Ilpo Andström

Hybridimobiilisovelluksen kehitys

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Ohjelmistotekniikka

Insinöörityö

2.5.2016

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Ilpo Andström Mobiilisovelluskehitys hybridiapplikaationa 45 sivua 2.5.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Juha Kämäri

Insinöörityössä tutustuttiin yleisellä tasolla mobiilisovelluskehitykseen. Selvitettiin, minkälaisia tekniikoita mobiilikehitykseen on käytettävissä ja mikä sopisi insinöörityön toteutukseen. Toteutustavaksi valittiin hybridikehitys ja sovelluskehykseksi Ionic Framework.

Raportti esittelee aluksi yleisimmät tavat kehittää mobiilisovellusta. Yleisen esittelyn jälkeen tutustutaan tarkemmin työhön valittuun tekniikkaan, eli hybridikehitykseen. Hybridikehityksestä siirrytään esittelemään työssä käytettyä Ionic Frameworkia. Lopuksi raportissa käsitellään kehityksen tulosta.

Insinöörityöprojektissa kehitetyn sovelluksen päätehtävä oli tuoda mobiilikäyttöön työkalu, joka helpottaa leivontareseptien skaalausta oikeaan kokoon. Sovelluksella piti pystyä laskemaan resepti haluttuun kokoon ja tallentamaan reseptejä. Lisäominaisuutena sovelluksen piti pystyä vastaanottamaan uutisia ulkopuoliselta palvelimelta.

Tuloksena syntyi määritelmän mukainen sovellus, joka kääntyy Android- ja iOS-alustalle. Sovelluksen laskurityökalu vaatii vielä testausta suljetulla käyttäjäjoukolla. Ulkopuoliselta palvelimelta tiedon vastaanottaminen toimii suunnitelman mukaisesti.

Avainsanat	HTML, CSS, JavaScript, Ionic, Hybridimobiilisovellus

Author(s) Title Number of Pages Date	Ilpo Andström Development of hybrid mobile applications 45 pages 2 may 2016
Degree	Bachelor of Engineering
Degree Programme	Information technology
Specialisation option	Software engineering
Instructor(s)	Juha-Pekka Kämäri, Senior Lecturer

The purpose of this thesis was to research mobile application development. At the beginning technologies available for a mobile development were examined and which one would fit the best for the project. Hybrid development and Ionic Framework met these criteria.

The most common techniques used to develop for mobile platforms are introduced in the beginning of the thesis followed by a more detailed explanation of the hybrid development and the Ionic Framework. Ionic tools and development with Ionic are also introduced. Lastly the study showcases a hybrid mobile application which is the practical part of the study.

The main functionality of the application is providing a baking recipe scaler for mobile platforms. The application has to be able to calculate the recipe for the requested size and save recipes saved to the favorites. A minor requirement for it was it to be capable to receive data from an external server.

As a result, the hybrid mobile application meeting the requirements was created. The application was created for Android and iOS platforms. The calculating tool of the application needs further manual testing. The application is capable of receiving data from a server.

Keywords	HTML, CSS, JavaScript, Ionic, Hybrid mobile application

Sisällys

Lyhenteet

API	Application Programming Interface. Ohjelmoinnissa käytettävä rajapinta.
DOM	Document Object Model. Dokumenttioliomalli, Kuvaa HTML-dokumentin puurakenteisena oliona.
CLI	Command-line Interface. Komentorivikäyttöliittymä.
CSS	Cascading Style Sheets. Porrastettu tyyliarkki.
HTML	HyperText MarkupLanguage. Hypertekstin merkintäkieli.
IAP	In-App Purchase. Sovelluksen sisällä tapahtuva osto.
IDE	integrated development environment. Kehitysympäristö, joka pitää sisällään koodieditorin, käännöstyökalut ja virheen korjaus työkalut.
iOS	Apple puhelinten käyttöjärjestelmä
JSON	JavaScript Object Notation. Tiedontallennus muoto, missä tieto on tallennettu ominaisuus-arvo pareina.
NPM	NodeJS Package Manager. NodeJS paketinhallintatyökalu
MVC	Model-View-Controller. Malli-Näkymä-Ohjain suunnittelumalli
MVP	Minimum-Viable-Product, eli mahdollisimman yksinkertainen, mutta toimiva sovellus tiettyyn tarkoitukseen.
MVVM	Model-View-Model View. Malli-Näkymä-Malli Näkymä suunnittelumalli.
REST	REpresentational State Transfer. Arkkitehtuurimalli ohjelmointirajapinnoille.
SDK	Software development kit. Sisältää ajurit ja kehitystyökalut tietylle alustalle.

URL	Uniform Resource Locator. Merkkijono, joka osoittaa WWW-sivuun.
XML	Extensible Markup Language. Tiedontallennusformaatti mikä on ihmisen ja koneen luettavassa muodossa.

1 Johdanto

Opinnäytetyön aihe rakentui osittain mielenkiinnosta mobiilisovelluskehitystä kohtaan ja osittain oikeasta tarpeesta. Tekila's Bakery -nimistä blogia pyörittävä bloggaaja halusi tuoda jotain uutta perinteiseen verkkobloggaamiseen. Mobiilisovellusta suunnitellessa huomasimme nopeasti, että pelkkä blogin siirtäminen mobiilialustalle olisi turha työmaa, koska nykyisten responsiivisten nettisivutekniikoiden avulla blogi toimii jo tarpeeksi sulavasti mobiilissa.

Päätimme jättää sovelluksessa blogin pienempään rooliin ja keskittyä tuomaan oikeasti jotain uutta blogin lukijoille. Sovelluksen toiminnallisuus painottuu kakun ja täytteiden ainemäärien selvittämiseen käyttäjän syöttämien määritelmien mukaan. Hybridimobiilisovellus tuntui sopivalta tekniikalta sovelluksen toteuttamiseksi.

Opinnäytetyössä tutustutaan mobiilisovelluskehitykseen ja vertaillaan erilaisia toteutustapoja, avataan taustoja, kuinka päädyttiin hybridisovellukseen eikä natiiveihin toteutuksiin, tutkitaan tarkemmin hybridimobiilisovelluskehitystä ja erityisesti tarkastellaan Ionic-sovelluskehystä, miten Ionic saa html5-websovelluksen pyörimään natiivin sovelluksen kaltaiseksi eri mobiilialustoilla. Lopuksi tehdään yhteenveto, kuinka suunnitelmissa pysyttiin ja saatiinko kaikki toteutettua.

2 Tavoitteet

Sovelluksen pitää toimia yleisimmillä mobiililaitteilla, käytännössä IOS- ja Android-alustoilla. Ainemäärälaskureiden pitää toimia myös ilman internetyhteyttä, joten piti kehittää jokin tapa tallettaa sovellukseen paikallisesti kaikkien kakkupohjien ja täytteiden ainekset. Vaikka pääpaino siirrettiin määrittelyvaiheessa pois blogin sisällöstä, niin silti sovelluksella pitää pystyä lukemaan ainakin uusimpia blogikirjoituksia, jos internetyhteys on käytössä.

Sovelluksen tuottamia reseptejä pitää voida jollain tapaa tallentaa. Reseptin paikallinen tallennus rajattiin toimimaan vain koko kakun ainemäärät laskevaan työkaluun. Tässä tapauksessa koko kakku käsittää pohjan ja täytteen. Paikalliseen tietojen tallennukseen valittiin SQLite -tietokanta, joka löytyy molemmista käyttöjärjestelmistä, joita sovelluksen

pitää tukea. Paikallisen tallentamisen lisäksi reseptit pitää saada tulostettavaan muotoon.

Ulkoasullisesti ohjelmasta piti saada selkeä, että laskureiden käyttö olisi helppoa. Laskureiden näyttämät tulossivut pitää noudattaa yleistä reseptien esitystapaa.

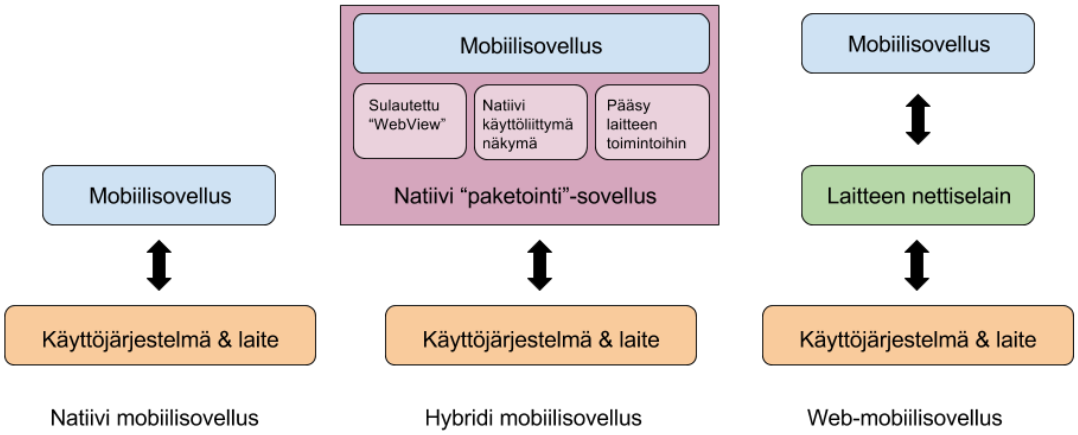
3 Mobiilisovelluskehitys yleisesti

Langattomien internetyhteyksien yleistyminen ja sitä kautta internetin käyttö mobiililaitteilla on lisääntynyt vuosi vuodelta. Mobiililaitteet ovat luoneet tarpeen tehdä verkkosivuista mobiililaitteilla toimivia versioita, eli niin sanotusti responsiivisia. Yleistyneet toimintatavat mobiililaitteilla voi olla hankala siirtää perinteiseen verkkosivuun, vaikka verkkosivun responsiivinen toteutus olisi onnistuneesti tehty.

Mobiilisovellukset voivat olla yksi ratkaisu välttää ylimääräinen osoitteiden kirjoittelu ja hankalien responsiivisten käyttöliittymien käyttäminen mobiililaitteilla. Sovelluksen voi suunnitella siten, että siihen on valittu verkkosivulta keskeisimmät toiminnot ja monesti sellaiset toiminnot, joita tarvitaan jossain muualla kuin pöytäkoneen ääressä.

Perinteisesti mobiilisovelluksen kehittäminen on ollut kallis ja pitkä prosessi. On pitänyt miettiä, mitä alustoja sovelluksen tarvitsee tukea. Missä sovellus jaetaan asiakkaiden saataville ja miettiä, onko yrityksellä tietotaito toteuttaa kaikki tarvittavat alustariippuvaiset asiat? Edellä mainittuja asioita pohtii Carl Vuorinen blogikirjoituksessaan. [1.]

Kuvaan 1 on havainnollistettu perusarkkitehtuuri natiivi-, hybridi- ja web-sovellukselle.



Kuva 1. Korkeantason arkkitehtuuri mobiilisovelluksille [4.]

3.1 Natiivi mobiilisovelluskehitys

Natiivilla mobiilisovelluskehityksellä tarkoitetaan prosessia, missä jokaiselle alustalle ohjelmoidaan alustakohtainen sovellus. Taulukossa 1 on esitelty kolmen yleisimmän alustan käyttämät natiivit ohjelmointikielet. Jos sovelluksen haluaa kaikille yleisille alustoille, pitää tiimissä olla jokaisen alustan asiantuntija.

Taulukko 1. Mobiilialustojen ohjelmointikielet [7, s 9-10.]

Alusta	Kieli	Kehitystyökalut
Android	Java	Android Studio
IOS	Objective-C / Swift	Xcode
Windows phone	C# / Visual Basic	Visual Studio

Seuraavaan listaan on koottu natiivin sovelluksen hyviä puolia:

- Helppo jakaa eri alustojen sovelluskaupassa.
- Sovellus voi olla maksullinen ja siihen voi laittaa IAP:eja ilman erillisiä rajapintoja tai verkkokauppatoimintoja.
- Natiivit sovellukset ovat suorituskykyisiä, varsinkin grafiikkaa sisältävissä sovelluksissa, kuten peleissä.
- Sovelluksella on pääsy suoraan laitteen toimintoihin ja rajapintoihin. [1; 2, s 4.]

Listan natiivin kehityksen heikkouksista:

- Jokainen alusta vaatii oman ohjelmointikielen ja tietämystä alustan API:sta.
- Samaa koodia ei voi käyttää muissa alustoissa, monin kertainen työmäärä suhteessa alustojen määrään.
- Vaatii paljon työtä ylläpitää montaa alustaa, mikä on kallista. [2, s 4-5.]

3.2 Web-sovelluskehitys

Mobiilialustojen internetselaimet on kehittynyt jo siihen pisteeseen, että nykypäivän web-sovelluksia voi kehittää mobiilikäyttöön tarkoitetuksi, mutta säilyttää myös pöytäkoneversion samasta sivusta. Carl Vuorinen näkee web-sovellus kehityksessä kustannustehokkaan vaihtoehdon ja samalla hyvin ylläpidettävän, koska kaikki sisältö tulee palvelimelta eikä tarvitse lokaalia sovelluspäivitystä. Nykyään web-sovelluksiin voidaan ottaa käyttöön jo jotain laitteen sisäisiä toimintoja. [1.]

Jeremy Wilken listaa osittain samoja web-sovelluksen vahvoja puolia kuin Vuorinen omassa tekstissään. Wilkenin lista web-sovelluksen vahvuuksista:

- Helppo ylläpitää, koska kaikki tieto tulee palvelimelta.

- Ei vaadi erillistä asennusta.

- Toimii kaikilla laitteilla, eikä ole sidottu eri alustojen sääntöihin tai ehtoihin. [2, s 6.]

Web-sovellus pyörii laitteen selaimessa, mikä aiheuttaa suurimman osan web-sovelluksen heikkouksista. Vuorisen tekstissä sivutaan osaa Wilkenin listaamista heikkouksista:

- Ei ole pääsyä kaikkiin natiiveihin ominaisuuksiin.

- Vaatii käyttäjältä osoitteen kirjoittamisen päästäkseen sovellukseen.

- Vaikea luoda toimiva käyttöliittymä yhtä aikaa mobiiliin ja pöytäkoneelle.

- Yleisellä tasolla mobiili internetin selaaminen vähenee ja applikaatioiden käyttö kasvaa. [1; 2, s 6.]

3.3 Hybridimobiilisovelluskehitys

Hybridimobiilisovellus on käsitteenä tarkoittaa web-tekniikoilla tehtyä sovellusta, joka ajetaan laitteissa erikseen asennettavana sovelluksena. Käytännössä tarvitaan mobiilisovellus, joka luo alustalle sopivan WebView-näkymän, joka on kuin sovelluksen sisään-rakennettu internetselain. Selain näkymä on ”full screen” -moodissa, eli ei näytetä osoitepalkkia, takaisin-nappia eikä muita selaimen toimintoja. Kuvassa 1 keskellä esitellään perusrakenne hybridimobiilisovellukselle. [1.]

Hybridimobiilisovelluksien tekemisessä käytetään Apache Cordovan ja Appcelerator Titaniumin kaltaisia sovelluskehyksiä, jotka hoitavat sovelluksen paketoinnin ja WebViewiä pyörittävän sovelluksen luomisen. Edellä mainittujen työkalujen tehtävä on myös hoitaa kommunikointi WebViewissä pyörivän web-sovelluksen ja laitteen natiivien toiminnollisuuksien välillä, joita ovat esimerkiksi kamera ja yhteystiedot. Apache Cordova -sovelluksen arkkitehtuuriesimerkki näkyy kuvassa 5.[2, s 6.]

Vuorinen ja Wilken listaavat teksteissään hybridisovellusten hyviä puolia:

- Samalla koodilla saa kehitettyä monelle alustalle.
- Sovelluksen jakaminen ja asentaminen ovat helppoa, koska sovellukset voi laittaa laitealustojen sovelluskauppoihin.
- Lähes kaikki natiivit ominaisuudet ovat käytössä erillisen ”paketointiohjelman” kautta.
- Sovelluksen käyttö ja käynnistäminen on natiivin sovelluksen kaltaista.

Hybridisovelluskehykset tarjoavat myös tuen yleisimmille eleohjauksille. Eleiden tunnistuksiin voidaan reagoida suoraan JavaScript-koodista, eikä tarvitse erikseen jokaisella alustalla varautua niihin. Hybridisovelluksissa on myös omat heikkoutensa, joita Wilken avaa kirjassaan

- Sovellus pyörii WebViewissä, joten suorituskyky on sidottu alustan selainmoottorin tasoon.
- Natiiveja ominaisuuksia käytetään kolmannen osapuolen apin kautta, joten kaikkiin ominaisuuksiin ei välttämättä päästä käsiksi tai joudutaan jo olemassa olevaa lisäosaa muuttamaan
- Ei voida käyttää natiiveja käyttöliittymäohjaimia, joten tarvitsee ulkopuolisen työkalun käyttöliittymän tekoon, esimerkiksi Ionic. [1; 2, s 7.]

Hybridisovellukset yhdistävät natiivien sovellusten ja web-sovellusten parhaat puolet. Hybridisovellukset eivät varsinaisesti ole mikään uusi tapa kehittää mobiilisovelluksia, mutta uudet kehitystyökalut ja sovelluskehykset yhdessä laitealustojen kehittyneiden selainominaisuuksien sekä laitteiden kasvaneen suorituskyvyn ansiosta hybridisovellukset ovat erittäin tehokas tapa kehittää mobiilisovelluksia. [1.]

3.4 Natiivi vai hybridi

Hybridisovellusten suorituskyvyn kasvaessa on aiheellista pohtia, millä tavalla sovellusta alkaa kehittää. John Bristowe kehottaa pohtimaan ainakin seuraavia asioita, ennen kuin valitsee sovelluksen tyypin:

- Mitä mobiilialustoja sovelluksen on tarkoitus tukea?
- Halutaanko sovellus julkaista laitealustojen sovelluskaupoissa?
- Pitääkö pystyä käyttämään kaikkia alustan natiiveja ominaisuuksia?
- Minkälaiset taidot kehitystiimillä on? [3.]

Krzysztof Marszałek kallistuu omassa tekstissään natiivin sovelluksen puolelle, mutta toteaa myös, että molemmille toteutuksille on käyttöä. Hän kertoo hybridisovelluksen toimivan hyvin siinä tapauksessa, että ohjelma on yksinkertainen, eikä sitä ole tarkoitus tehdä helposti jatkettavaksi, vaan on niin sanottu MVP-sovellus. Jos sovelluksesta on tarkoitus tehdä jatkokehitettävä ja muutenkin verrattain pitkä projekti, niin kannattaa ehdottomasti tehdä natiivina sovelluksena, vaikka kustannukset ja tieto taito vaatimukset ovat korkeammat, niin silti lopputulos on laadukkaampi. [5.]

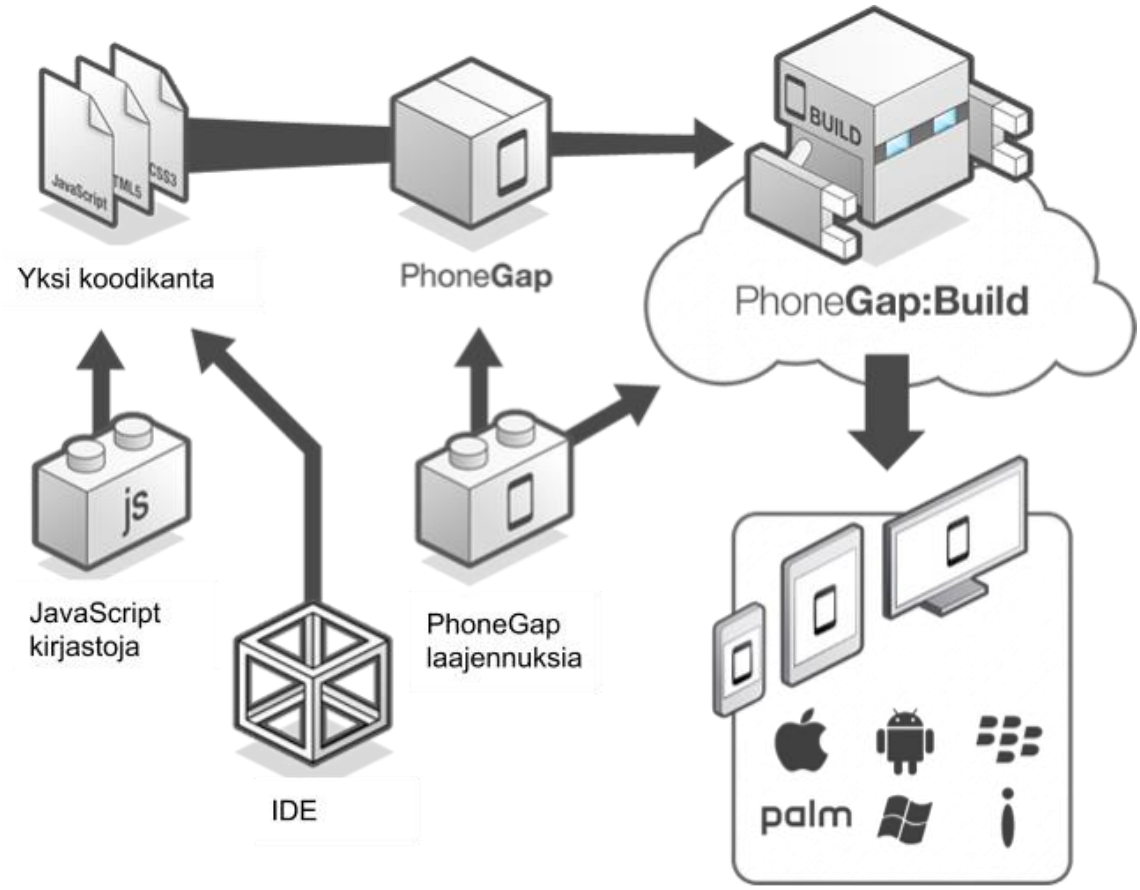
4 Hybridimobiilisovelluskehyksiä

Hybridisovelluksen kehitykseen käytetään usein HTML5- ja JavaScript-kieliä. Mutta niiden lisäksi tarvitaan jokin sovelluskehys, minkä kautta voidaan käyttää laitteen natiiveja ominaisuuksia. Mahesh Panhale esittelee kirjassaan tunnetuimpia työkaluja hybridisovelluskehitykseen. Kaikki esitellyt kehykset ovat niin sanottuja käyttöliittymäkehyksiä. [7, s 19.]

4.1 Phonegap

PhoneGap on enemmänkin ”paketointi”-kehys. Siinä voi kehittäjä itse valita JavaScript-kirjaston, jota käyttää, kuten jQuery, plainJS ja AngularJS. PhoneGap-ohjelman voi

kääntää pilvipalvelun kautta, jolloin ei tarvitse asentaa itselle paikallisesti SDK:ta jokaista alustaa varten. Kuvassa 2 on esitelty PhoneGap sovelluksen arkkitehtuuri. [7, s 49.]



Kuva 2. PhoneGap-sovelluksen arkkitehtuuri [8.]

4.2 Ionic

Ionic on avoimen lähdekoodin projekti, joka keskittyy tuottamaan sovelluksia malleja ja tapoja hyödyntäen. Ionicin JavaScript-ohjelmointiin käytetään AngularJS-kirjastoa. Ionic on sidottu Malli-Näkymä-Näkymämalli-suunnittelumalliin (engl. MVVM - Model – View – ViewModel pattern). MVVM-suunnittelumalli muodostuu kuvan 3 mukaisesti. [7, s 49.]

Kuva 3. MVVM-suunnittelumalli [9, s.12]

Ionicissa MVVM-malli toimii seuraavalla tavalla. Malli sisältää JSON- tai XML-tietoa, mikä voi olla tallennettuna paikallisesti ohjelmaan tai vastaanotettuna ulkopuoliselta palvelimelta. Näkymä on sovelluksen käyttöliittymä, jonka avulla käyttäjä voi seurata ja mahdollisesti muokata mallissa olevaa tietoa. Näkymän ja mallin välissä on näkymämalli. Näkymämalli yhdistää näkymän ja mallin tarjoamalla rajapinnan, millä mallissa olevan tiedon saa näkymän käyttämään muotoon. Näkymämallin ero yleisesti käytettyyn kontrolleriin tulee siitä, että myös näkymässä voidaan muokata mallissa olevaa tietoa, eli käytetään kahdensuunnan tiedon sidontaa. [7, s 49.]

4.3 AppBuilder

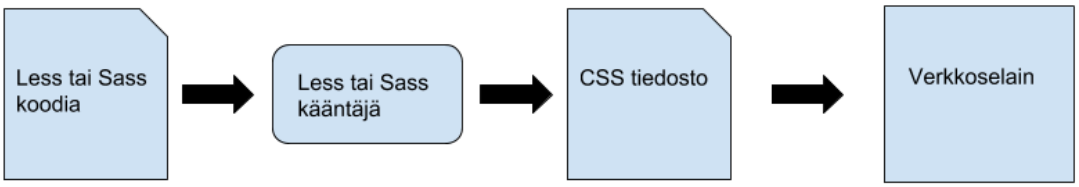
AppBuilder on Telerikin valmistama sovelluskehys. Aiemmin nimenä oli Icenium. AppBuilderissa on paljon samoja ominaisuuksia kuin PhoneGapissa, esimerkiksi pilvipalvelukääntäjän, komentorivikäyttöliittymän, selainkäyttöliittymän ja Windowsilla toimivan graafisen käyttöliittymän. Lisäksi AppBuilderista on saatavilla lisäosa (engl. plug-in) suosittuihin IDE:ihin, kuten Visual Studio ja Sublime Text. [7, s 50.]

4.4 KendoUI

KendoUI on toinen Telerikin valmistama sovelluskehys. Se on lähinnä tarkoitettu ammattilaiskäyttöön. Siinä on sisäänrakennettuna yli 70 käyttöliittymäkomponenttia ja monta valmista ulkoasuteemaa. KendoUI tukee AngularJS- ja bootstrap-kehyksiä. [7, s 50.]

Twitter Bootstrap

Bootstrap on suosittu HTML-, CSS- ja JavaScript-sovelluskehys. Se on suunniteltu erityisesti mobiilikäyttöliittymien tekemiseen. Bootstrap tarjoaa tuen myös kahdelle suositulle CSS-ennakkoprosessointimoottorille (engl. preprocessor), Less ja Sass. Kuvassa 4 esitellään CSS-ennakkoprosessoinnin idea. [11.]



Kuva 4. CSS-ennakkoprosessointi

4.5 Sencha Touch

Sencha Touch on tehokas sovelluskehys hybridisovellusten kehittämiseen. Sencha Touch sisältää yli 50 käyttöliittymäkomponenttia, joita yrityssovellukset tarvitsevat sulavan käyttökokemuksen saavuttamiseksi. Sencha on nopea käytössä. Se käyttää perinteistä malli-näkymä-ohjain (engl. Model-View-Controller) suunnittelumallia. JavaScript-kirjastoksi suositellaan ExtJS-kirjastoa. [7, s 51.]

Sencha Touch tarjoaa laitteelle mukautuvat käyttöliittymäasettelut ja tehokkaat animaatiot ja laitteelle sopivan kuvan vierityksen, jotta käyttökokemus olisi mahdollisimman natiivi. Sencha sisältää myös tiedon käsittelyyn räätälöidyt mallit, jotka helpottavat serveriltä tulleen tiedon esittämistä. Kaavioiden ja taulukoiden tekemiseen löytyy myös sisäänrakennetut toiminnot, joten niihinkään ei tarvitse ulkopuolisia kirjastoja. [12.]

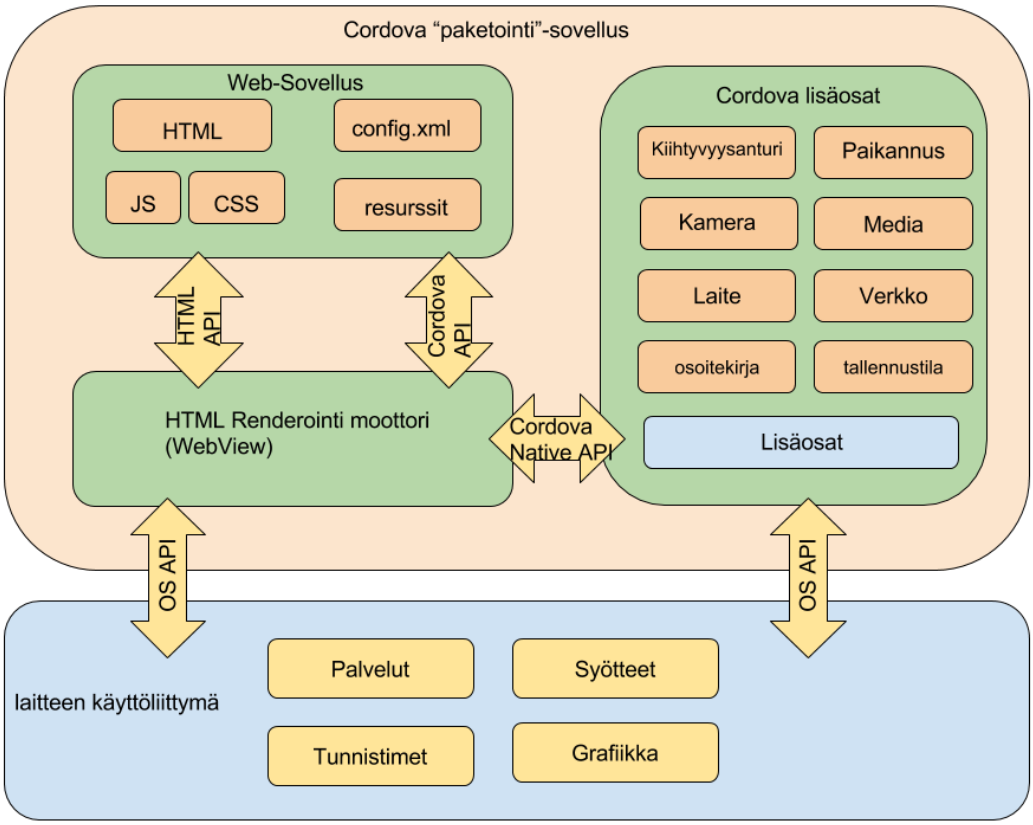
4.6 Mobile Angular UI

Mobile Angular UI on hyvin paljon Sencha Touchia muistuttava sovelluskehys. Sen käyttöliittymä ja ulkoasu hoidetaan aiemmin esitellyllä Bootstrapilla ja AngularJS:llä. Angular UI:n bootstrap on hiukan muokattu versio normaalista bootstrapista, sillä normaalisti se ei sisällä kaikkia mobiilissa tarvittavia ominaisuuksia. Koko sovelluskehys on ilmainen ja avoimen lähdekoodin projekti. [7, s 50.]

5 Apache Cordova

Kaikki edellisessä luvussa esitellyt sovelluskehykset käyttävät Apache Cordova -moottoria kääntämään ”paketointi”-sovelluksen kohde alustalle ja sen päälle rakentavat

WebViewissä pyörivän sovelluksen. Apache Cordovalla käännetyn sovelluksen korkean tason arkkitehtuuri selviää kuvasta 5. [7, s 49.]



Kuva 5. Apacher Cordova -sovellusarkkitehtuuri [6.]

5.1 WebView

Cordovan luoma natiivi ”paketointi”-sovellus luo opinnäytetyössä jo muutaman kerran mainitun WebView’n. Se on sovelluksen osa, mihin sovellus renderoi hybridisovellukseen sisältyvä web-sovelluksen. [6.]

5.2 Web-sovellus

Web-sovellus on Cordova-sovelluksen osa, missä sijaitsee Ionicilla tai jollain muulla sovelluskehyksellä kehitetty ohjelmakoodi. Kaikki JavaScript-, Html- ja CSS-tiedostot pysyvät siinä muodossa, missä ne kirjoitetaan. Sovelluksen pitää sisältää index.html-tiedosto, aivan kuten normaalissa web-sovelluksessakin, joka kertoo mitkä kaikki tiedostot

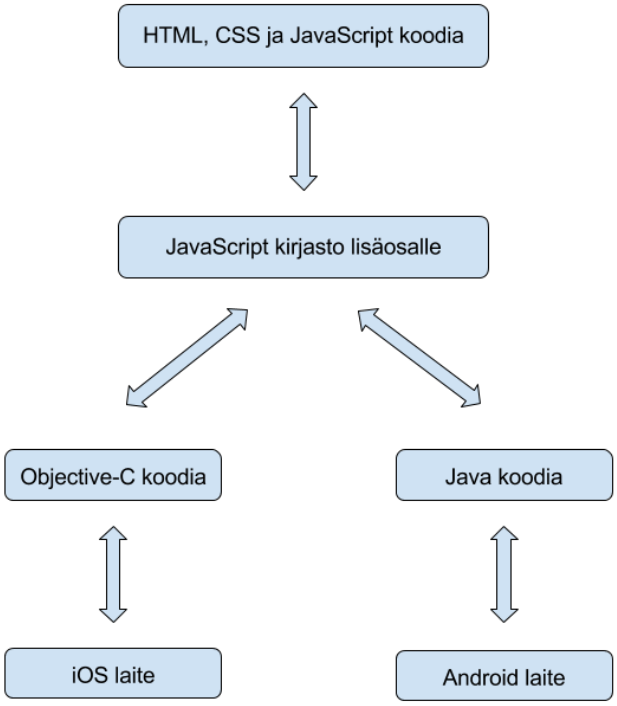
sovellus tarvitsee. Web-Sovelluksesta löytyy myös config.xml tiedosto, joka pitää sisällään koko sovellukselle tärkeitä tietoja, kuinka sen kuuluu toimia. Kuvassa 6 on esimerkki config.xml-tiedostosta. [6.]

```
<?xml version='1.0' encoding='utf-8'?>
  <widget id="io.cordova.hellocordova" version="0.0.1"
xmlns="http://www.w3.org/ns/widgets"
xmlns:cdv="http://cordova.apache.org/ns/1.0">
      <name>HelloCordova</name>
      <description>
          A sample Apache Cordova application that responds to the deviceready
event.
      </description>
      <author email="dev@cordova.apache.org" href="http://cordova.io">
          Apache Cordova Team
      </author>
      <content src="index.html" />
      <plugin name="cordova-plugin-whitelist" spec="1" />
      <access origin="*" />
      <allow-intent href="http://*/*" />
      <allow-intent href="https://*/*" />
      <allow-intent href="tel:*" />
      <allow-intent href="sms:*" />
      <allow-intent href="mailto:*" />
      <allow-intent href="geo:*" />
      <platform name="android">
          <allow-intent href="market:*" />
      </platform>
      <platform name="ios">
          <allow-intent href="itms:*" />
          <allow-intent href="itms-apps:*" />
      </platform>
  </widget>
```

Kuva 6. config.xml-esimerkki [13.]

5.3 Cordova-lisäosat

Lisäosat ovat keskeinen osa Cordovan ekosysteemiä. Ne luovat rajapinnan, jonka avulla Cordova ja natiivit ominaisuudet voivat keskustella keskenään. Lisäosat luovat myös sidoksen laitteen API:in. Se mahdollistaa laitteen API:n käytön suoraan JavaScript-koodista. Kuva 7 näyttää lisäosa-arkkitehtuurin moni-alustan lisäosille.



Kuva 7. Lisäosa-arkkitehtuuri [14.]

Apache Cordovan kehitystiimi ylläpitää ydinlisäosat (engl. Core Plugins) -nimistä kokoelmaa, joka sisältää laitteen yleisimpiä toimintoja, esimerkiksi akunkeston seurannan, kameran ja yhteystiedot.

Ydinlisäosien lisäksi on tarjolla paljon kolmannen osapuolen (engl. third party) kehittämiä lisäosia. Näistä kaikki eivät välttämättä toimi kaikilla alustoilla, joten pitää olla tarkkana lisäosaa asennettaessa, että kohdelaite varmasti tukee kyseistä lisäosaa. Lisäosia voi myös itse tehdä, mikä on tärkeää varsinkin, jos on tehnyt jotain omia natiiveja komponentteja kohdealustalle. [6.]

5.4 Sovelluskehitys Apache Cordovalla

Apache Cordovalla voi kehittää mobiilisovelluksia ilman mitään kolmannen osapuolen työkaluja. Se ei sisällä käyttöliittymätyökaluja eikä mitään suunnittelumallin mukaista sovelluskehystä, joten on yleistä käyttää jotain edellisessä luvussa esiteltyä sovelluske-

hystä tai vaihtoehtoisesti luoda itse sovelluskehys. Cordova-kehityksessä on kaksi vaihtoehtoa saavuttaa sama tavoite, monialustatyötapa (engl. Cross-platform workflow) ja alustakeskeinen työtapa (engl. Platform-centered workflow), mutta molemmissa on omat vahvuutensa. [6.]

5.4.1 Monialustatyötapa

Monialustatyötapa on hyödyllinen silloin, kun halutaan tukea mahdollisimman montaa alustaa mahdollisimman vähäisellä alustakohtaisella kehityksellä. Tämä työtapa keskittyy käyttämään Cordova CLI:tä. CLI on korkeantason työkalu, mikä abstrahoi mahdollisimman paljon matalan tason koodin toiminnollisuuksia. CLI luo jokaiselle alustalle alikansion, johon se kopioi yleiset web-tiedostot, tekee tarvittavat muutokset asetustiedostoon ja kääntää jokaiselle alustalle ajettavan tiedoston. CLI tarjoaa myös kaikille alustoille yhteisen käyttöliittymän lisäosien liittämiseen projektiin. [6.]

5.4.2 Alustakeskeinen työtapa

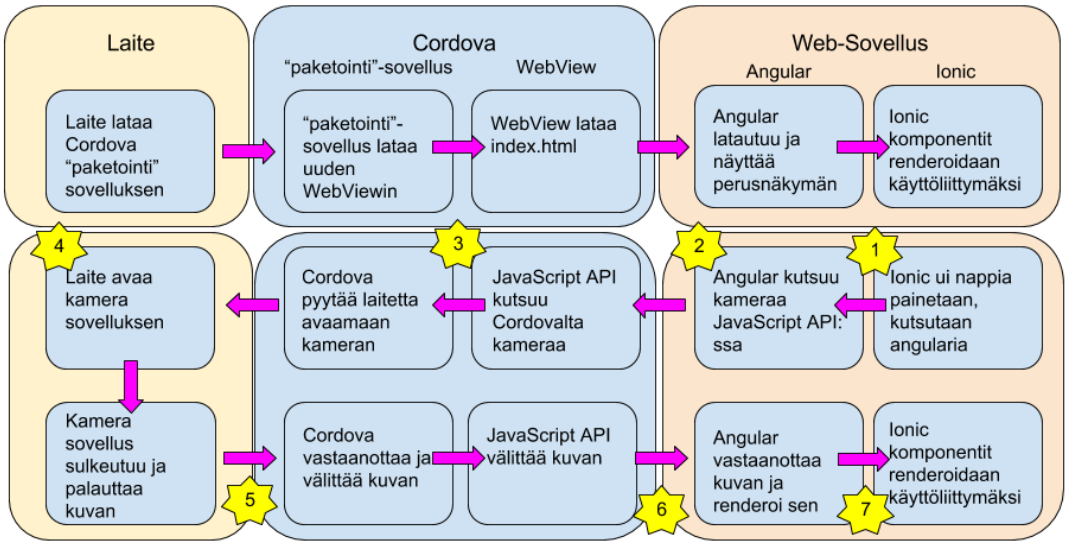
Alustakeskeistä työtapaa kannattaa käyttää, jos haluaa keskittyä vain yhteen alustaan kerrallaan ja pitää pystyä muokkaamaan sovellusta matalalla tasolla. Tätä työtapaa on pakko käyttää silloin, jos halutaan yhdistellä sovelluksessa räätälöityjä natiiveja komponentteja ja web-pohjaisia Cordova-komponenteja. Perussääntönä voidaan pitää, että alustakeskeinen työtapa on oikea silloin, jos sovellusta halutaan muokata alustan omaa SDK:ta hyödyntäen. [6.]

6 Ionic-mobiilisovelluskehys

Opinnäytetyön käytännön toteutus on kehitetty Ionic-sovelluskehyksellä. Ionic-mobiilisovelluskehitystä on helppo lähestyä, jos on perustietämystä yleisimmistä verkkosovelluskehityskielistä, eli HTML:stä, CSS:stä ja JavaScriptistä. Ionicin tekniikka jakautuu kolmeen osaan, joita ovat Ionic, AngularJS ja Apache Cordova. [2, s 7;11]

6.1 Teknologia

Kuvan 8 yläosassa tarkastellaan Ionicin tapausta kuvan 1 hybridisovellusarkkitehtuurista. Toisella ja kolmannella rivillä on käyttötapaus, joka ottaa kuvan Ionic-sovelluksessa laitteen kameralla.



Kuva 8. Ionic arkkitehtuuri ja käyttötapaus esimerkki [2, s 8.]

Kuvan 8 käyttötapaus on numeroitu yhdestä seitsemään.

1. Käyttäjä painaa käyttöliittymän nappia.

2. Napin painallus kutsuu Angular ohjainta, joka kutsuu Cordovaa JavaScript API:n kautta.

3. Cordova keskustelee laitteen kanssa käyttäen laitteen natiiveja ajureita.

4. Laite pyytää luvan kameran avaamiseen ja avaa kameran käyttäjälle.

5. Kuvan ottamisen jälkeen kamera sovellus sulkeutuu ja palauttaa kuvan Cordovalle.

6. Cordova välittää kuvan Angular ohjaimelle.

7. Kuva renderoidaan näytölle ja samalla päivitetään Ionicin luoma käyttöliittymä.

Kolmesta käytetystä tekniikasta Apache Cordova on esitelty kappaleessa 5, mutta Ionic on esitelty vain päälisin puolin eikä AngularJS:ää vielä ole esitelty.

6.1.1 Ionic-käyttöliittymäkehys

Ionicin pääominaisuus on tuoda käyttöliittymätyökalut hybridisovelluskehitykseen, mitkä puuttuvat HTML:stä, mutta ovat yleisiä mobiilisovelluksissa. Ionic on hyvin dokumentoitu ja siitä löytyy sisäänrakennettuna lähes kaikki mobiilisovelluksissa tarvittavat käyttöliittymäkomponentit. Sovelluksen ulkoasu on perusasetuksilla natiivin iOS-sovelluksen näköinen ja tuntuinen, mutta helposti muokattavalla CSS:llä voi ulkoasua muuttaa mieleisekseen. Ionic tukee myös kappaleessa 4.4.1 esiteltyä Sass-ennakkoprosessointimoottoria. [2, s.8-10]

6.1.2 AngularJS

AngularJS on 2009 aloitettu JavaScript-sovelluskehys. Kehityksen aloitti Misko Hevery ja Adam Abrons, mutta myöhemmin Hevery meni Googlelle, ja tätä nykyä AngularJS on googlen kehityksessä. [2, s 10.]

Arvind Ravulavaru esittelee Angularia kirjassaan MVC (engl. Model-View-Controller) suunnittelumallin mukaisena sovelluskehyksenä. Angularin tapauksessa roolit voidaan jakaa seuraavalla tavalla:

- Malli (engl. model) on scope-objekteissa oleva tieto.
- Näkymä (engl. view) on HTML-tiedoston luoma näkymä.
- Ohjain (engl. controller) on JavaScript koodissa olevat Angular ohjaimet. [16, s 5.]

Angularin voi myös mieltää MVC-mallin sijasta MVVM-mallin mukaiseksi, mikä on esitelty luvun 4.2 kuvassa 3. MVVM-mallissa korvataan ohjain näkymä-mallilla. Näkymämalli sopii ohjainta paremmin Angularin kahden suunnan tiedonsidontaan. [15, s 2.]

6.2 Ionicin tarjoamia työkaluja

Ionic-sovelluskehys tarjoaa kehitystä varten monia työkaluja, jotka helpottavat ja nopeuttavat jokapäiväistä kehitystyötä Ionicilla. Ionic CLI on työkalu monien toimintojen tekemiseksi komentoriviltä. Ionic Platform on pilvessä pyörivä graafinen käyttöliittymä sovellusten hallinnointiin. Ionic Creatorilla voidaan tehdä nopeita prototyyppisovelluksia raahaa ja pudota (engl. drag and drop) periaatteella. Sovelluksia voi kehitysvaiheessa testata Ionic Viewillä ilman erillistä käännösprosessia jokaiselle alustalle. Ionic CLI-komentorivi työkalulle on graafinen vastine Ionic Lab, [16; 17]

6.2.1 Ionic CLI

Helpoin tapa asentaa ja ottaa käyttöön Ionic CLI on NodeJS-paketinhallintatyökalulla (npm). Esimerkki asennuskomennosta kuvassa 9.

```
$ npm install -g ionic
```

Kuva 9, Ionic CLI -asennuskomento [17.]

Seuraavassa listausta Ionic CLI:n yleisimmistä komennoista.

Ionic serve, emulate and run

Serve, emulate ja run ovat käytetyimpiä ja hyödyllisimpiä komentoja Ionic CLI:ssä. Näillä komennoilla voidaan käynnistää sovelluksen suoraan komentoriviltä eri alustoille. Serve käynnistää sovelluksen selaimessa, emulate laitteen simulaattorissa ja run suoraan laitteella. Kuvaan 10 on koottu komennot ja niille mahdolliset parametrit. [16, s 339.]

```
$ ionic serve [options]


[--consolelogs|-c] ......  Tulostaa sovelluksen logit komentoriville
[--serverlogs|-s] .......  Tulostaa serverin logit komentoriville
[--port|-p] .............  Dev server HTTP portti (8100 default)
[--livereload-port|-i] ..  Live Reload portti (35729 default)
[--nobrowser|-b] ........  Poistaa selaimen käynnistyksen
[--nolivereload|-r] .....  Poistaa live reload -toiminnon
[--noproxy|-x] ..........  Ei käytä välityspalvelinta

$ ionic emulate <alusta> [options] ja $ ionic run <alusta> [options]

[--livereload|-l] .......  Live Reload sovelluksen tiedostot laitteeseen (beta)
[--consolelogs|-c] ......  Tulostaa sovelluksen logit komentoriville (live
reload req.)
[--serverlogs|-s] .......  Tulostaa serverin logit komentoriville (live reload
req.)
[--port|-p] .............  Dev server HTTP portti port (8100 default, live
reload req.)
[--livereload-port|-i] ..  Live Reload portti (35729 default, live reload req.)
[--debug|--release]
```

Kuva 10. Serve-, emulate- ja run-komennot ja parametrit [18.]

Ionic start

```
$ ionic start (nimi) [malli]

--appname, -a  .......  Ihmisen luettava nimi
                        (käytä lainausmerkkejä nimen ympärillä)
--id, -i  ............  Pakettinimi sovellukselle <widget id> config
                        ex: com.mycompany.myapp
--no-cordova, -w  ....  Ei kohdenneta sovellusta cordova alustalle
```

Kuva 11. ionic start-komento ja parametrit [18.]

Kuvan 11 start-komentoon malliparametrin (engl. template) kohdalle voi olla nimetty malli, github-varasto, codepen-projekti tai paikallinen hakemisto. [18.]

Ionic login, upload ja share

Ionic login-komennolla voi kirjautua Ionic-pilvitilille, joka on tehty Ionicin verkkosivulla. Kuvassa 12 on kaksi eri tapaa käyttää login-komentoa. Ensimmäisessä tavassa käyttöliittymä kysyy ensiksi sähköpostiosoitteen ja sen jälkeen salasanan. Toisessa tavassa sähköpostiosoite ja salasana annetaan parametrina. [16, s 331.]

```
$ ionic login
```

tai

```
$ ionic login –email [sähköpostiosoite] –password [salasana]
```

Kuva 12. Ionic login-komennon vaihtoehtoiset käyttötavat [16, s 331.]

Ionic upload-komento pyytää aluksi kirjautumaan Ionic-tilille, jos ei ole vielä kirjauduttu. Onnistuneen kirjautumisen jälkeen komento lähettää sovelluksen tiedostot Ionicin pilvipalveluun.

Pilvipalvelussa olevia projekteja voi jakaa kenen kanssa tahansa käyttämällä Ionic share komentoa. Komennolle annetaan parametrina vastaanottajan sähköposti. [18.]

Ionic resources

Aina kun projektiin lisätään uusi alusta, niin resurssit-kansio luodaan ja generoidaan Ionicin icon.png ja splash.png pohjalta alustalle kuvake ja latausruutu. Sovellukselle voi luoda itse kuvakkeen ja latausruudun. Omalla kuvakkeella pitää korvata resurssikansiossa oleva ”icon.png”-tiedosto ja latausruudun tiedostolla korvataan ”splash.png”-tiedosto. Molemmissa tiedostoissa pitää kuitenkin säilyttää alkuperäinen nimi, että ”resources”-komento osaa generoida niistä sopivat jokaiselle alustalle. [16, s 338.]

```
$ ionic resources [parametri]
```

ei mitään	generoidaan kuvakkeet ja latausruutu
--icon	generoidaan vain kuvakkeet
--splash	generoidaan vain latausruutu

Kuva 13. Ionic resources ja parametrit [18.]

Kuvassa 13 malli ”resources”-komennosta ja sen parametreista.

6.2.2 Ionic Platform

Ionic Platform on Ionicin pilvipalvelu, joka tarjoaa palveluita, joiden avulla voidaan mm. kääntää sovellus, lähettää sovelluspäivityksiä ja skaalata sovellusta tehokkaasti. [17.]

Ionic package

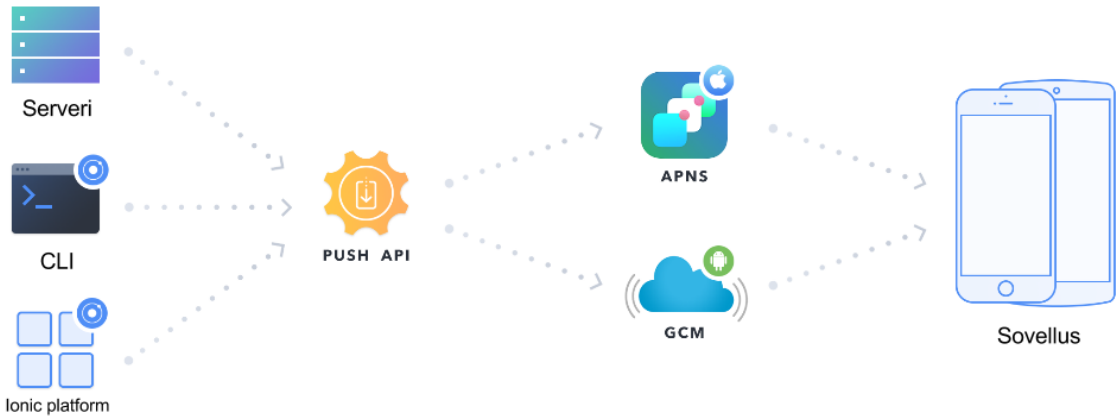
Ionic package on toiminto, jonka avulla voidaan lähettää valmiin sovelluksen lähdekoodit pilvipalvelulle, joka kääntää sovelluksen halutulle alustalle. Pilvipalvelukääntämisen hyvä puoli on se, että voidaan kääntää sellaisille alustoille mitä käyttöjärjestelmä ei paikallisesti tue, kuten Windows-käyttöjärjestelmässä iOS-käännöksiä. Käännöksen valmistuttua pilvipalvelussa pyörivä käännösserveri lähettää valmiin IPA- (iOS) tai APK- (Android) tiedoston, jonka voi suoraan lähettää sovelluskauppaan. Käännöstapahtuma on esitelty kuvassa 14. [17.]

Kuva 14. Ionic package -työnkulku [17.]

Ionic push

Graafisesta käyttöliittymästä voi lähettää muuttuvia ja kohdennettuja push-viestejä kaikille käyttäjille, tai erikseen valituille ryhmille. Push-viestejen lähetyksen voi helposti ajoittaa eri ryhmille, joten saman viestin voi näyttää jokaisella aikavyöhykkeellä eri aikaan. Ionic tarjoaa myös työkalut analysoimaan push-viestien saavuttavuutta. Ionic tarjoaa myös rajapinnan kehittäjän omille palvelimille, joiden kautta push-viestit voidaan lähettää. Kuva 13 havainnollistaa push-viestin lähetystä Ionicissa.

Push-api toimii Ionicin tarjoamana välikätenä sovelluksen ja alustojen push-palvelimien välissä. Se tarjoaa normaaleiden viestien lähetyksen lisäksi mm. viestianalytiikkaa ja laitetiedoille tallennustilan. [17.]



Kuva 15. Push-viestin lähetys [18.]

Ionic user

User on palvelu, minkä avulla sovellukseen voidaan luoda käyttäjän todennusta. Todennukseen voidaan käyttää Ionicin tarjoamaa autentikointipalvelinta, oma kehittämää autentikointipalvelinta tai jonkin sosiaalisenverkoston autentikointipalvelinta. Koodiesimerkki käyttäjän lisäämisestä järjestelmään kuvassa 16. [18.]

```
var details = {
  'email': 'email@example.com',
  'password': 'secret'
}

// vaihtoehtoisesti voi antaa käyttäjänimen muodossa
// details.username = 'ionitron';

Ionic.Auth.signup(details).then(signupSuccess, signupFailure);
```

Kuva 16. Ionic user -käyttäjän lisäys JavaScript [18.]

Ionic.Auth.signup -funktio tarkistaa, onko syötetty käyttäjätunnus tai sähköpostiosoite valmiiksi palvelimen tietokannassa. Jos ei ole, niin uusi käyttäjä luodaan ja mennään ”lupaus täytetty” (engl. promise onFulfilled) -rutiiniin, esimerkkitapauksessa signupSuccess-funktioon. Silloin kun käyttäjän lisäyksessä menee jokin pieleen, mennään lupauksen ”lupaus hylätty” (engl. promise onRejected) -rutiiniin, esimerkissä signupFailure-funktioon. Funktio saa argumenttina error-merkkijonon, joka kertoo virheen syyn ja siihen voidaan sovelluksessa reagoida. Taulukossa 2 on lueteltu mahdolliset error-merkkijonot. [18; 19]

Taulukko 2. Lupaus hylätty -rutiinin virheet [18.]

Error	Kuvaus
`required_email`	sähköpostikenttä puuttu
`required_password`	salasanakenttä puuttu
`conflict_email`	sähköpostiosoite on jo käytössä
`conflict_username`	käyttäjänimi on jo käytössä
`invalid_email`	email ei kelpaa

Käyttäjän lisäyksen onnistuessa autentikointi on turvallista ja voidaan siirtyä sisäänkirjautumiseen. Siitä lisää kuvassa 17.

```
Ionic.Auth.login(provider, settings, data).then(success, failure);
```

Kuva 17. Ionic sisäänkirjautumisen funktio [18.]

Sisäänkirjautumisen funktiossa on käytetty taas JavaScriptin lupaus-oliota. Samaan tapaan kuin käyttäjän lisäyksessä sisäänkirjautumisen onnistuessa kutsutaan kuvan 17 mukaisesti ”success”-funktiota ja epäonnistuessa ”failure”-funktiota. Parametreille ”provider”, ”settings” ja ”data” mahdolliset tyypit ja arvot listataan taulukossa 3. [18.]

Taulukko 3. Ionic login parametrejen arvot [18.]

Muuttuja	Tyyppi	Kuvaus	Arvot
provider	String	Autentikointipalveluntarjoajan nimi	`basic`, `facebook`, `github`, `google`, `instagram`, `linkedin`, `twitter`
settings	Object	Autentikointiasetukset, tällä hetkellä ainoa tuettu arvo on remember, true tai false	{ "remember": true }
data	Object	Mahdolliset lisätiedot autentikointia varten	Basic auth vaatii objektin missä on email ja salasana

Ionic deploy

Ionic deploy -palvelu mahdollistaa muutosten tekemisen jo jakelukanavissa olevaan sovellukseen. Palvelun kautta tehtyihin muutoksiin ei tarvitse joka kerta odottaa sovelluskauppojen varmistusprosessia, vaan muutokset näkyvät käyttäjällä heti, kun uudet tiedot ladataan laitteeseen päivityksenä sovelluksen sisältä. Muutosten kanssa täytyy kuitenkin olla tarkkana, sillä sovellus poistetaan sovelluskaupoista, jos tyyli muuttuu liika tai uudet

ominaisuudet rikkovat sovelluskauppojen sääntöjä. Uudet muutokset eivät myöskään voi käyttää mitään uusia ominaisuuksia, jotka vaatisivat sovelluksen binäärikodin muuttamista, kuten uudet lisäosat Cordovaan. Muutokset voidaan kumota samasta käyttöliittymästä, mistä ne laitetaan käyttöönkin. Jos halutaan kokeilla montaa uutta ulkoasua yhtä aikaa, voidaan palvelun kautta tehdä myös niin sanottua AB-testausta, missä eri käyttäjä ryhmille annetaan eri versio sovelluksesta. [17; 18]

Ionic analytics

Ionic analytics tarjoaa työkaluja haluttujen asioiden käyttöasteen seuraamiseen. Jokainen seurattu tapahtuma lähettää Ionicin palveluun tapahtumaobjektin (engl. event object). Jokainen tapahtumaobjekti pitää sisällään tapahtuman tyypin ja tapahtuman tiedot. Tapahtuman tyyppi on merkkijono, joka kertoo mistä tapahtumasta on kyse. Tapahtuman tiedoista selviää yksityiskohtaisempaa tietoa tapahtumasta. Käyttöönoton jälkeen Ionic analytics kerää automaattisesti tietoa sovelluksen käynnistyskerroista, joka on tärkeää tietoa varsinkin sovelluksissa, jotka sisältävät maksullista sisältöä. [18.]

```
.run(function($ionicPlatform, $ionicAnalytics) {
  $ionicPlatform.ready(function() {

    $ionicAnalytics.register();

  });
})
```

Kuva 18. Ionic analytics rekisteröinti käytettäväksi [18.]

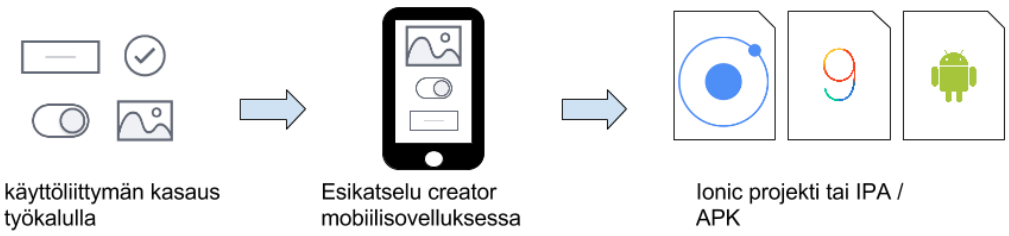
Kuvassa 18 on esimerkki, kuinka Ionic analytics otetaan käyttöön sovelluksessa. Tapahtuman lähetyksen analytiikkatyökalulle voi toteuttaa kahdella tavalla. Tiedon voi lähettää joko JavaScript-koodista tai suoraan Html-direktiivin kautta. Direktiivit esitellään tarkemmin luvussa 6.3.3. Koodimuotoisena molemmat tavat näkyvät kuvassa 19.

```
//JavaScript toteutus

.controller('controller', function($ionicAnalytics) {

  $ionicAnalytics.track('Buy soul', {
    item_id: 666,
    item_name: 'satan's soul'
  });

});

//Html directive toteutus Html button elementtiin

<button ion-track-tap="Sell" ion-track-data="{ item:20 }">Sell</button>
```

Kuva 19. Tapahtuman lähetys JavaScriptistä ja Html-direktiivistä [18.]

6.2.3 Ionic Creator

Ionic Creator on pilvipalveluna toimiva käyttöliittymäsuunnitteluun tarkoitettu työkalu. Raahaa ja pudota -tyyppisellä tavalla creatorilla on helppo ja nopea suunnitella tulevan ohjelman prototyyppiä. Creatorin käyttöön ei tarvitse välttämättä tietää ohjelmoinnista mitään, koska valmiin ulkoasun voi ladata Ionic-projektina ja lisätä siihen myöhemmin tarvittavat koodit. Kuvan 20 kulkukaaviosta voi tutustua creatorin työnkulkuun. [16, s 341-341; 17.]



Kuva 20. Ionic creator -työnkulku [17.]

Ionic creator -palvelussa luodaan projekti, jonka jälkeen voidaan raahata halutut komponentit paikalleen. Suunnitteluvaiheessa olevaa projektia voidaan tarkastella mobiililaitteessa creatorin tarjoamalla mobiilisovelluksella. Valmiin projektin voi ladata jatkokehitettäväksi Ionic-projektina. Jos projekti ei vaadi enää erikseen lisättävää ja on valmis julkaistavaksi, voi creatorista ladata myös valmiiksi käännetyn sovelluksen iOS- tai Android-alustalle. [17.]

6.2.4 Ionic View

Ionic View on mobiilialustojen sovelluskaupoista ladattava sovellus, joka mahdollistaa kehitettävän sovelluksen testauksen eri laitteilla. Sovelluksen suurin hyöty on testaaminen alustalla, jota kehitysympäristö ei tue, esimerkiksi iOS-sovellus Windows-ympäristössä.

6.2.5 Ionic Lab

Ionic Lab on Ionic CLI:n graafinen vastine. Lab on saatavilla kaikille yleisimmille käyttöjärjestelmille, eli Windows, Linux ja Mac. Ionic Labissa voi helposti seurata kaikkia kehityksessä olevia projekteja. Sovelluksen kääntäminen Labin avulla halutuille alustoille on komentoriviin verrattuna vaivattomampaa, eikä tarvitse muistaa niin paljon ulkoa käännösvaiheen komentoja. [17.]

6.3 Kehitystyö Ionicilla

Ionic-kehityksessä käytetyt CSS, HTML5 ja JavaScript mahdollistavat lähes minkä tyyppisen sovelluksen kehittämisen. Omien CSS-tiedostojen tekeminen ei ole Ionicin tapauksessa välttämätöntä, sillä Ionic tarjoaa jo itsessään laajan valikoiman käytettäviä CSS-luokkia.

6.3.1 CSS

CSS eli porrastettu tyyliarkki (engl. Cascading Style Sheet) on erityisesti WWW-dokumenteille kehitetty tyylitiedosto, joka antaa ehdotuksen, miltä verkkosivun pitäisi näyttää. [21.]

Ionicin tarjoamat valmiit CSS-luokat sisältävät kaikki perinteiset mobiilisovelluksessa tarvittavat käyttöliittymäkomponentit. Esimerkkinä mainittakoon natiiveista sovelluksille tyypilliset ylä- ja alapalkki, välilehdet ja alustariippuvaiset napit ja valintaruudut. Ionic tarjoaa myös käyttöliittymäkomponenttien sijoittelua helpottavan solukkosysteemin. Esimerkki solukon käytöstä on kuvassa 21.

```
//Tasan jaetut sarakkeet
<div class="row">
  <div class="col">.col</div>
  <div class="col">.col</div>
  <div class="col">.col</div>
</div>

//Eri levyiset sarakkeet
<div class="row">
  <div class="col col-33">.col</div>
  <div class="col col-67">.col</div>
</div>

//Sisennetyt sarakkeet
<div class="row">
  <div class="col col-33 col-offset-33">.col</div>
  <div class="col col-33 col-offset-67">.col</div>
</div>
```

Kuva 21. Ionic -ruudukkosysteemi [20.]

Kuvan 21 ensimmäisessä kohdassa on esimerkki, missä rivi jaetaan kolmeen yhtä suureen sarakkeeseen. Toisessa kohdassa rivi jaetaan kahteen sarakkeeseen, joista toinen on kolmanneksen ruudusta ja toinen on kaksi kolmannesta. Viimeisessä esimerkissä on ruudukolla tehty sisennys. Ensimmäinen sarake on sisennetty kolmasosan ruudusta ja toinen rivi kaksi kolmasosaa.

Jos sovellukseen halutaan tehdä animaatiota, niin kannattavin tapa se on toteuttaa CSS-animaationa. Animaatioiden tekemiseen Ionic ei suoraan tarjoa työkaluja, mutta tukee räätälöityjä ratkaisuja. Animaatio CSS-koodista on kuvassa 22. [20.]

```
@keyframes slideInUp {
 0% {
  transform: translate3d(0%,100%,0);
 }

 100% {
  transform: translate3d(0,0,0);
 }
}

.slide-in{;
 animation: slideInUp ease-in 1;
 animation-fill-mode: forwards;
 animation-duration: 750ms;
}
```

Kuva 22. CSS-animaatioesimerkki

Kuvan 22 animaatio esimerkissä esitellään koodimuodossa sisällön ilmestyminen ruudulle liukuen alhaalta ylöspäin.

6.3.2 HTML5

HTML5 on uusin versio verkkosivujen tekemiseen tarkoitetusta HTML:stä. HTML5 keskeinen uudistus on, että se kuvaa tarkasti dokumenttioliomallia (engl. Document Object Model). Nykyinen määrittely kuvaa HTML-dokumenttia oliona, jolla on puurakenne. Puurakenne helpottaa HTML-dokumentin ohjelmallista käsittelyä esimerkiksi JavaScript koodista. HTML5-tagit ovat väline esittää puurakennetta tekstimuodossa. [22.]

Ionicissa HTML5 on näkymäosuus. Kaikki mitä CSS-tiedostoissa ja JavaScript-koodissa tapahtuu, heijastetaan käyttäjälle HTML5-dokumentin kautta.

6.3.3 AngularJS

Moduulit

Angular-moduulit ovat paketteja, jotka sisältävät sovelluksen itsenäiset koodiosuudet. Moduuleissa on kaksi osaa. Ensimmäisessä osassa voidaan määrittää moduulille omat ohjaimet, palvelut ja direktiivit. Näitä kaikkia voidaan käyttää moduulin kautta. Toisessa osassa moduulille voidaan määrittää riippuvaisuuksia toisiin moduuleihin, jolloin moduulin sisältä voidaan kutsua kaikkia riippuvuuksissa lueteltujen moduuleiden funktioita ja palveluita. Moduulisysteemi mahdollistaa koodin uudelleenkäytettävyyden. Moduuleilla siis sidotaan koodit yhdeksi toimivaksi kokonaisuudeksi. [15, s 15.]

```
//HTML toteutus
<html ng-app=”exampleApp”>
 <head><title>Hello</title></head>
 <body>
  Angular app {{1+1}}
 </body>
</html>

//JavaScript osuus
angular.module(‘exampleApp’, []);
```

Kuva 23, AngularJS-moduuli esimerkki [15, s 17.]

Kuvaan 23 on kirjoitettu pieni esimerkkiohjelma, kuinka AngularJS moduuli luodaan JavaScriptissä ja otetaan käyttöön ng-app direktiivillä HTML-koodissa. JavaScript koodissa oleva ”angular.module” saa ensimmäisenä parametrina moduulin nimen ja toisena parametrina riippuvuudet taulukkona. Tässä tapauksessa ei ole riippuvuuksia.

Ohjaimet

Ohjain on Angularin niin sanottu työhevonen. Ohjaimet ovat JavaScript-funktioita, jotka hoitavat lähes kaikki käyttöliittymätehtävät. Yleisimmät ohjaimen vastuut Angular-sovelluksessa ovat:

- Ladata serveriltä oikea tieto sen hetkiselle näkymälle.
- Päättää, mitkä tiedot näytetään käyttäjälle.
- Esitystapalogiikka. Määrittää millaisena käyttöliittymä esitetään.
- Reagoida käyttäjän painalluksiin ja syötteisiin.

Ohjaimet ovat lähes poikkeuksetta yhteydessä HTML-näkymään. Ne toiminnollisuudet, jotka eivät liity näkymään, kuuluvat palveluihin. Ohjaimen tehtävä on siis välittää tietoa mallilta näkymälle. [15, s 17.]

```
//JavaScript
angular.module('exampleApp', [])
 .controller('esimCtrl', [function() {
   //tänne controllerin koodia
 }])

//HTML
<html ng-app="exampleApp">
 <body ng-controller="esimCtrl">
   Täällä on käytössä esimCtrl!
 </body>
</html>
```

Kuva 24. AngularJS-ohjainesimerkki [15, s 18.]

Kuvassa 24 aluksi luodaan edellisestä kappaleesta tuttu AngularJS-moduuli. Moduulin sisälle luodaan ohjain nimeltä "esimCtrl". Ohjain saa toisena parametrina taulukon, joka sisältää kaikki ohjaimen riippuvuudet. Tässä esimerkissä ohjaimella ei ole riippuvuuksia, joten taulukossa on vain yksi alkio, ohjaimen toteutus. Taulukon viimeinen alkio pitää olla aina toteutusfunktio.

Palvelut (engl. services)

Palvelut voivat olla funktioita tai objekteja, jotka sisältävät toimintoja tai sovelluksen tilaan liittyviä muuttujia. Jokainen palvelu on singleton-suunnittelumallin mukainen, eli siitä luodaan sovelluksessa vain yksi ilmentymä. Tämän ansiosta palvelut toimivat hyvin tiedon välittäjinä sovelluksen sisällä. [15, s 69.]

Tyler McGinnis avaa omassa tekstissään kolme eri tapaa luoda palvelu AngularJS:ssä. Käytetään tehdasmetodia (engl. factory), jos halutaan tehdä funktionaalinen palvelu. Palvelu metodia käytetään olio tyyppisessä ratkaisussa. Jos halutaan käyttää palvelua sovelluksen config-metodissa, silloin palvelun pitää olla tarjoajametodi (engl. provider). [23.]

Scope

AngularJS $scope-muuttuja on objekti, joka näkyy JavaScript-koodissa ja HTML-näkymässä. Objekti voi sisältää muuttujia, tietorakenteita tai funktioita. Scope-muuttuja saadaan käyttöön näkymässä ng-controller-direktiivillä. Scope-muuttujan voi korvata ”controllerAs”-syntaksilla. Kuva 25 havainnollistaa scope-muuttujan ja ”controllerAs”-syntaksin. [15.]

```
//JavaScript
angular.module('exampleApp', [])
  .controller('esimCtrl', ['$scope', function($scope) {
    $scope.teksti = "laalaa";
    this.toinenTeksti = "baabaa";
  }])

//HTML
<html ng-app="exampleApp">
  <body ng-controller="esimCtrl as ctrl">
    {{teksti}}
    {{ctrl.toinenTeksti}}
  </body>
</html>
```

Kuva 25. $scope- ja controllerAs-esimerkit

Direktiivit

Direktiivit ovat Angularin tapa tuoda lisää mahdollisuuksia DOM-puun rakenteen muokkaamiseen ja kolmannen osapuolen komponenttien lisäämiseksi käyttöliittymään. Direktiivit voidaan jakaa kahteen pääryhmään:

- Käyttäytymistä muokkaavat direktiivit. Tällaiset direktiivit liitetään valmiisiin käyttöliittymäkomponentteihin tai html-koodinpätkiin. Esimerkkinä tämän tyyppisestä direktiivistä on ng-show, joka mahdollistaa käyttöliittymäkomponenttien piilottamisen ja näyttämisen sidottuna tiettyyn arvoon.

- Uudelleen käytettävät direktiivit. Näillä voidaan helposti tuoda pitkä HTML-koodi yhdellä rivillä uudelleen käytettäväksi toisessa näkymässä. Esimerkkejä tästä ryhmästä ovat yleisesti käytetty karuselli kuvia varten ja erilaiset diagrammit. [15, s 169-170.]

Esimerkkejä direktiivin käytöstä voi katsoa kuvasta 25. Siinä esiintyvät ng-app ja ng-controller ovat molemmat yleisiä direktiivejä.

Filtterit

Filtterit ovat yleensä viimeinen vaihe tiedon muuntamisessa luettavaan muotoon. Yleinen esimerkki filtterin käytöstä on laittaa arvon perään valuutan tunnus tai mittayksikkö. Filttereitä voidaan käyttää suoraan Angularin tarjoamalla HTML-lausekkeella (engl. expression), eli ympäröidään arvon muuttuja hakasulkeilla ja käytetään arvon jälkeen pystyviiva-merkkiä ilmaisemaan, mikä filtteri otetaan käyttöön. Toinen tapa on kutsua filtteriä ohjaimessa, jolloin arvo muutetaan jo JavaScript-puolella oikeanlaiseksi. Kuvaan 26 on muokattu kuvan 25 esimerkkiä käyttämään filttereitä HTML-koodissa. [15, s 121.]

```
//JavaScript
angular.module('exampleApp', [])
 .controller('esimCtrl', ['$scope', function($scope) {
  $scope.teksti = "10101;
  this.toinenTeksti = "baabaa";
 }])

//HTML
<html ng-app="exampleApp">
 <body ng-controller="esimCtrl as ctrl">
  {{teksti | currency}}
  {{ctrl.toinenTeksti | uppercase}}
 </body>
</html>
```

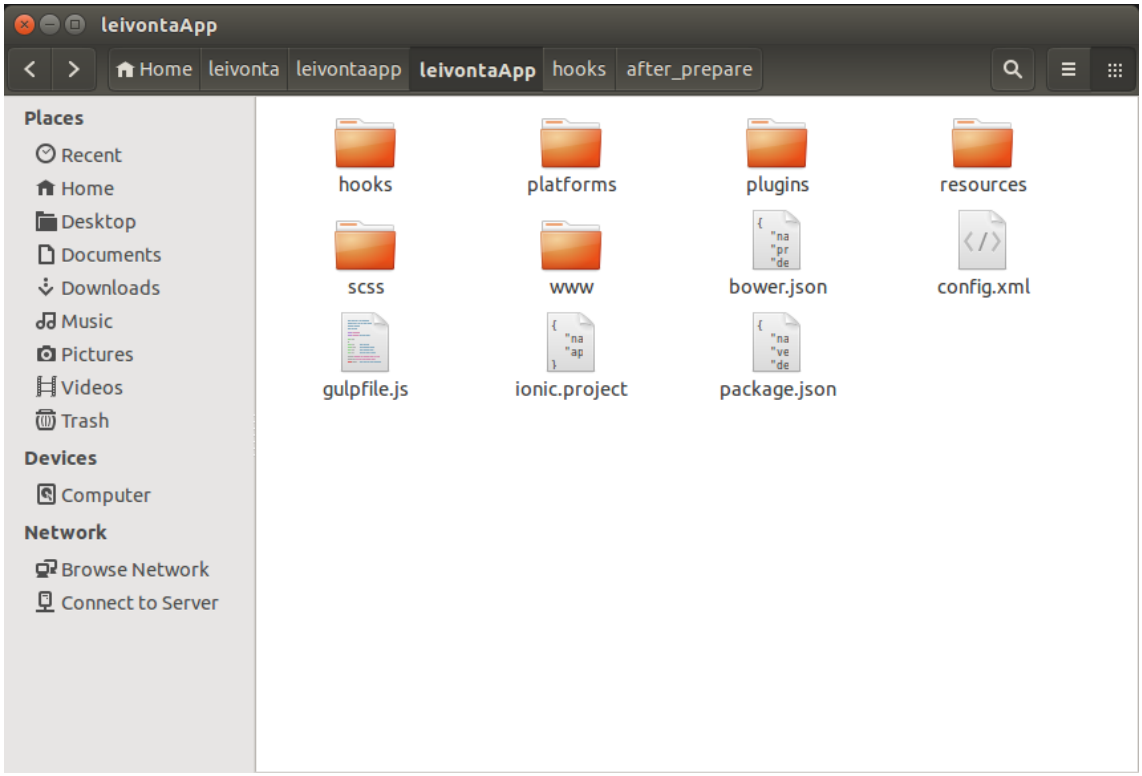
Kuva 26. Filtteriesimerkki HTML-koodissa

7 Hybridimobiilisovellustoteutus

Suunnitelman laatimisen jälkeen oli helppo lähteä toteuttamaan suunnitelmanmukaista sovellusta, koska kehitystyökalut olivat ennestään tuttuja. Tutustuin työssä käytettyyn Ionic-sovelluskehykseen ja siihen liittyviin tekniikoihin jo aikaisemmasta projektista, jonka piti tulla insinöörityön käytännön toteutukseksi. Sovelluksen tarkoitus on tarjota työkalu ainemäärien määrittämiseksi leivonnasta kiinnostuneille henkilöille. Sovellus tarjoaa myös monipuolisen valikoiman ohjeita eri kakkupohjien ja täytteiden valmistamiseen. Tämä luku käsittelee tarkemmin kehityksen tulosta.

7.1 Tiedostorakenne

Kuva 27 näyttää sovelluksen tiedostorakenteen. Tiedostorakenne on luotu Ionic CLI-komennon kautta. Oleellisimmat osat peruskehitystyön osalta ovat www-kansio, resources-kansio ja config.xml-tiedosto.



Kuva 27. Sovelluksen tiedostorakenne

Www-kansio sisältää koko web-sovelluksen, eli käytännössä koko kehitettävän sovelluksen. Tärkein osa www-kansiossa on index.html, joka toimii sovelluksen ns. main-funktiona, joka kertoo, mitkä kaikki tiedostot sovelluksen pitää ladata toimiakseen. Www-kansio kannattaa jäsennellä huolellisesti, koska sovelluksen kasvaessa on hyvä tietää, missä kansiossa on mitäkin sovelluksen lähdekoodia.

Resources-kansiossa on sovelluksen käyttämä ikoni, joka näkyy laitteelle asennuksen jälkeen laitteen sovelluslistassa ja käyttöliittymässä. Kansio sisältää myös ns. splash-kuvan, joka näkyy sovelluksen käynnistyessä config.xml-tiedostossa määritellyn ajan.

Config.xml-tiedosto on Apache Cordovasta tuleva osa, joka listaa kaikki sovelluksen määritykset. Tärkeimpinä tietoina on sovelluksen id ja versio. Ilman id tietoa sovellusta ei voi julkaista missään sovelluskaupassa. Versio on sen takia tärkeä olla, että sovellusta päivittäessä versio numeron pitää olla muuttunut. Muuten päivitystä ei hyväksytä sovelluskaupoissa.

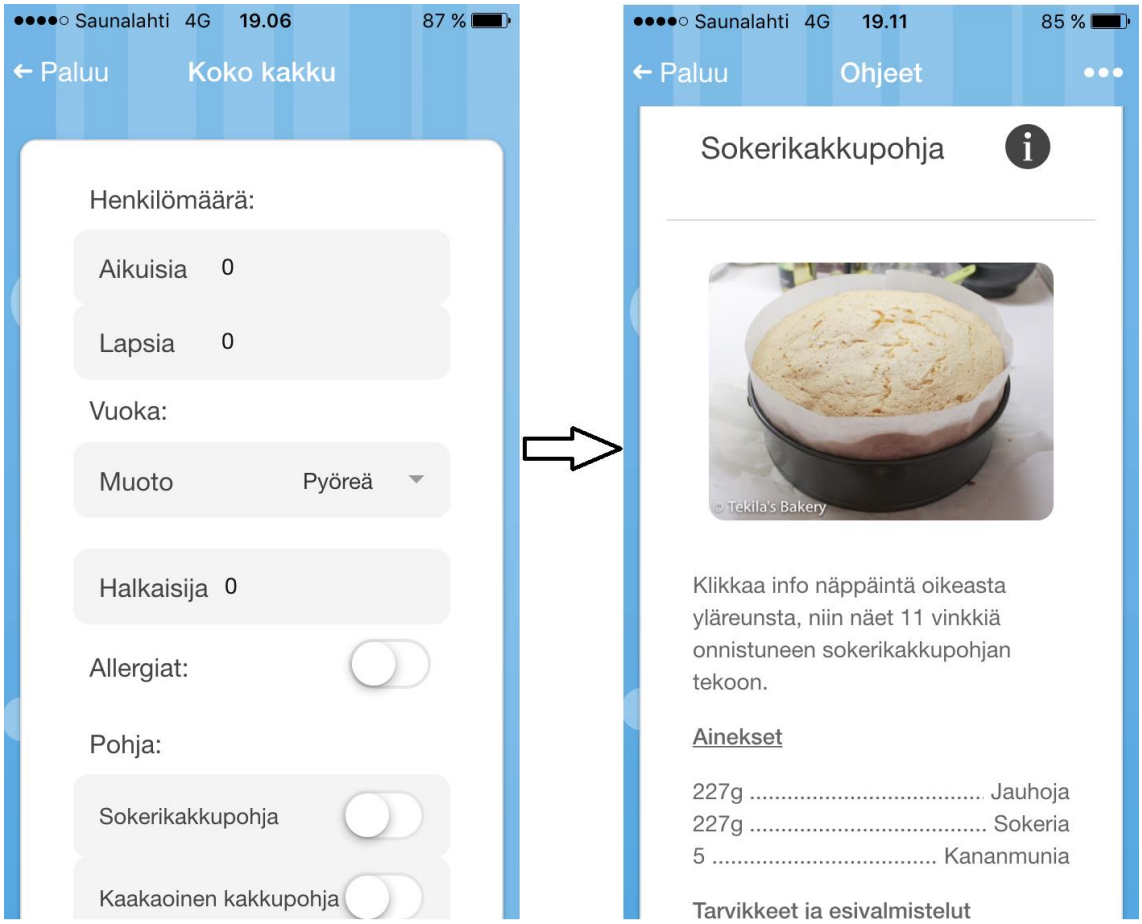
7.2 Näkymäkerros

Kuvan 28 ruudunkaappaus on aloitusnäkymä. Aloitusnäkymän on suunniteltu helposti lähestyttäväksi, joten näkymän asetteluksi valittiin yleisimpien mobiilialustojen käyttämää ikonityyliä jäljittelevä ratkaisu.

Kuva 28. Ruudunkaappaus aloitusnäkymästä

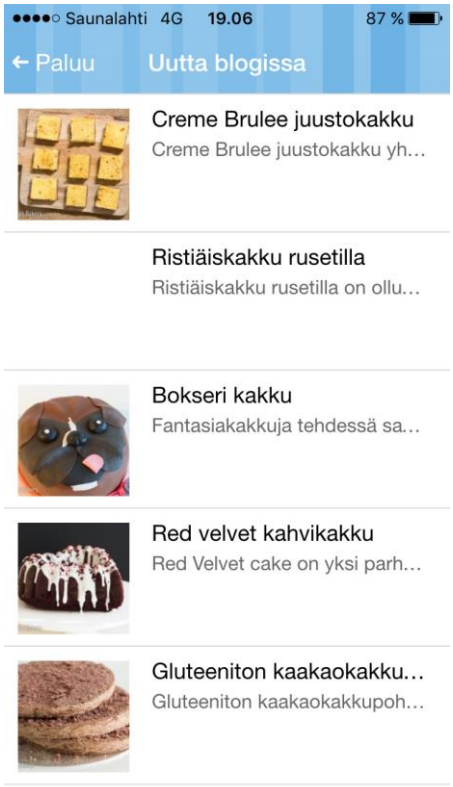
Sovellus koostuu seitsemästä päänäkymästä ja kolmella päänäkymistä on alinäkymä. Siirtyminen päänäkymästä toiseen vaatii aina kulkemisen aloitusnäkymän kautta. Kuva 28 on ruudunkaappaus sovelluksen aloitusnäkymästä. Jokaiselle näkymällä on määritelty oma ohjain. Sovelluksen kontrollereista lisää seuraavassa luvussa.

Kuvan 29 vasemmalla puolella on aloitusnäkymän ”koko kakku”-painikkeen avaama kakun suunnittelu näkymä. Näkymä koostuu erilaisista syötekentistä, joiden avulla sovellus osaa laskea ohjenäkymässä olevan kakun valmistusohjeen ja aineiden määrän. Ruudunkaappaus ohjenäkymästä on kuvan 29 oikealla puolella.



Kuva 29. Kakun suunnittelunäkymä ja ohjenäkymä

Päänäkymistä kakkupohjanäkymä ja täytenäkymä ovat idealtaan täysin samat kuin ”koko kakku” -näkymä. ”blogi” -painikkeella avautuva näkymä edustaa sovelluksen toista käyttötarkoitusta. Bloginäkymässä listataan kymmenen uusinta blogipostausta ulkopuoliselta palvelimelta. Ruudunkaappaus bloginäkymästä on kuvassa 30.



Kuva 30. Bloginäkymä

Kaikki näkymät ovat HTML-sivuja, joista löytyy Ionic CSS -komponentteja ja Angular-direktiivejä. Aloitusnäkymän kaikki painikkeet ovat <img> -komponentteja, joihin on lisätty Ionicin tarjoama on-tap-direktiivi. Painikkeiden asettelu on hoidettu Ionicin ruuduk-kosysteemillä. Animointi on tehty CSS-animaatiolla. Esimerkiksi painikkeiden liukuminen ruudulle käyttää kuvassa 22 esiteltyä slideIn -animaatiota. Kuvaan 31 on leikattu yhden painikerivin luonti HTML-koodilla.

```
<div class=”row row-center”>
 <div class=”col text-center”>
  <img ng-src=”img/CakeCrust.png” class=”buttonSize” on-tap=”nextPage(‘cakeOnly’)/>
  <p>Kakkupohjat</p>
 </div>
 <div class=”col text-center”>
  <img ng-src=”img/NakedCake.png” class=”buttonSize” on-tap=”nextPage(‘wholeCake’)/>
  <p>Kakkupohjat</p>
 </div>
</div>
```

Kuva 31. Painikerivin HTML-esimerkki

7.3 Ohjainkerros

Edellisessä luvussa mainittiin, että jokaiselle näkymälle on luotu oma ohjain. Työkalunäkymissä, eli ”koko kakku”-näkymä, täytenäkymä ja pohjanäkymä, ohjaimen päätehtävänä on huolehtia käyttäjän antamien syötteiden validoinnista ja päivittää sovelluksen palvelussa olevaa kakkutietuetta vastaamaan annettuja arvoja. Kakkutietue on palvelussa, koska sitä samaa tietoa tarvitaan kahdessa eri ohjaimessa, joten palvelu on helppo tapa pitää tieto saatavilla molemmissa ohjaimissa. Typistetty versio sovelluksen käyttämästä kakkutietuepalvelusta koodimuodossa on kuvassa 32. Palvelu ilmoitetaan ohjaimen riippuvuusluettelossa. Funktio getData palauttaa viitteen palvelussa sijaitsevaan cakeData-nimiseen objektiin.

```
//palvelu löytyy sovelluksen päämoduulista
.service('DataStore', function() {
        this.cakeData = {
                persons: {
                        adults:0,
                        chidren:0,
                        total:0
                },
                panSize:0
        };
        this.getData = function() {
                return this.cakeData;
        }
});
```

Kuva 32. Koodiesimerkki palvelusta

Kaikkien kolmen työkalun ohjealinäkymissä ohjaimen tehtävänä on hakea palveluun talletettu kakkutietue ja muokata se haluttuun muotoon. Ohjenäkymät sisältävät myös lisäominaisuuksia, jotka aukeavat näkymän oikeassa ylälaidassa olevalla painikkeella. Ohjain hoitaa oikeat Ionic-modaalit ja ponnahdusikkunat oikeaan aikaan näkyville. Koodiesimerkki ponnahdusikkunan avaamisesta on kuvassa 33.

```
//HTML
<ion-nav-buttons side="right">
   <button class="button button-clear button-dark button-icon icon ion-more"
    ng-click="showPopover($event)"></button>
</ion-nav-buttons>

//JavaScript
//popOverin alustus käyttäen valmista html-tiedostoa mallina
$ionicPopover.fromTemplateUrl('templates/popoverTemp.html', {
                        scope: $scope
                }).then(function(popover) {
                        $scope.popover = popover;
                })
//showPopover function toteutus
$scope.showPopover = function($event) {
        $scope.popover.show($event);
}
```

Kuva 33. Esimerkki ponnahdusikkunan avaamisesta

Jokaisen ohjenäkymän lisäominaisuuksista löytyy lähetä-toiminto. Toiminto kokoaa ohjeen yhdeksi HTML-malliksi merkkijonomuodossa. Alustan oletussähköpostiohjelman avaamiseen käytetään Apache Cordovan ”email composer” -lisäosaa. Lisäosa tarjoaa JavaScript-koodissa käytettävän ”cordova.plugins.email” -palvelun. Palvelulla on open-funktio, joka saa parametrina objektin, joka sisältää mm. kenelle viesti lähetetään, laitetaanko viestiin liitteitä ja onko viesti tekstiä vai HTML-syntaksia. Sovelluksen kannalta oleellisimmat objektin kentät ovat viestin aihe ja sisältö. Aiheeksi liitetään kakkureseptin nimi ja sisällöksi liitetään aiemmin koottu HTML-malli. Kuva 34 havainnollistaa sähköpostin avaamisen koodimuodossa.

```
this.openEmail = funtion(header, bodyText) {
   cordova.plugins.email.isAvailable(function(isAvailable) {
      if(isAvailable) {
         var email = { isHtml:true}
         email.subject = header;
         email.body = bodyText;
         cordova.plugins.email.open(email);
      }
   }
}
```

Kuva 34. Esimerkki sähköpostiohjelman avaamisesta

Bloginäkymään listautuvat tiedot haetaan kokonaisuudessaan ulkopuoliselta palvelimelta. Sovelluksen käyttämällä palvelimella pyörii WordPress -julkaisualusta. WordPressiin on asennettu kolmannen osapuolen tarjoama lisäosa, joka mahdollistaa palvelimen käytön samanaikaisesti sekä sisällönhallintaohjelmistona että REST-rajapintana. Tiedon hakemiseen ulkopuoliselta palvelimelta käytetään AngularJS:n tarjoamaa http-palvelua. Palvelin palauttaa onnistuneessa kutsussa response-objektin, joka sovelluksessa kootaan haluttuun muotoon.

Jokainen listalla oleva komponentti toimii linkkinä määrättyyn blogipostaukseen. Käyttäjän painettua linkkiä kutsutaan funktiota, joka saa parametrina postauksen URL-osoitteen. Funktion toteutuksessa tarkistetaan, millä alustalla käyttäjä on, ja sen mukaan kutsutaan haluttua Cordovan tarjoamaa rajapintaa avaamaan URL-osoitteen alustan oletusselaimella. Kuvassa 35 on edellä mainitun funktion toteutus.

```
$scope.viewFeed = function(url) {
        if(ionic.Platform.isAndroid()) {
                window.open(url, '_blank');
        } else {
                window.open(url, '_system');
        }
}
```

Kuva 35. URL-osoitteen avaamiseen tarkoitettu funktio.

Kuvan 35 funktiossa tarkastetaan, onko käytössä oleva alusta Android. Jos alusta on Android kutsutaan window.open -funktiota parametreilla URL-osoite ja ”_blank”. Parametri ”_blank” tarkoittaa, että sovellus avaa itsensä sisään uuden verkkoselainnäkymän ja näyttää siinä kyseisen URL-osoitteen. Alustan ollessa jokin muu kuin Android, eli käytännössä iOS. Tässä tapauksessa annetaan ”_blank”:in tilalle parametri ”_system”, jolloin URL-osoite avataan uudessa oletusselain ikkunassa.

8 Yhteenveto

Insinöörityön toteuttaminen aloitettiin suunnitelman laatimisella. Piti kartoittaa, millaiselle sovellukselle olisi käyttöä ja miten sovellus toteutettaisiin. Tekniikan valinta oli haastavaa, koska sovelluksen kohderyhmä on verrattain pieni, mutta silti haluttiin tukea yleisimpiä mobiilialustoja, eli Android ja iOS:ää. Kokemattomuus natiivista mobiilisovelluskehityksestä yhdessä aikataulun kanssa rajasi pois vaihtoehdon, että kehitetään molemmille alustoille natiivisovellus. Hybridisovelluskehitys osoittautui jo aiemmassa projektissa varteenotettavaksi vaihtoehdoksi natiivinsovelluksen korvaajaksi, joten kehitystavaksi valittiin hybridikehitys.

Työ kehitettiin Ionic-sovelluskehyksellä. Ionic tarjoaa kattavan valikoiman erilaisia käyttöliittymäkomponentteja. Ionic on kehitetty Apache Cordovan päälle, mikä mahdollistaa alustan natiivejen palveluiden, kuten kamera ja yhteystiedot, käytön hybridisovelluksessa. Ionic valittiin sovelluskehykseksi monesta eri syystä, mutta suurimmat syyt olivat Ionicin kattava dokumentaatio ja aikaisempi kokemus Ionicin käytöstä.

Sovelluksen kehitysprosessi opetti paljon uutta ja päivitti aikaisempaa tietoa tekniikoista. Hybridisovelluskehitys on osittain verkkosivukehitystä, eli käytetään paljon CSS- ja HTML-tekniikkaa. Aikaisempaa kokemusta CSS- ja HTML-kehityksestä ei juuri ole, joten projektissa oppi paljon uutta kyseisistä tekniikoista. AngularJS:n tarjoamien rakenteiden käyttö kehittyi. Varsinkin ohjainten käyttöön tuli paljon uusia ideoita projektin edetessä. Kehitysprosessista puuttui kokonaan automaatiotestaus, mikä oli koko projektin suurin virhe. Projektin laajentuessa olisi kehittämisen kannalta tärkeää, että olisi työkalut millä todeta nopeasti toimiiko kaikki vanha vielä uuden lisäämisen jälkeen. AngularJS tukee hyvin testattavan koodin kirjoittamista.

Insinöörityön kirjoitusvaiheessa kehitettävässä sovelluksessa oli jo kaikki suunniteltu toiminallisuus toteutettu testausta vaille valmiiksi. Ohjelmistoalalla valitseva muutos on siirtyä vesiputousmallisista projekteista ketterämpiin projektimalleihin, joten tässäkin projektissa on jatkettu kehitystä samana projektina, koska lopputulokseen ei olla vielä oltu tyytyväisiä.

Lähteet

1 Vuorinen Carl. Kolme tapaa kehittää mobiilisovellus. Verkkodokumentti. http://www.w3.fi/kolme-tapaa-kehittaa-mobiilisovellus/ Luettu 15.4.2016.

2 Wilken Jeremy. Ionic in Action. 2015. Manning Publications Co.

3 Bristowe John. What is a Hybrid Mobile App. Verkkodokumentti. http://developer.telerik.com/featured/what-is-a-hybrid-mobile-app/ Luettu 17.4.2016.

4 Nimimerkki Globetrotter. Hybrid applications and android native browser. Verkkodokumentti. https://myshadesofgray.wordpress.com/2014/04/15/hybrid-applications-and-android-native-browser/ Luettu 17.4.2016.

5 Krzysztof Marszałek. Native vs Hybrid - Demystifying the Technology Dilemma. Verkkodokumentti. http://howwedostartups.com/articles/Native-vs-Hybrid Luettu 17.4.2016.

6 Apache Cordova docs. Verkkodokumentti. https://cordova.apache.org/docs/en/latest/guide/overview/ Luettu 17.4.2016.

7 Panhale Mahesh. Beginning Hybrid Mobile Application Development. 2016. Apress.

8 Whetton Matt. PhoneGap Fundementals: What you need to know. Verkkodokumentti. http://www.codenutz.com/what-you-need-to-know-about-phonegap/ Luettu 18.4.2016.

9 Osmani Addy. Learning JavaScript design patterns. 2012. O'Reilly Media.

10 Lehtinen Antti. Diplomityö. Verkkodokumentti. http://dspace.cc.tut.fi/dpub/bitstream/handle/123456789/20983/lehtinen.pdf?sequence=3&isAllowed=y Luettu 18.4.2016.

11 Bootstrap docs. Verkkodokumentti. http://getbootstrap.com/ Luettu 18.4.2016.

12 Sencha Touch docs. Verkkodokumentti. https://www.sencha.com/products/touch/#overview Luettu 18.4.2016.

13 Apache Cordova docs Reference. Vekkodokumentti. https://cordova.apache.org/docs/en/latest/config_ref/index.html#sample-configxml luettu 18.4.2016.

14 Camden Raymond K. Apache Cordova in Action. 2015. Manning Publications Co.

15 Seshadri Shyam, Green Brad. AngularJS: Up and Running. 2014. O'Reilly Media.

16 Ravulavaru Arvind. Learning Ionic. 2015. Packt Publishing.

17 Ionic kotisivu. Verkkodokumentti. http://ionic.io/ luettu 21.4.2016.

18 Ionic.io docs. Verkkodokumentti. http://docs.ionic.io/ luettu 21.4.2016.

19 JavaScript promise kalvosetti. Verkkodokumentti. www.cs.tut.fi/~seitti/2015/kalvot/promise/ luettu 22.4.2016.

20 Ionic framework docs. Verkkodokumentti. http://ionicframework.com/docs/overview/ luettu 25.4.2016.

21 CSS wikipedia. Verkkodokumentti. https://fi.wikipedia.org/wiki/Cascading_Style_Sheets luettu 25.4.2016.

22 Html5 wikipedia. Verkkodokumentti. https://fi.wikipedia.org/wiki/HTML5 luettu 26.4.2016.

23 McGinnis Tyler. AngularJS: Factory vs Service vs Provider. Verkkodokumentti. http://tylermcginnis.com/angularjs-factory-vs-service-vs-provider/ luettu 27.4.2016.