tilausnumeron, merkin sekä toivotun toimituspäivämäärän. Toiminnanohjausjärjestelmässä on mahdollista määrittää asiakkaalle oletusmyyjä, joka valitaan automaattisesti. Jos asiakkaalla on vain yksi yhteyshenkilö, niin liittymä valitsee yhteyshenkilön automaattisesti.

5.9.1 Nimikkeiden etsiminen ja lisääminen

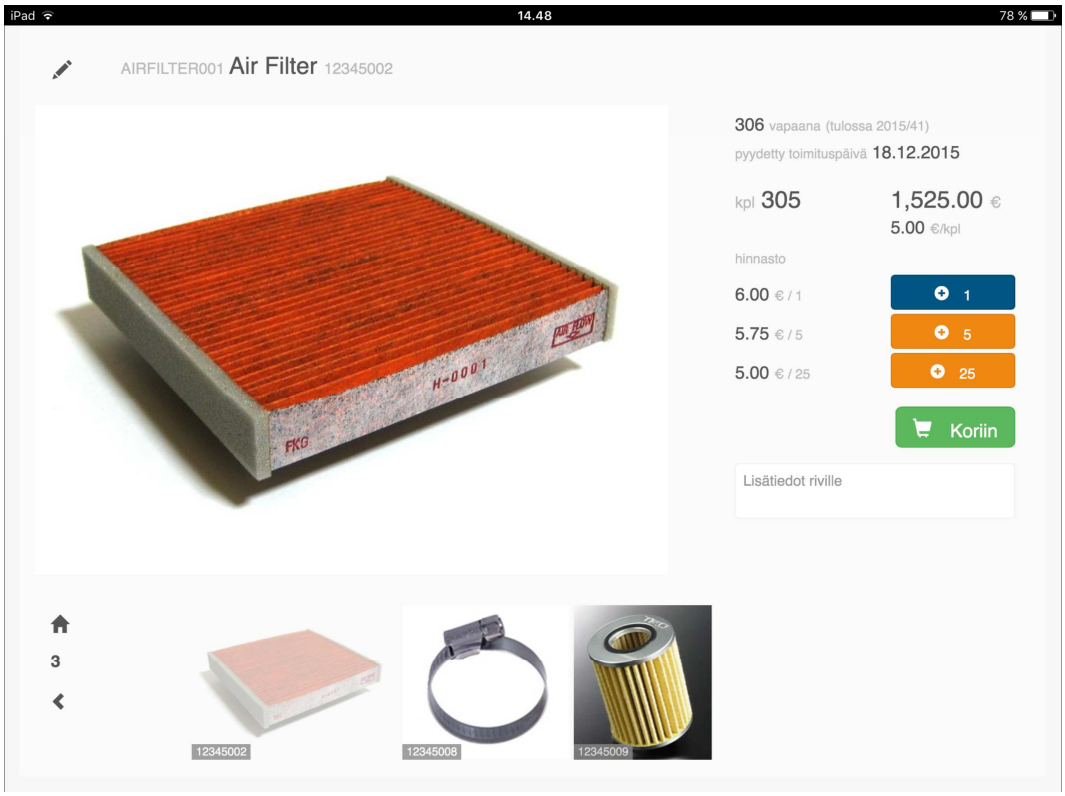
Liittymässä tuli olla mahdollista selailla sekä hakea nimikkeitä eri tavoilla. Lähimyynnissä myyjällä on käytössä viivakoodinlukija, jonka avulla myyjä pystyy lukemaan nimikkeen viivakoodin, jonka avulla sovellus etsii luetun nimikkeen tiedot. Etämyynnissä ei ole käytettävissä nimikkeen viivakoodia, jolloin tuotteita järjestelmästä löytyviä nimikkeitä tulee pystyä selailemaan sekä hakemaan. Etämyynnissä sovelluksen viivakoodikenttään voidaan myös kirjoittaa osa tuotteen nimestä, jolloin sovellus etsii tuotteiden nimistä syötettyjä sanoja ja antaa kättäjälle listan, josta voidaan valita haluttu tuote.

Jokaiselle tuotteelle on määritetty osasto, ja osasto voi kuulua yläosastoon. Tämän avulla käyttäjällä on käytettävissä puurakenne osastoista, jonka sisällä pystytään siirtymään sekä esittelemään samantyyppisiä nimikkeitä. Osastorakennetta hyödynnetään myös viivakoodilla luettaessa niin, että tuotteen tietojen alapuolelle tulee näkymään samassa osastossa olevia nimikkeitä. Näiden lisäksi nimikkeitä on mahdollist hakea nimellä, jossa käytetään avuksi ennakoivaa syöttöä,

eli sovellus hakee aktiivisesti nimikkeitä, jotka vastaavat käyttäjän syötettä. Tämä on nähtävissä kuvion 22 alareunassa.

Toiminnanohjausjärjestelmä laskee nimikkeelle aina asiakaskohtaisen hinnan kuvion 22 tapaan. Hinta koostuu nimikkeelle määritetystä hinnoitteluhinnasta sekä mahdollisista kampanjoista sekä asiakaskohtaisista alennuksista. Näiden lisäksi nimikkeelle voidaan määrittää hintaportaita, joiden mukaan ostaja saa alennusta. Hintaportaiden mukaan käyttöliittymään lisättiin painikkeet, joiden avulla myyjä pystyy lisäämään suoraan hintaportaaseen tarvittavan määrän. Tarvittaessa myyjä pystyy muuttamaan tilatun määrän kokonaishintaa tai yksikköhintaa.

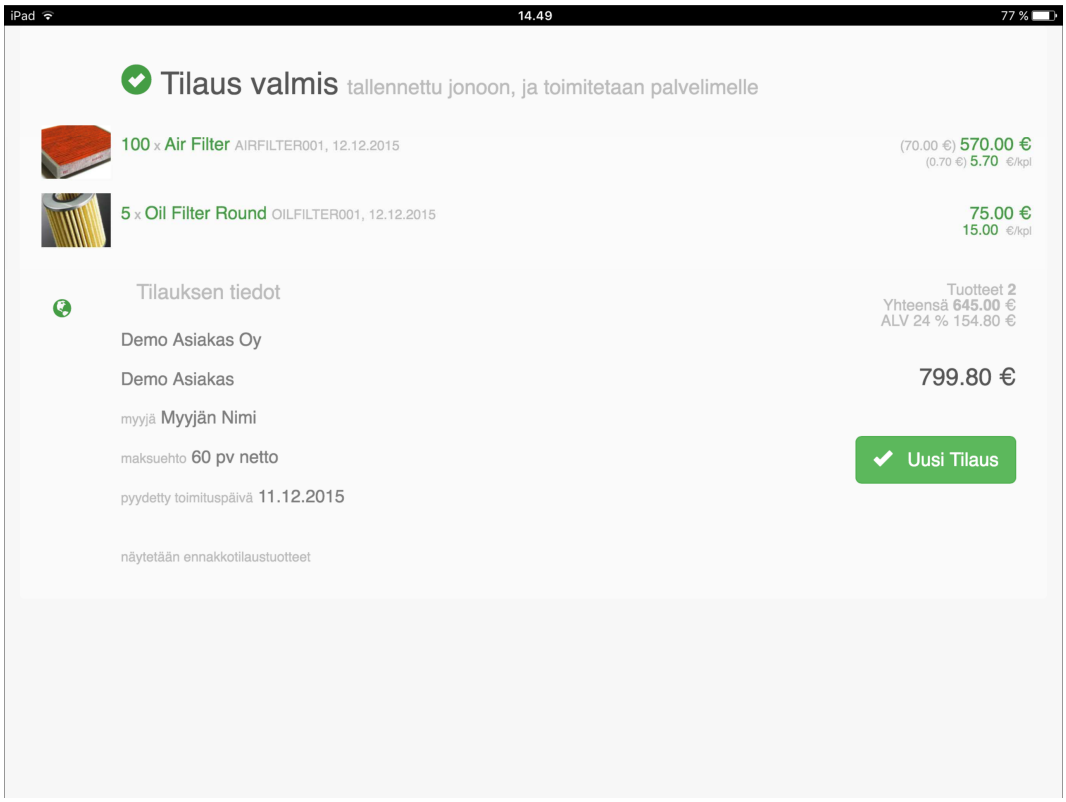
Sovellus kertoo käyttäjälle nimikkeen tämänhetkisen varastosaldon, tulossa olevan määrän ja sen, koska nimikettä on tulossa. Jos käyttäjä syöttää tilaukseen enemmän kuin nimikettä on saatavilla hankinta-ajan sisäpuolella, niin sovellus antaa tästä palautetta muuttamalla Koriin-napin oranssiksi. Myös napit, joiden avulla voidaan lisätä kappalemäärää, muuttuvat oransseiksi, jos sitä painamalla tilattu määrä ylittäisi saatavilla olevan määrän. Tämä on nähtävissä kuviossa 22.



KUVIO 22. Nimikkeen lisääminen ostoskoriin

5.9.2 Tilauksen viimeistely

Ostoskorissa asiakkalle näytetään tilauksen kokonaissumma sekä yhteenveto tilauksesta toimituspäivineen. Ostoskorista on mahdollista muokata sekä poistaa nimikkeitä tilaukselta. Kun tilaus on saatu täytettyä, se voidaan lähettää järjestelmälle käsiteltäväksi. Lähetyksen yhteydessä liittymä tarkistaa, että tilaus on oikein täytetty, ja antaa tarvittaessa palautteen. Tilauksen lähetyksen jälkeen liittymä siirtyy kuvion 23 tilaan, jossa tilausta ei enään voida muokata.

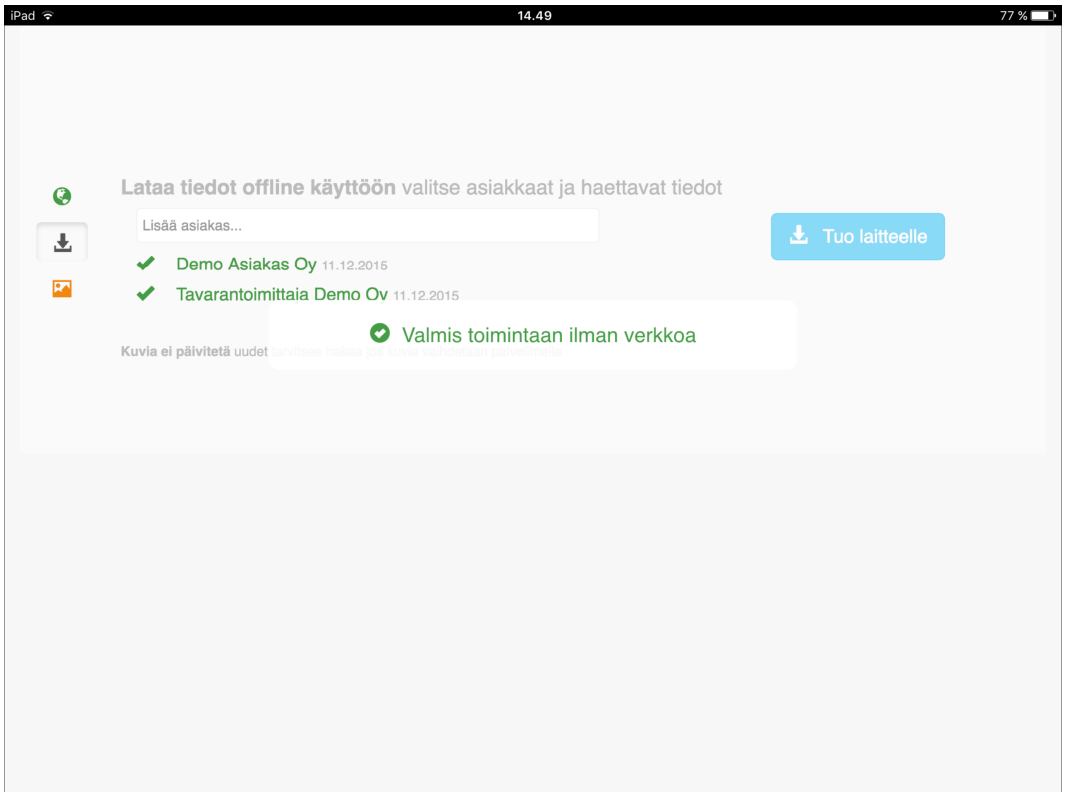


KUVIO 23. Tilauksen yhteenveto

5.9.3 Yhteydetön tila

Asiakkalla voi olla asiakaskohtainen hinnasto, jonka laskeminen on monimutkainen prosessi. Laskentaa ei siirretty sovellukseen, mikä tarkoittaa sitä, että sovelluksella täytyy olla yhteys www-sovelluspalveluun, jotta se pystyy näyttämään oikeita hintoja. Etämyyntinä tapahtuvassa myynnissä ei ole aina tarjolla internetyhteyttä, minkä takia sovellus ei pysty kommunikoimaan www-sovelluspalvelun kanssa.

Liittymään luotiin mahdollisuus hakea hinnastoja paikalliseen tietokantaan valitsemalla listaan asiakkaita kuvion 24 tapaan. Jos asiakkaan hinnastoa ei ole haettu laitteelle valmiiksi ennen yhteydettömään tilaan siirtymistä, niin liittymään on mahdollista määrittää jokin tunnettu asiakas, jonka hinnastoa käytetään aina kun valitulle asiakkaalle ei löydy hintaa. Liittymä myös tallentaa aina hintoja yhteydellisessä tilassa, jolloin se pystyy tarjoamaan viimeisimmän hinnan.



KUVIO 24. Asiakkaiden hinnastojen lataaminen yhteydettömään tilaan

6 YHTEENVETO

Opinnäytetyön tavoitteena oli luoda pohja mobiilisovellukselle, jonka avulla pystytään toteuttamaan erilaisia asiaksprojekteja yrityksen toiminnanohjaukseen liittyen. Tätä varten tutkittiin erilaisia ratkaisuvaihtoehtoja, miten saadaan rakennettua sovellus mahdollisimman laajalle pohjalle.

Sovelluksen toimintamalliksi valikoitui hybridisovellusalusta Cordova, koska se tarjosi pohjan eri käyttöjärjestelmille. Cordova tarjoaa sovelluskehitysalustan markkinoiden suurimmille käyttöjärjestelmille, jolloin pystytään kattamaan mahdollisimman laaja käyttäjäkunta. Sovelluksen logiikan sekä rakenteen suorittajaksi valikoitui AngularJS, koska sen yhteisö on erittäin laaja ja sille on saatavilla paljon ilmaisia kolmannen osapuolen kehittämiä liitännäisiä. Sovelluksen ulkoasu on toteuttettu Twitter Bootstrapillä, koska se tarjoaa responsiivisen ulkoasukehityksen. Tällöin saadaan sovelluksen näkymä sopimaan monelle eri näyttölle ilman, että jokaiselle näytölle tarvitsisi kehittää oma näkymä.

Opinnäytetyön aikana sovelluksen ydin saatiin tehtyä ja sen päälle toteutettua yksi kokonainen asiakasprojekti, joka on asiakkaalla päivittäisessä käytössä. Tulevaisuudessa sovelluksen ydintä ja sen ympärille rakennettavaa infrakstruktuuria tullaan vielä jatkokehittämään, jotta siitä saadaan mahdollisimman monipuolinen sekä kattava eri tarpeisiin. Sovelluksen ytimen rakennetta pyritään yksinkertaistamaan sekä dokumentoimaan, jolloin uusien kehittäjien on helpompi tutustua alustan tarjoamiin mahdollisuuksiin sekä laajentaa sitä tarvittaessa.

Haasteita sovelluksen kehityksessä oli sovelluksen saaminen monipuoliseksi sekä eri sovelluksen käyttäminen erilaisilla laitteilla, jotta sillä pystyttäisiin tekemään muutakin kuin yhtä toimintoa. Tämän lisäksi yhdeksi suurimmaksi haasteeksi muodostui yhteydettömän tilan toteuttaminen, koska toiminnanohjauksessa käytettävä data on liian suuri ladattavaksi mobiililaitteelle, jolloin päädyttiin ratkaisuun vain ladata tarvittava tieto. Tämä tarkoittaa sitä, että jokaista projektia varten täytyy

miettiä mitä tietoa tullaan käyttämään ja mitä pitää ladata etukäteen laitteelle, jotta sitä voidaan käyttää yhteydettömässä tilassa. Tämän lisäksi tiedon synkronointi toiminnanohjausjärjestelmän kanssa tuotti vaikeuksia, koska datan vaihteluvuutta on vaikea ennustaa eikä siihen ollut valmista ratkaisua rakennettu.

Sovelluksen käyttämistä kirjastoista julkaistaan jatkuvasti uusia versioita. Uudemmat versiot tarjoavat paremman tuen uusille, juuri julkaistuille laitteille, mutta saattavat rikkoa vanhempien laitteiden tukia. Sovelluksen kirjastot ovat riippuvia toisistansa, minkä takia kirjastot, joita ei enää kehitetä, tulevat elinkaarensa päähän, kun kehitys etenee. Tämä tarkoittaa sitä, että kehitettyä mobiilisovellusalustaa pitää päivittää ja muokata jatkuvasti vastaamaan huomisen tarpeita.

LÄHTEET

Apple. 2015a. App Store Review Guidelines [viitattu 15.3.2015]. Saatavissa: https://developer.apple.com/app-store/review/guidelines/

Apple. 2015b. Apple Developer. UI Elements - Bars [viitattu 10.9.2015]. Saatavissa: https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Bars.html#//apple_ref/doc/uid/TP40006556-CH12-SW3.

Apple. 2015. Choosing iOS Developer Program [viitattu 15.3.2015]. Saatavissa: https://developer.apple.com/programs/start/ios/.

Ariel 2015. App Stores Growth Accelerates in 2014 [viitattu 16.4.2015] Saatavissa: http://blog.appfigures.com/app-stores-growth-accelerates-in-2014/.

Bootstrap. 2015. Components [viitattu 15.10.2015]. Saatavissa: http://getbootstrap.com/components/.

Bootstrap. 2015. Get Bootstrap! [viitattu 9 7, 2015]. Saatavissa: http://getbootstrap.com/.

Burns, J. 2015. To Use or Not to Use a Database? That is the Question [viitattu 5.10.2015]. Saatavissa: http://www.htmlgoodies.com/primers/database/article.php/3478121.

CDS. 2015. Touchscreen market growing 10 times faster than other displays [viitattu 15.5.2015]. Saatavissa: http://crystal-display.com/touchscreens/touchscreen-market-growing-10-times-faster-than-other-displays/.

Chartier, D. 2015. iPhone OS gets new name, video calling [viitattu 10.3.2015]. Saatavissa: http://www.macworld.com/article/1151812/iphone_os_4_wwdc.html.

Clifford, C. 2015. By 2017, the App Market Will Be a $77 Billion Industry (Infographic) [viitattu 8.3.2015]. Saatavissa: http://www.entrepreneur.com/article/236832.

Cordova. 2015. Apache Cordova [viitattu 15.5.2015]. Saatavissa: https://cordova.apache.org/.

Cowart, J. 2013. Telerik. Demystifying Apache Cordova and PhoneGap [viitattu 7.12.2015]. Saatavissa: http://www.telerik.com/blogs/demystifying-apache-cordova-and-phonegap.

Creative Workline 2015 [viitattu 19.2.2015]. Saatavissa: http://www.creativeworkline.com/2015/02/mobile-design-trends-guidelines-2015/.

Eason, J. 2015. Android 5.1 Lollipop SDK [viitattu 3.8.2015]. Saatavissa: http://android-developers.blogspot.fi/2015/03/android-51-lollipop-sdk.html.

Ferguson, D. 2004. Exclusive .NET Developer's Journal "Indigo" Interview with Microsoft's Don Box [viitattu 8.11.2015]. Saatavissa: http://dotnet.sys-con.com/node/45908.

Google. 2015a. Android 5.0 Compatibility Definition [viitattu 10.92015]. Saatavissa: https://static.googleusercontent.com/media/source.android.com/en//compatibility/5.0/android-5.0-cdd.pdf. 35.

Google. 2015b. AngularJS by Google [viitattu 10.9.2015]. Saatavissa: https://angularjs.org/.

Google. 2015c. AngularJS Directives [viitattu 10.9.2015]. Saatavissa: https://docs.angularjs.org/guide/directive.

Google. 2015d. Google Cloud Messaging for Android [viitattu 1.4.2015]. Saatavissa: https://developer.android.com/google/gcm/index.html.

Google. 2015e. Launch Checklist [viitattu 15.3.2015]. Saatavissa: http://developer.android.com/distribute/tools/launch-checklist.html.

Google. 2015f. Services [viitattu 7.9.2015]. Saatavissa: https://docs.angularjs.org/guide/services.

Google. 2015g. Tutorial: Routing & Multiple Views [viitattu 10.9.2015]. Saatavissa: https://docs.angularjs.org/tutorial/step_07.

Google. 2015h. Understanding Controllers [viitattu 6.11.2015]. Saatavissa: https://docs.angularjs.org/guide/controller.

Google. 2015i. What is a Module? [viitattu 6.9.2015]. Saatavissa: https://docs.angularjs.org/guide/module.

Hollister, S. 2015. Microsoft prepping Windows Phone 7 for an October 21st launch? (update: US on Nov. 8) [viitattu 24.5.2015]. Saatavissa: http://www.engadget.com/2010/09/26/microsoft-prepping-windows-phone-7-for-an-october-21st-launch/.

IDC. 2015. Smartphone OS Market Share, Q4 2014 [viitattu 8.3.2015]. Saatavissa: http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

JL-Soft. 2015. JL-Soft [viitattu 10.7.2015]. Saatavissa: http://jlsoft.fi/.

Microsoft. 2015a. Account types, locations, and fees [viitattu 15.3.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/windows/apps/jj863494.aspx.

Microsoft. 2014. Announcing the .NET Framework 4.5.2 [viitattu 15.3.2015]. Saatavissa: http://blogs.msdn.com/b/dotnet/archive/2014/05/05/announcing-the-net-framework-4-5-2-release.aspx.

Microsoft. 2015b. How to navigate using the back stack for Windows Phone 8 [viitattu 10.9.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/windows/apps/hh394012(v=vs.105).aspx.

Microsoft. 2015c. Understanding app and in-app product submission [viitattu 15.3.2015]. Saatavissa: https://msdn.microsoft.com/library/windows/apps/jj206729.aspx.

Miles, S. 2015. Windows Phone 8: New hardware specs offer a new start [viitattu 15.3.2015] Saatavissa: http://www.pocket-lint.com/news/115972-windows-phone-8-hardware-specs.

Morrill, D. 2008. Announcing the Android 1.0 SDK, release 1 [viitattu 23.9.20015]. Saatavissa: http://android-developers.blogspot.fi/2008/09/announcing-android-10-sdk-release-1.html.

Korf, M ja Oksman, E. 2015. Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options [viitattu 8.11.2015]. Saatavissa: https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options.

Pushwoosh. 2015. Pushwoosh [viitattu 9.12.2015]. Saatavissa: https://www.pushwoosh.com/.

Salesforce. 2015. Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options [viitattu 1.12.2015]. Saatavissa: https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options.

Shiny Development. 2015. Average App Store Review Times [viitattu 10.3.2015]. Saatavissa: http://appreviewtimes.com/

Singh, A. 2013. A Brief History of Mac OS X [viitattu 15.10.2015]. Saatavissa: http://osxbook.com/book/bonus/ancient/whatismacosx/history.html

Staff, V. 2015. iOS: A visual history [viitattu 10.3.2015]. Saatavissa: http://www.theverge.com/2011/12/13/2612736/ios-history-iphone-ipad.

Swider, M. 2015. techradar. iOS 9 release date, features and news [viitattu 1.12.2015]. Saatavissa: http://www.techradar.com/news/software/operating-systems/ios-9-what-we-want-to-see-1253732.

Anthony, T. 2015. Finger-Friendly Design: Ideal Mobile Touchscreen Target Sizes [viitattu 15.5.2015]. Saatavissa: http://www.smashingmagazine.com/2012/02/21/finger-friendly-design-ideal-mobile-touchscreen-target-sizes/.

Telerik. 2015. What is a Hybrid Mobile App? [viitattu 5.4.2015]. Saatavissa: http://blogs.telerik.com/appbuilder/posts/12-06-14/what-is-a-hybrid-mobile-app-.

T-Mobile. 2015. T-Mobile has sold 1 million G1 Android phones [viitattu 8.3.2015]. Saatavissa: http://www.cnet.com/news/t-mobile-has-sold-1-million-g1-android-phones/.

W3. 2010. Web SQL Database [viitattu 10.5.2015]. Saatavissa: http://www.w3.org/TR/webdatabase/.

W3C. 2009. W3C. XML Protocol Working Group [viitattu 16.4.2015]. Saatavissa: http://www.w3.org/2000/xp/Group/.

W3Schools. 2015. XML Soap [viitattu 30.11.2015]. Saatavissa: http://www.w3schools.com/webservices/ws_soap_example.asp.

Warren, T. 2015. Microsoft begins sharing Windows Phone 8.1 with developers [viitattu 16.4.2015]. Saatavissa: http://www.theverge.com/2014/2/10/5399264/microsoft-begins-sharing-windows-phone-8-1-with-developers/in/4584231.

Whitney, L. 2012. Offline Capabilities: Native Mobile Apps vs. Mobile Web Apps [viitattu 15.5.2015]. Saatavissa: http://www.sitepoint.com/offline-capabilities-native-mobile-apps-vs-mobile-web-apps/.

Wikipedia. 2015. .NET Framework [viitattu 15.3.2015]. Saatavissa: http://en.wikipedia.org/wiki/.NET_Framework.

Wikipedia. 2015. Mobile database [viitattu 15.3.2015]. Saatavissa: http://en.wikipedia.org/wiki/Mobile_database.

Voo, B. 2015. HONGKIAT. Mobile Device Storage: How Much Do You (Really) Need? [viitattu 21.11.2015]. Saatavissa: http://www.hongkiat.com/blog/managing-mobile-storage-space/.

Xamarin. 2015. Xamarin [viitattu 2.12.2015]. Saatavissa: https://xamarin.com/.