Erik Rantanen

Iphone-sovelluksen kehittäminen Ionic-sovellus-kehyksellä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

28.11.2016

Metropolia

Tekijä Otsikko Sivumäärä Aika	Erik Rantanen Iphone-sovelluksen kehittäminen Ionic-sovelluskehyksellä 33 sivua 28.11.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaajat	Lehtori Ilpo Kuivanen Aino Salo

Tämän insinöörityön tarkoituksena oli toteuttaa asiakkaalle Villiruoka-aiheinen iPhone-sovellus. Asiakas tarjosi sovellusta varten Photoshopilla tehdyn staattisen suunnitelman, jolloin kehittäjän tehtäväksi jäi sovelluksen tekninen rakentaminen sekä animaatioiden toteutus. Sovellus päädyttiin toteuttamaan hybridisovelluksena käyttäen Ionic-sovelluskehystä suunniteltujen ominaisuuksien teknisen yksinkertaisuuden ja kehityksen nopeuden vuoksi.

Työssä esiteltiin erillaisia tapoja toteuttaa mobiilisovelluksia, niiden hyviä ja huonoja puolia sekä milloin niiden käyttö on järkevintä. Tämän jälkeen keskityttiin kuvailemaan sovelluksen toteutuksessa käytettyä Ionic-sovelluskehystä ja sen riippuvuuksia: Cordovaa ja Webviewiä ja lopuksi Ionicin käyttämiä web-teknologioita, joista tärkein oli Angular.

Työn käytännön osassa käytiin läpi sovelluksen vaatimukset ja perusteltiin niiden pohjalta Ionic-sovelluskehyksen olleen paras vaihtoehto kehittämistä varten. Kehitystyöstä käytiin läpi sen aloitus sekä sen aikana nousseet Ionicin hyvät ja huonot puolet. Hyvinä puolina nousivat kehityksen helppous ja nopeus. Huonoina todettiin ulkonäön muokkaamisen vaikeus vaativammissa tarpeissa sekä hybridisovellusten venymättömyys osaan iOS-alustan komponenteista.

Avainsanat	Ionic, Angular, Javascript, Cordova, hybridisovellus

Author Title Number of Pages Date	Erik Rantanen Development of iPhone Application with Ionic Framework 33 pages 28 November 2016
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructors	Ilpo Kuivanen, Senior Lecturer Aino Salo

The purpose of this thesis was to develop a 'Villiruoka' iPhone application to a customer. The customer provided a static graphical plan made with Photoshop for the application, which left the creation of the technical implementation and animations to the developer. It was decided that the application would be made as a hybrid application using the Ionic framework due to the technical simplicity of the application and the speed of the development with Ionic.

The thesis introduces a few different ways to develop mobile applications, their pros and cons and when it is sensible to use each of them. The study also introduces the Ionic framework, its dependencies, Cordova and WebView and the web technologies it uses, of which the most important one is Angular.

In the practical part of the thesis the requirements of the application are described and based on them, the use of the Ionic framework is justified as the best possible option for the application. The study describes early stages of the development process and the good and bad qualities found out during the Ionic development are introduced. Fast and simple development was seen as a benefit, whereas the difficulty of making demanding edits to the style and the inability to create some iOS-components were considered to be drawbacks.

Keywords	Ionic, Angular, Javascript, Cordova, hybrid application

Sisällys

Lyhenteet

HTML5	Hypertext Markup Language 5. Merkintäkieli, jota käytetään kuvaamaan internetsivun rakenne.
CSS	Cascading Styles Sheet. Tyylitiedosto, jolla voidaan kuvata HTML-elementtien ulkonäköä.
SDK	Software Development Kit. Kehitysympäristö, jonka avulla voidaan tietynlaisia sovelluksia.
SASS	Syntactically Awesome Style Sheets. Uudempi tyylitiedoston vaihtoehto CSS:lle, joka antaa mm. mahdollisuuden määritellä muuttujia.
MVC	Model, View ja Controller. Suunnittelumalli sovelluksille, jossa koodi jaetaan kolmeen osaan selkeyden vuoksi. Model toimii tietomallina, esimerkiksi listana kasveja, View toimii näkymänä, johon kaikki käyttäjälle näkyvä piirretään ja jossa Controller ohjaa sovelluksen toimintaa.
DOM	Document Object Model. Ohjelmointirajapinta, joka kuvaa HTML-, XHTML- tai XML-tiedoston sisällön puurakenteena, jossa jokainen sen osa toimii sivun osaa edustavana objektina, joita voidaan muokata Javascriptin avulla.
URL	Uniform Resource Locator. Osoite, jolla viitataan verkossa olevaan palveluun, kuten verkkosivuun tai tietokantaan.
JSON	JavaScript Object Notation. Ihmisen luettavissa oleva tiedosto, joka säilyttää tietoa avain- ja arvopareilla.

1 Johdanto

Insinöörityö tehtiin asiakkaalle, joka halusi hänen Photoshopilla tehdystä suunnitelmasta Iphonella toimivan sovelluksen. Sovelluksen oli tarkoitus olla kokoelma Suomen luonnosta löytyvistä kasveista, joita pystyy käyttämään ruokana tai mausteina. Kasvikokoelmaa oli toteutuksessa tarkoitus pystyä filtteröimään muutamilla eri tavoilla sekä hakemaan niitä nimen perusteella. Käytännössä asiakas tarjosi suunnitelman sovelluksen ulkonäölle sekä kaikkien sovellukseen tulevien kasvien tekstit, joten kehittäjän harteille jäi toiminnallisuuksien rakentaminen sekä suunnitelman mukaisen ulkonäön toteuttaminen.

Sovellusta lähdettiin toteuttamaan Ionic-sovelluskehyksellä, jonka avulla pystytään rakentamaan älypuhelinsovelluksia web-teknologioita käyttäen. Kyseisen sovelluskehyksen avulla voidaan toteuttaa niin sanottuja hybridisovelluksia, joiden rakentaminen on tavallisia älypuhelinsovelluksia nopeampaa, mutta mahdollisuudet käyttää puhelimien omia toimintoja ovat pienemmät.

Tämän insinöörityön tavoitteena on kuvata sovelluksen kehittämisprosessia ja kertoa sen aikana ilmenneistä hyvistä ja huonoista puolista Ionic-sovelluskehyksessä. Lisäksi työssä annetaan lukijalle kuva tilanteista, joissa hybridisovelluksen kehittäminen on varteenotettava vaihtoehto. Insinöörityö aloitetaan kuvailemalla käytettyjä teknologioita hybridisovellusten käsitteestä käytettyihin web-teknologioihin asti ja sen jälkeen kerrotaan itse sovelluksen kehitysprosessista.

2 Mobiilisovellukset

Mobiilisovellukset ovat älypuhelimella tai tabletilla ajettavia sovelluksia, jotka tekevät jotain tiettyä asiaa, kuten näyttävät esimerkiksi säätiedon tai joiden kautta pystyy puhumaan toisten sovelluksen käyttäjien kanssa. Niitä ladataan älypuhelimeen useimmiten alustan mukaisesta sovellusverkkokaupasta, kuten Iphoneilla AppStoresta ja Androideilla Play Storesta. Vuoden 2008 jälkeen niiden kysyntä on kasvanut räjähdysmäisesti ja sen takia on kehitetty useita eri tapoja niiden rakentamista varten. (1.)

Käytännössä mobiililaitteilla ajettavia sovelluksia on mahdollista tehdä kolmella eri tavalla: natiiveinasovelluksina, verkkosovelluksina ja hybridisovelluksina. Jokainen kolmesta vaihtoehdosta tarjoaa omat vahvuutensa ja heikkoutensa, jotka täytyy ottaa huomioon sovellusta suunniteltaessa.

2.1 Vaihtoehdot mobiilisovellukselle

Natiivit sovellukset on suunniteltu ja koodattu tietyn tyyppiselle laitteelle kuten Androidille, Iphonelle tai Windows Phonelle. Tällöin on siis käytettävä laitteen vaatimaa koodikieltä: Androidille koodataan Javalla, Iphonelle Objective-C:llä tai Swiftillä ja Windows Phonella C#. Koska ne ovat alusta lähtien suunniteltu tietylle laitteelle, ne toimivat useimmiten muilla tavoilla tehtyjä nopeammin, luotettavammin ja ne pystyvät käyttämään puhelimen toimintoja kuten kameraa ja kiihtyvyysanturia paljon laajemmin. Niiden heikkoutena on sitoutuminen tiettyyn alustaan: jos natiivin Android-sovelluksen haluaa Iphonelle, koodi on kirjoitettava uudestaan toisella kielellä. (2; 3.) Lisäksi kehittäminen on useimmiten hitaampaa, koska testaamista varten koodi on rakennettava uudelleen muutoksien jälkeen, mikä vie pidemmissä projekteissa paljon aikaa.

Verkkosovellukset ovat käytännössä verkkosivuja, joita käyttäjä ajaa puhelimen internetselaimessa. Ne koodataan usein jollakin yhdistelmällä lukuisia eri web-kieliä, joihin kuuluvat vähintään HTML5, CSS ja Javascript. Verkkosovelluksia ei kuitenkaan voi julkaista alustan sovelluskaupassa, siis Iphonen Appstoressa tai Androidin Play Storessa, koska pelkkiä verkkosivuja ei niihin sallita. Mitään pääsyä niillä ei kuitenkaan ole puhelimen omille toiminnoille eivätkä ne yleensä toimi yhtä sulavasti kuin natiivit sovellukset. Etuna niissä kuitenkin on niiden toimivuus lähes kaikissa laitteissa, joista löytyy internetselain. (3.)

Hybridisovellukset ovat yhdistelmä natiivi- ja verkkosovellusten ominaisuuksia sekä samalla yhdistelmä niiden hyviä ja huonoja puolia. Kuten verkkosovellukset, hybridisovellukset rakennetaan käyttämällä web-kieliä, mutta niistä syntyvä verkkosivu ajetaan puhelimen WebView-komponentissa, ja lopuksi se paketoidaan puhelimessa ajettavaksi sovellukseksi. Paketointi voidaan tehdä ainakin Apachen Cordovalla Iphonelle tai Android-laitteille, joten teoriassa samaa koodia voidaan käyttää kummallakin alustalla. Lisäksi sovelluksen testaaminen on mahdollista selaimessa, mikä on nopeampaa kuin suoraan laitteella testaaminen. Laitteellakin on testattava ajoittain, mutta sillä testaamista ei tarvitse tehdä niin usein kuin natiiveja sovelluksia kehitettäessä. Heikkoutena ne perivät verkkosovelluksien huonomman toimivuuden verrattuna natiiveihin sekä kaikkia laitetason ominaisuuksia ei myöskään saada käyttöön. Nämä kaksi heikkoutta käytännössä sulkevat mahdollisuuden toteuttaa tietynlaisia sovelluksia, kuten pelejä.(2; 3.)

2.2 Sopivin valinta

Suunniteltavan mobiilisovelluksen ominaisuudet on kartoitettava ennen kuin valintaa natiivin, hybridisovelluksen ja verkkosovelluksen välillä voidaan tehdä. Lisäksi suunnittelussa on huomioitava mobiilisovelluksen ennakoitu kohdeyleisö, kehitystiimin taitotaso sekä projektin aikataulu.

2.2.1 Natiivi vs. verkkosovellukset

Ensimmäisenä on hyvä tehdä valinta kahden ääripään välillä: halutaanko sovellusta jakaa mobiililaitteen sovelluskaupassa? Jos halutaan, verkkosovellus sulkeutuu pois vaihtoehdoista. Puhelimen selaimessa pyörivää verkkosovellusta on mahdollista harkita, jos sen tulevat käyttäjät suosivat puhelimen selainta palvelua käyttääkseen. Usein esimerkiksi uutissivustoista tehdään älypuhelimien selaimia varten oma versio, koska käyttäjäkunta on tottunut käyttämään sivua pöytäkoneellaan. Sovellusten käyttö on kuitenkin edelleen kasvussa ja ihmiset ovat myös valmiita maksamaan niistä. Vuoden 2016 alussa Applen lausunnon mukaan ihmiset olivat käyttäneet 40 miljardia dollaria sovelluksiin (4), joten sovellusmarkkinoita voidaan pitää houkuttelevina. Jos sovellukselle ei kuitenkaan nähdä sopivaa markkinarakoa alustan sovelluskaupassa ja halutaan parantaa jonkin olemassaolevan verkkosivun käytettävyyttä mobiililaitteilla, verkkosovellus on sopiva vaihtoehto.

2.2.2 Natiivi vs. hybridisovellus

Valinta natiivin ja hybridisovelluksen välillä on ensisijaisesti riippuvainen sovellukseen kaavaillusta ominaisuuksista. Käytännössä hybridisovellukset voidaan sulkea pois, jos kaavaillaan paljon graafista laskentaa vaativaa peliä. Vaikka hybridisovelluksella on mahdollista toteuttaa peli esimerkiksi HTML5:n Canvas-elementillä, saattaa sen laaja käyttö toimia laitteella hitaasti, koska kaikkia laitteen tehoja ei saada sovelluksen käyttöön. Jos tulevaisuudessa web-teknologioiden suorituskyky paranee, hybridisovellukset voisivat nousta varteenotettavaksi vaihtoehdoksi myös pelien kanssa. Jos taas kaavaillut ominaisuudet ovat suhteellisen yksinkertaisia, kuten tiedon hakemista, näyttämistä ja muokkaamista, hybridisovellus vaatii paljon vähemmän työtä ja tulos voi parhaimmillaan olla yhtä näyttävä kuin natiivilla. (2; 5.)

Hybridisovelluksien etuna on myös mahdollisuus käyttää samaa koodia usealla alustalla. Tällöin useammalle alustalle julkaisu on halvempaa, koska projektiin ei tarvitse palkata sekä iOS että Android alustojen osaajia, joista molemmat ovat jo valmiiksi hyvin kalliita palkata. Hybridisovelluksia voi kehittää eteenpäin pelkästään web-teknologioiden osaamisella, joten hybridisovellusten kehitys on halvempaa. Android- ja iOS- järjestelmillä on kuitenkin omat eronsa käytettävyydessä ja tuntumassa, joita on vaikea jäljitellä Hybridisovellusten käyttämässä WebView'ssä. Niille alusta lähtien suunniteltu natiivi koodi tarjoaa ilman jäljittelyä alustalle ominaisen tuntuman ja laajemman pääsyn laitteen ominaisuuksiin. Viimeisenä vertailukohtana natiivien sovelluksien kehittäminen on hitaampaa kuin hybridisovelluksien kehittäminen osaavasta tiimistä huolimatta, koska natiivit sovellukset on rakennettava kehitysympäristössä puhelimella ajoa varten aina, kun uusia ominaisuuksia testataan, mikä vie pitemmissä projekteissa paljon työaikaa. Hybridisovelluksia täytyy myös testata laitteilla, mutta niitä voidaan myös testata ensin selaimessa, johon muutokset tulevat näkyviin heti. Tämä nopeuttaa esimerkiksi sovelluksen sisäisen logiikan varmistamista oikeaksi. (2; 5.)

Oman kiinnostavan poikkeuksen natiivi ja hybridisovelluksille tarjoaa React Native, jonka tavallisesti verkkosovelluksiin käytettävä React-pohjainen koodi ajetaan alustan Javascript-virtuaalikoneessa eli JavaScriptCoressa, jonka kautta React-koodi voi käyttää alustan natiiveja komponetteja. Tämä mahdollistaa hybridisovellusten nopean kääntämisen, natiivimman tuntuman sekä mahdollisuuden optimoida koodia myös natiivissa ympäristössä. React eroaa kuitenkin merkittävästi suurimman osan web-kielien raken-

teesta, mikä vaikeuttaa perinteiseen web-ohjelmointiin tottuneiden siirtymistä mobiilimaailmaan React Nativen avulla. Kirjoitushetkellä Facebookin mobiiliversio on tehty React Nativella. (6.)

Taulukko 1. Kolmen sovellustyypin kannattavuus eri tilanteissa

Ehto	**Kyllä**	**Ei**
Halutaanko sovellus julkaista AppStoressa tai Play Storessa?	Natiivi sovellus, hybridisovellus	Natiivi sovellus, hybridisovellus, verkkosovellus
Käyttääkö sovellus suurinta osaa laitteen ominaisuuksista?	Natiivi sovellus	Hybridisovellus, verkkosovellus
Vaatiiko sovellus paljon graafista laskentaa?	Natiivi sovellus	Natiivi sovellus, hybridisovellus, verkkosovellus

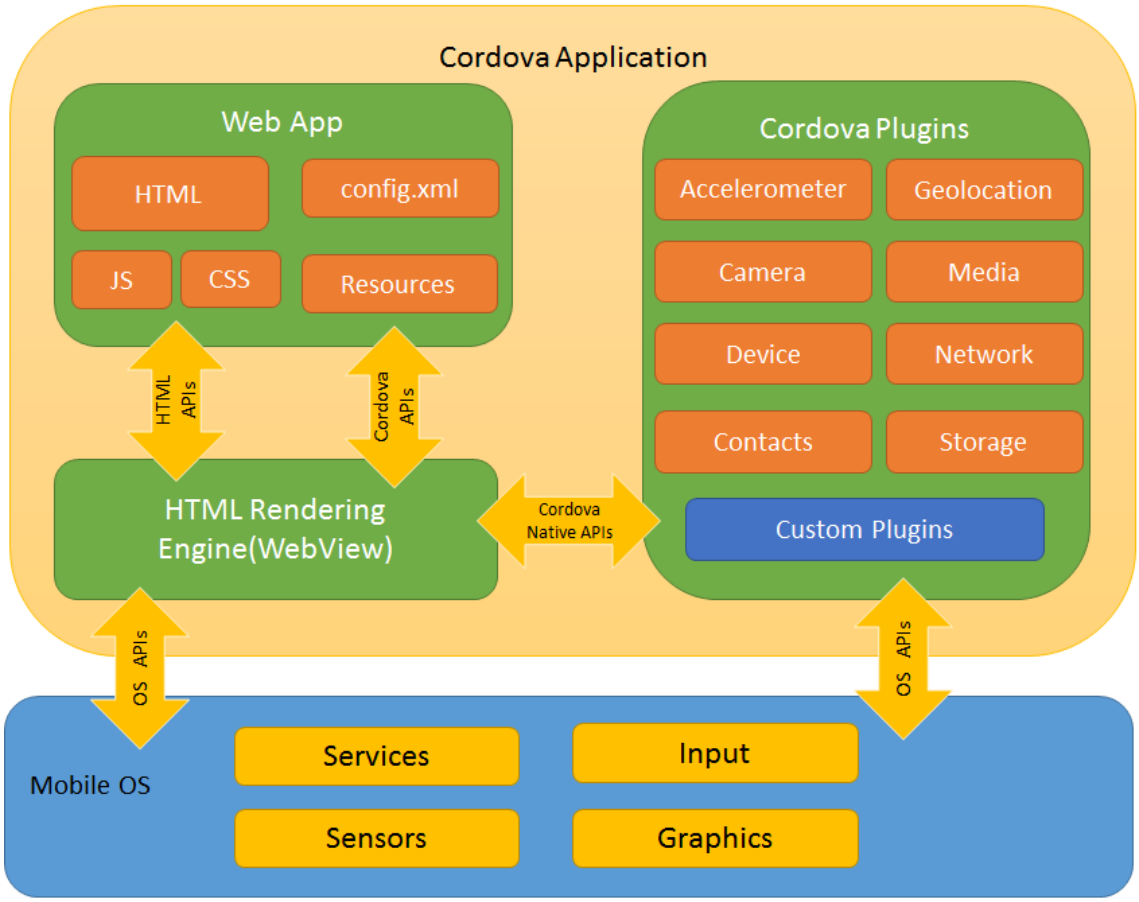
Jos siis sovelluksen tuntumasta halutaan moitteeton ja kaikki älypuhelimen ominaisuudet halutaan käyttöön, eikä tarkoitus ole säästää kehitystyössä rahaa eikä aikaa, natiivi sovellus on paras vaihtoehto. Hybridisovellusta kannattaa harkita, jos haluaa säästää kehitystyössä ja on valmis tinkimään sovelluksen tuntumasta eikä sovellus tarvitse puhelimelta paljoa tehoja. Verkkosovellusta voidaan harkita, jos valmiille internetsivulle halutaan puhelimessa toimiva versio.

3 Ionic Framework

Ionic Framework on Drifty Co:n vuonna 2012 aloittama HTML5-pohjainen mobiilisovelluskehys, joka keskittyy tarjoamaan front-end-toimintoja, eli selainpuolen toimintoja hybridisovelluksia varten. Se tarjoaa kehittäjälle työkalut sovelluksen ulkonäköön ja sisäisen logiikan työstämiselle CSS:n ja Angular.js:n avulla, mutta itsenäisesti Ionic ei pysty käyttämään puhelimen tarjoamia ominaisuuksia. Tätä varten Ionicin kehittäjät suosittelevat käyttämään Apachen Cordovaa puhelimen rajapintaa varten. Ymmärtääkseen Ionicilla tehdyn mobiilisovelluksen toimintaa, on siis myös ymmärrettävä minkä osien avulla se toimii, kuten Cordovan ja WebView'ien toimintaa. Ionicin kehitystä jatkentaan edelleen, ja tämän insinöörityön kirjoitushetkellä siitä on julkaistu versio 2.1.12. (7.)

3.1 Ionicin riippuvuudet

Internetsivut koostuvat kahdesta kerroksesta: front-endistä eli selainpuolesta ja back-endistä eli palvelinpuolesta. Front-end sisältää toiminnan, joka tapahtuu käyttäjän selaimessa, kuten esimerkiksi valikkojen piirtäminen ja syötteen lukeminen. Back-end taas sisältää tapahtumat internetsivun palvelimella, esimerkiksi saadun tilauksen kirjaamista tietokantaan. Koska hybridisovellus on puhelimen WebView'ssä toimiva internetsivu, samat front-endin ja back-endin periaatteet pätevät myös hybridisovelluksissa. Ionic tarjoaa front-endiin hybridisovellukselle natiivien sovelluksien tyyliä mukailevia graafisia komponenteja sekä mahdollisuuden sivun sisäiselle logiikalle Javascriptiin pohjautuvan Angular.js:n avulla. Back-endiksi voidaan ajatella Apachen Cordovaa, jonka avulla pystytään kutsumaan älypuhelimen toimintoja, kuten kameraa tai kontaktiluetteloa.(7; 8.)



Kuva 1. Cordovaa käyttävän hybridisovelluksen rakenne. Ionic-sovelluskehys toimii HTML Rendering Enginen sisällä. (8.)

3.1.1 Cordova

Phonegapina alunperin tunnettu Apache Cordova tarjoaa lähes koko hybridisovelluksen tarvitseman rungon kuvan 1 mukaisesti. Vain WebView-komponentin sisältö jää Ionicin toteutettavaksi. Cordovan tärkein toiminto on toimia välittäjänä puhelimen ja web-sovelluksen välillä, eli esimerkiksi välittää tieto puhelimen akun virran loppumisesta sovellukseen, jolloin web-sovellus voi esimerkiksi pyytää käyttäjää tallentamaan tilanteen. (8.)

Cordova tarjoaa sovellukselle puhelimen toimintoja pluginien avulla. Suurin osa perustoiminnoista tarjotaan ”core pluginien” muodossa, mutta Cordovan ulkopuoliset kehittäjät voivat myös tehdä omia liitännäisiään puhelimien ominaisuuksia varten, jotka eivät ole saatavilla kaikkilla alustoilla. Liitännäiset ladataan erikseen Node Package Managerilla eli npm:llä tarpeen mukaan. Npm on yhteisö ja ohjelma, jonka avulla Javascript-kehittäjät

voivat jakaa omia pakettejaan muille kehittäjille. Cordovan plugineja voidaan kutsua Webview'ssä olevasta verkkosovelluksesta Javascriptin avulla, josta siis käsky välitetään Cordovalle, joka muuntaa käskyn alustan mukaiseksi koodiksi ja kutsuu sillä puhelimen rajapintaa. Riippuen toiminnosta, ne tehdään joko kutsumalla niitä suoraan tai kuuntelemalla jatkuvasti puhelimen rajapintaa niitä varten kuten alla olevassa esimerkkikoodissa 1. (8; 9.)

```
window.addEventListener("batterylow", onBatteryLow, false);

function onBatteryLow(status) {
    alert("Vähän virtaa jäljellä! Tallenna tilanteesi.");
}
```

Esimerkkikoodi 1. Lisätään kuuntelija, joka käyttää Cordovan liitännäistä akun jäljellä olevan virran kuuntelemiseen sekä funktio, jota kutsutaan kuuntelijasta.

Toinen tärkeä Cordovan toiminto Ionicille on itse sovelluksen rakentaminen mobiilisovellukseksi. Jotta rakentaminen onnistuu, kehitysympäristössä on oltava halutun alustan vaatimat kehitysympäristöt, eli esimerkiksi Androidille Android SDK, eli Software Development Kit ja iOS:illa XCode. Applen iOS:ille kehittäessä tulee myös huomata, että ilman Macintosh-käyttöjärjestelmää on mahdotonta rakentaa sovelluksia iPhonelle. On kuitenkin mahdollista rakentaa Iphone-sovelluksia virtuaalikoneen avulla, vaikkakin virtuaalikoneelle Macintoshin asentamiseen tarvitaan Macintoshin käyttöjärjestelmän asennuslevy. Sovelluksen rakentaminen on pitemmän päälle paljon aikaa vievää, joten hybridisovelluksissa on kannattavaa testata aluksi sovelluksen perustoimivuus selaimessa ja vasta ominaisuuksia ja ulkonäköä viimeisteltäessä on kannattavaa rakentaa sovellus ja testata sitä joko emulaattorissa tai itse laitteella. (10.)

3.1.2 Webview

Ionicilla rakennettu verkkosivu näytetään käyttäjälle alustan natiivissa WebView komponentissa. Androidilla Ionicin käyttämä komponentti on nimetty samoin WebViewiksi ja iOS:illa se on UIWebView. Kyseessä on siis kuin internetselainikkuna, joka on asetettu koko ruudun täyttävään tilaan. Lopullisessa hybridisovelluksessa WebView on siis ainoa natiivi komponentti: suurin osa käyttäjälle näkyvästä toiminnallisuudesta näytetään selainikkunasta. (11.)

Webvieweillä on kuitenkin ongelmansa. Applen iOS-alustalla on UIWebView ei ole saanut Applelta täyttä tukea, koska se ei voi käyttää iOS-laitteen Safari-selainta, jolla on

käytössään nopeampi Nitro-Javascript-moottori. Tämän takia UIWebViewissä pyörivät sovellukset ovat saaneet iOS:illa heikon maineen. Uusi WKWebView iOS8-järjestelmällä tarjoaa kuitenkin uuden nopeamman vaihtoehdon. Ionic on kirjoitushetkellä ottamassa käyttöön WKWebViewiä Cordovan pluginien avulla. Androidin ongelmana on alustan hajanaisuus, minkä Androidin käyttäjät pyörittävät hyvin erillaisia WebViewejä puhelimissaan. Android 5.0 julkaisun jälkeen puhelin päivittää Webviewiä itsenäisenä osana, mikä vähentää hajanaisuutta, mutta on kuitenkin riippuvainen Androidin käyttäjien siirtymisestä Androidin viidenteen versioon. (11; 12.)

3.2 Ionicin käyttämät teknologiat

Itsenäisesti Ionicilla rakennetaan vain verkkosovellus, jota voitaisiin käyttää helposti myös tietokoneen selaimessa. Ionicin ideana on siis vain jäljitellä natiivin sovelluksen tuntumaa verkkosivujen tekoon käytettyjen työkalujen avulla. Ionic tarjoaa omia graafisia komponenttejaan, mutta niitä on myös mahdollista rakentaa itse. Itse sivu rakennetaan perinteisellä HTML5-merkintäkielellä, sen elementeille määritellään tyylit joko CSS3:lla tai uudemmalla SASS:illa eli Syntactically Awesome Style Sheetsillä ja sovelluksen ohjelmistokehyksenä käytetään Angularia sivun sisäistä logiikkaa varten. (7.)

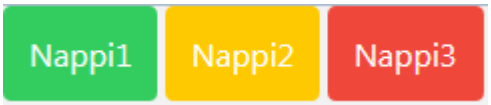
3.2.1 HTML5

HTML5 eli HyperText Markup Language on merkintäkieli, joka on viides versio HTML-standardeista, jotka jotka ovat olleet käytössä vuodesta 1993 asti. HTML5:llä kuvataan sivun sisältöä elementtien avulla, joille on mahdollista asettaa luokkia ja tunnisteita, joihin voidaan viitata tyylitiedostoissa sekä käyttää Javascriptin funktioissa. Ionic tarjoaa joukon omia elementtejä sekä luokkia HTML-elementeille, joiden avulla voidaan luoda mobiilisovelluksien tyyliä mukailevia komponentteja hybridisovellukseen. (13.)

```
<button class=”button button-balanced”> Nappi1 </button>
<button class=”button button-energized”> Nappi2 </button>
<button class=”button button-assertive”> Nappi3 </button>
```

Esimerkkikoodi 2. HTML5:llä luotu kolme nappia

Esimerkkikoodissa 2 luodaan kolme HTML-elementtiä, joille asetetaan jokaiselle oma Ionicin luokka, joille on määritelty oma tyylinsä. Kolme nappia näyttävät selaimessa kuvan 2 mukaisilta.



Kuva 2. Ionicin tuottamat kolme nappia esimerkkikoodi 2:sta

3.2.2 CSS3

CSS eli Cascading Style Sheets on HTML5-elementtien tyylin kuvaamiseen käytettävä kieli, josta CSS3 on sen tällä hetkellä laajimmin käytössä oleva versio. CSS3:a käytetään ensin kohdistamalla kuvaus kohteeseen, esimerkiksi luokan tai tunnisteen avulla, vaikkakin hyvänä käytäntönä on pidetty vain luokkien käyttämistä tyylien kohdistamisessa. Kohdistamisen jälkeen aaltosulkujen sisälle voidaan asettaa elementille esimerkiksi väri tai koko. Koodiesimerkissä 3 on katkelma kuvan 2 vihreän napin tyylikuvauksesta. (14.)

```
.button.button-balanced {
    border-color: transparent;
    background-color: #33cd5f;
    color: #fff; }
```

Koodiesmierkki 3. Kuvan 2 vihreän napin käyttämä tyylikuvaus.

Koodiesimerkissä 3 kuvataan Kuvan 2 vihreän napin ulkonäköä kolmella eri tavalla: border-color: transparent määrittää napin rajan olevan läpinäkyvä, background-color: #33cd5f kuvaa sen värin heksakoodilla, jonka tilalla voisi käyttää rgb eli Red Green Blue arvoa rgb(51, 205, 95). Viimeinen color: #fff kuvaa tekstin värin heksakoodilla valkoiseksi. Lisäksi koodiesimerkin valitsimessa .button.button-balanced on asetettu kaksi luokkaa, joista molemmat vaaditaan tyylin käyttöönottoon. Lisäksi tyyli on kohdistettu kahteen luokkaan: button ja button-balanced, jotka molemmat vaaditaan tyylin käyttöönottoon.

CSS-säännöillä on myös omat prioriteettinsa tai spesifisyytensä, jotka ovat käytössä, kun saman elementin samaa ominaisuutta kuvataan useammassa eri paikassa. Tällöin valitaan tyylikuvaus, joka on tarkin mahdollinen tai viimeisin löytyvä, jos löytyvät kuvaukset ovat yhtä tarkkoja. Tyylikuvauksen tarkkuus määräytyy osoittimien mukaan: tunnistimella saa suurimman tarkkuuden, luokkia käyttämällä toiseksi suurimman tarkkuuden ja pelkillä elementeillä pienimmän tarkkuuden. Viimeisenä tyylikuvauksen osoittimessa olevien kohteiden määrä vaikuttaa nostavasti spesifisyyteen. Lisäksi on olemassa !important-sääntö, jolla tyylikuvauksen osa voidaan pakottaa, mikä on kuitenkin pidemmän

päälle haitallista. Jos tyylikuvasta halutaan myöhemmin laajentaa, !important-sääntö on edelleen pakottamassa osaansa ja sitä on silloin vaikeaa ylikirjoittaa.

3.2.3 Angular.js

Angular.js on pääasiassa Googlen ylläpitämä verkkosovelluskehys, joka kehitettiin helpottamaan yksisivuisten verkkosovellusten rakentamista vuonna 2010. Siitä on julkaistu kaksi versiota, Angular 1 ja 2, joista Ionic 1 käyttää ensimmäistä Angularia ja uudempi Ionic 2 vastaavasti versiota kaksi. Se tarjoaa mahdollisuuden MVC-mallin käyttöön eli Model-, View- ja Controller-malliin, mikä jakaa sovelluksen toiminnan kolmeen osaan. Model toimii tietomallina, view käyttäjälle näkyvänä osana ja controller sovelluksen tapahtumien ohjaajana. Tämä tapahtuu muun moassa laajentamalla HTML:n käytössä olevia keinoja näyttää sisältöä dynaamisesti: Angular mahdollistaa näkymän ja tietomallin synkronoinnin, eli jos tietomalliin tapahtuu esimerkiksi uuden olion lisäys, muutos näkyy suoraan käyttäjälle. (15.)

Angularin tuomat direktiivit helpottavat dynaamisen sisällön näyttämistä käyttäjälle. Niitä käyttämällä HTML-tiedostoon voidaan lukea arvoja suoraan esimerkiksi tietomallista eli MVC-mallin model-osasta. Koodiesimerkissä 4 on esimerkki ng-repeat-direktiivin käytöstä. (16.)

```
<div ng-controller=”ItemController”>
  <p ng-repeat=”item in items”>
    Hello, {{item}}!
  </p>
</div>
```

Koodiesimerkki 4. Esimerkki Angularin direktiivien käyttämisestä div- ja p-elementeillä. Ng-controller määritellään koodiesimerkissä 5.

Koodiesimerkissä 4 käytetään Angularin ng-repeat-direktiiviä, jonka avulla voidaan tulostaa kaikki ”items”-taulukosta löytyvät esineet. Perinteisellä Javascriptillä tämä on jouduttu tekemään kokonaan muokkaamalla sivun elementtejä suoraan DOM:ista eli Domain Object Modelista, mutta Angularia käyttäen sivun sisällön muutokset voidaan erottaa omaksi HTML-tiedostossa kuvailtavaksi osaksi. Tämä selkeyttää koodia ja mahdollistaa rajan vetämisen MVC-mallin view- ja controller-osien välille. Jos items taulukkoon tehtäisiin mikä tahansa muutos kuten lisäys, näkyisi divin sisällä uusi tervehdys uudelle item:ille. (16.)

MVC-mallin controller-osaan eli sovelluksen sisäisen logiikan rakentamiseen Angular tarjoaa kaksi hyödyllistä työkalua: controllerit ja niihin kuuluvat scopet. Scopet ovat yhdelle controllerille kuuluvia tietynlaisia toiminnanpiirejä, joihin on mahdollista lisätä muuttujia tai funktioita. HTML-tiedostossa niihin voidaan viitata suoraan niiden nimellä koodiesimerkin 4 tavalla, jossa siis koodiesimerkissä 5 luotu controller on sijoitettu div-elementtiin, items-lista on sijoitettu itemControllerin scopeen ja koodiesimerkissä 4 sen sisältö tulostetaan. Controllerit ovat hyvien käytäntöjen mukaisesti tarkoitettu vain scopejen arvojen asettamista varten, vaikkakin Javascriptin rajattomassa maailmassa tätä käytäntöä on helppo rikkoa. (16.)

```
var myApp = angular.module('myApp',[]);

myApp.controller('ItemController', ['$scope', function($scope) {
  $scope.items = ['world', 'darkness', 'my old friend'];
}]);
```

Koodiesimerkki 5. Controllerin lisääminen koodiesimerkin 4 HTML-koodille

Ionic käyttää tavallisen Angularin lisäksi yhtä sen laajennusta: Angular-ui:n ui-routeria. Se mahdollistaa erillaisen tilojen määrittelyn verkkosovelluksille, joille kaikille voi liittää omat template-HTML-sivunsa sekä ohjaimet. Tilat otetaan käyttöön joko kutsumalla niitä tai siirtymällä niille asetettuun paikalliseen url-osoitteeseen, eli Uniform Resource Locator -osoitteeseen. (17.)

```
$stateProvider.state('tab.months', {
  url: '/months',
  views: {
    'menuContent': {
      templateUrl: 'templates/directory-months.html',
      controller: 'PlantsMonthCtrl'
    }
  }
})
```

Koodiesimerkki 6. Tilan lisääminen Villiruokasovelluksessa.

Koodiesimerkki 6 on Villiruokasovelluksen tilojen määrittelystä, jossa UIWebViewissä toimivalle verkkosivulle lisätään kuukausisivulle oma tilansa. Määritellyn tilan tunnisteeksi annetaan ensimmäisellä rivillä ”tab.months”, josta months on tab-tilan lapsi, joka taas on abstracti tila, joka sisältää sovelluksen yläpalkin käyttämiä arvoja, josta kerrotaan enemmän luvussa 4.4. Tilan luonnin sisälle on asetettu url-osoite, jonne mentäessä näytetään sovellukseen asetetaan ”views”:in sisältö, johon kuuluu templateURL:n tiedostopolun html-sivu. Ohjaimeksi asetetaan PlantsMonthCtrl. ”MenuContent” taas on näkymän nimi, johon tilan sisältö asetetaan. (17.)

4 Villiruokasovellus

Tämän insinöörityön käytännön osana toteutettiin Villiruoka-niminen sovellus asiakkaalle. Sovelluksen ideana oli tarjota käyttäjille kokoelma Suomen luonnosta löytyviä kasveja, joita voi käyttää ruokana tai yrtteinä. Asiakkaalla oli tarjota sovellusta varten Macintosh-tietokone sekä graafinen suunnitelma sovelluksen rakenteesta. Kyseinen suunnitelma koostui staattisista, Photoshopilla tehdyistä kuvista, joten sovelluksen teknisen toiminnan ja animoinnin suunnittelu ja toteutus jäi kehittäjän harteille. Kommunikointi projektin etenemisestä hoidettiin Skrum-henkisesti, jossa asiakkaalle viestitettiin jokaisena koodauspäivänä päivän tulokset, ongelmat sekä mitä tullaan tekemään seuraavaksi.

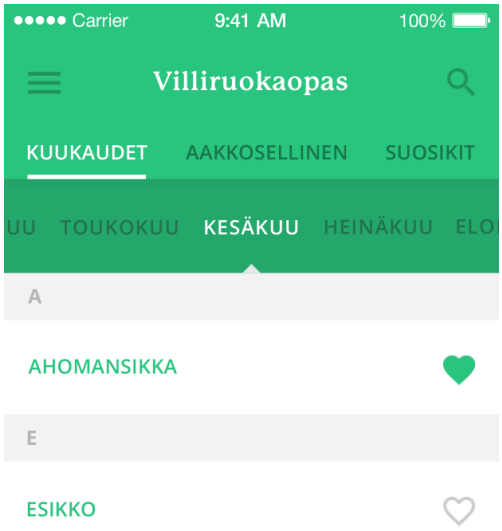
4.1 Suunnitellut ominaisuudet

Villiruokasovelluksen ominaisuudet olivat suunnitelman mukaan suhteellisen yksinkertaisia: Sovellukseen haluttiin kolme osiota, joista ensimmäisessä voitaisiin katsoa kasvilistaa, hakea, filtteröidä sekä järjestää sitä. Sovelluksen toisessa osiossa olisi yleistä tietoa kasvien keräämisestä sekä puhelinnumero myrkytystietokeskukseen, johon voi soittaa myrkytystapauksessa. Kolmas osio sisältäisi tiedot sovelluksen kehittäjistä sekä yleistä tietoa, mutta sen sisältöä ei määrittelyvaiheessa vielä suunniteltu, joten se sovittiin myöhemmin päivitettäväksi. Lisäksi vaatimuksena oli sivulta aukeava punainen valikko, josta voi liikkua sovelluksen eri osioiden välillä sekä viestittää sovelluksesta sosiaalisessa mediassa muutamien eri nappien avulla. Viimeisenä vaatimuksena jokaiselle kasville tuli olla myös oma sivunsa, jossa oli kuvaus kasvista sekä kuvia siitä.

4.1.1 Hakemisto-osio

Sovelluksen tärkein osio sisälsi suunnitelmassa kolme välilehteä, joita käytiin kasvilistan järjestämiseen ja filtteröintiin, eli vain tietyn kriteerien täyttävien kasvien näyttöön. Ensimmäinen niistä oli Kuukaudet-välilehti, jossa kasveja pystyi filtteröimään kuukausien mukaan, joina ne olivat käytettävissä. Käytettävä kuukausi valittaisiin sivun headerin alla olevasta valikosta, jota pystyy vierittämään sekä vasemmalle että oikealle. Valikon tulisi myös keskittää valittu kuukausi sekä valikon vierittäminen tulisi lukkiutua kuukausien keskelle vieritysvauhdin loppuessa. Ylävalikon alla olevalle kasvilistalle haluttiin myös harmaa palkki ennen jokaista uudella alkukirjaimella alkavaa kasvia. Toinen välilehti oli

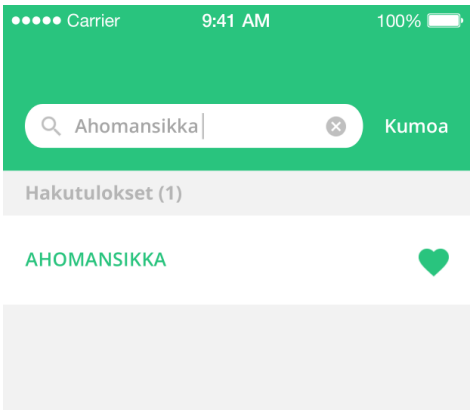
”aakkosellinen”, johon kaikki sovelluksen kasvit lajiteltiin vain aakkosjärjestykseen ilman minkäänlaista filtteröintiä. Tämän välilehden kasvilistauksen väliin sijoitettiin myös harmaita välipalkkeja uudella alkukirjaimella alkavien kasvien ylle. Kolmas välilehti oli ”suosikit”, jossa kaikki käyttäjän suosikiksi asettamat kasvit näytetään, mutta uusien alkukirjaimen edelle ei harmaita välipalkkeja ei siellä käytetty. Kasveja oli mahdollista asettaa suosikiksi joko niiden listauksessessa näkyvää sydäntä painamalla tai kasvien omalla sivulla näkyvästä sydämestä.



Kuva 3. Suunnitelman hakemisto-osio kuukaudet välilehdessä.

4.1.2 Haku-osio

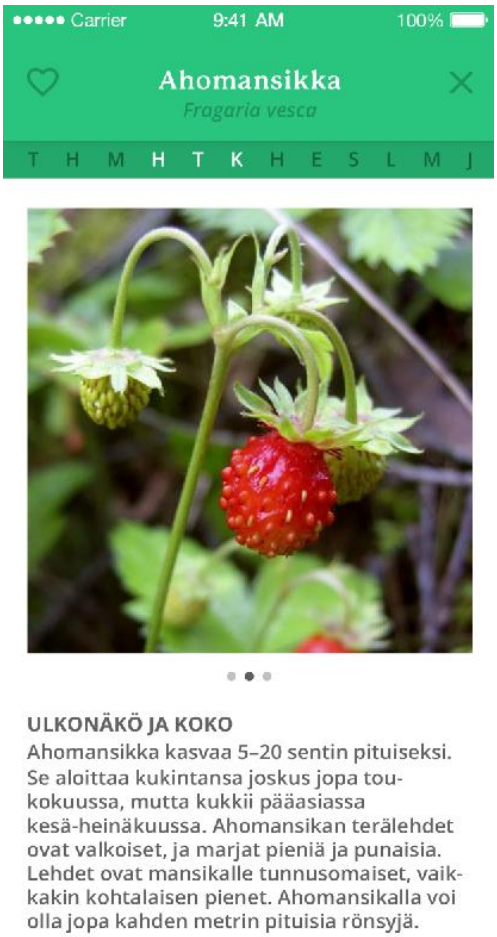
Sovellukseen kuului hakuosio, johon pääsi painamalla sovelluksen oikeassa yläreunassa olevaa suurennuslasia, joka näkyi kaikkialla muualla paitsi kasvien omilla sivuilla. Suurennuslasin ikoni oli Googlen Material Designin ikonikirjastosta, kuten kaikki sovelluksen muutkin ikonit. Hakuosiossa oli hakupalkki, johon kirjoittamalla sovellus haki kasveja nimen mukaan ja näytti ne hakupalkin alla olevassa listauksessa. Hakupalkissa oli nappi, josta sen sisällön pystyi tyhjentämään sekä ”kumoa”-nappi, josta pääsi pois hakusivulta takaisin edelliselle sivulle.



Kuva 4. Suunnitelman haku osio.

4.1.3 Kasvisivu

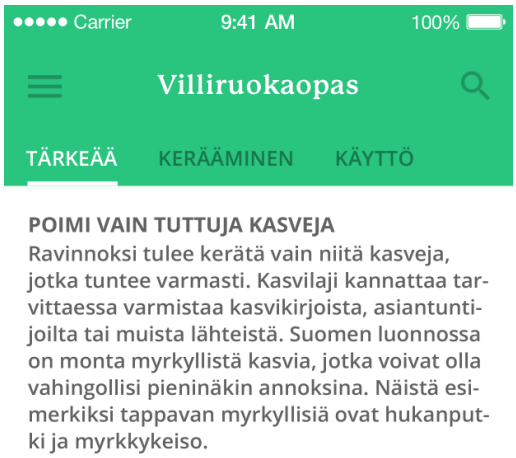
Jokaiselle kasville tuli olla oma sivunsa, johon käyttäjä pääsee painamalla kasvin nimeä hakemisto-osiosta. Sivun yläosan headerista näki kasvin nimen ja latinalaisen nimen sekä pystyi asettamaan kasvin suosikiksi painamalla sydäntä ja poistumaan sivulta painamalla ruksia. Headerin alla oli vielä toinen pienempi header, jossa näkyi kahdentoista kuukauden alkukirjaimet, joista kuukaudet, jolloin kasvia pystyi käyttämään, oli värjätty valkoiseksi. Muuten itse kasvisivu sisälsi kuvia kasvista, joita pystyi selaamaan vetämällä kuvaa vasemmalle tai oikealle sekä kuvaustekstin kasvista, joka sisälsi tekstiä sekä väliotsikoita.



Kuva 5. Suunnitelman kasvisivu.

4.1.4 Hyvä tietää ja info -osiot

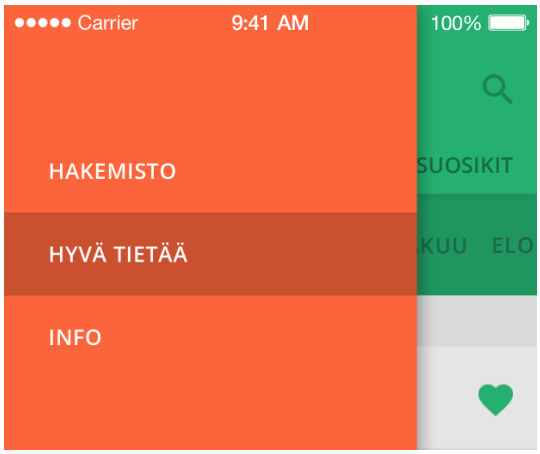
Hyvä tietää -osio sisälsi kolme välilehteä samoin kuin hakemisto-osio: tärkeää, kerääminen ja käyttö. Teknisesti kolme välilehteä olivat lähes identtisiä: ne sisälsivät vain tekstiä väliotsikoineen. Ainoastaan tärkeää-välilehti sisälsi yhden puhelinnumeron, johon pystyisi soittamaan numeroa painamalla. Info-osion sisältöä ei sovellusta suunniteltaessa määritelty muuten kuin että sinne tulisi voida laittaa saman näköistä tekstiä kuin hyvä tietää -osiossa. Sinne haluttiin kaksi välilehteä, joille ei määritelty nimiä eikä sisältöä, joten sinne sovittiin sijoitettavan väliaikaista sisältöä, joka olisi myöhemmin mahdollista korvata oikeaksi tekstiksi.



Kuva 6. Suunnitelman ”hyvä tietää -sivut”. Infosivut olivat tyyliltään samanlaisia, mutta niihin ei määritelty suunnitteluvaiheessa tekstiä.

4.1.5 Sivuvalikko

Kaikilta sivuilta paitsi kasvisivulta on myös mahdollista avata sivuvalikko painamalla vasemmassa yläkulmassa olevaa nappia, jossa on kolme päällekäistä viivaa, eli hampurilaisvalikkoa. Sivuvalikko toimi sovelluksen päänavigaatioväylänä, josta voidaan siirtyä kasvihakemistoon, hyvä tietää-sivulle tai infosivuille. Lisäksi sieltä oli tarkoitus käyttää sovelluksen sosiaalisen median toimintoja painamalla neljää eri nappia. Niille ei määritetty tarkempaa toimintaa, koska mitään sosiaalisen median sivuista kuten Facebook- ja Instagram-sivuja ei ole vielä tehty.



Kuva 7. Suunnitelman sivuvalikko.

4.2 Toteutuksen lähtökohdat

Lukuun 4.1 luetellut sovellukseen halutut ominaisuudet eivät vaikuttaneet kovinkaan monimutkaisilta tai raskailta, eivätkä ne edes vaatineet puhelimen ominaisuuksien käyttöä, mikä mahdollisti hybridisovelluksen tekemisen. Parhain lopputulos olisi varmasti saavutettu natiivilla koodilla, mutta kokemuksen puute Objective C:stä tai Swiftistä sekä ajan puute opettelua varten ajoi hybridisovellusta kohti. Suunnitelman mukaan sovellus oli kuitenkin hyvin yksinkertainen, joten voidaan myös ajatella aika-tulos-suhteella hybridisovelluksen olleen parempi vaihtoehto. Oman hyvän puolensa tuo myös mahdollisuus sovelluksen siirtämiseen Androidille suhteellisen kevyesti, vaikka sitä ei asiakas pyytänytkään.

Ionic valikoitui hybridisovelluksen sovelluskehykseksi sen selkeän dokumentaation ja kasvavan kehitysyhteisön vuoksi. Lisäksi se tarjosi useita iOS:a muistuttavia graafisia komponentteja, mitä pystyisi käyttämään lähes suoraan Villiruokasovelluksessa, joita käyttämällä sovelluksen rakentaminen nopeutuisi. Kehityksen alkaessa Ionicista oli tarjolla kaksi versiota: Ionic 1 ja vielä sillon beta-vaiheessa oleva Ionic 2. Ionic 1 valittiin, koska se oli valmiimpi ja varmempi vaihtoehto muutenkin hiukan epävarmassa hybridisovellusten maailmassa. Se osoittautui kuitenkin myöhemmin virheeksi, koska Ionic 2 ehdittiin julkaista kesken sovelluksen kehitysprosessia. Ionic 2 olisi tarjonnut muutaman sovellusta parantavan ominaisuuden, kuten nopeamman WKWebViewin käytön.

Toinen varteenotettava vaihtoehto olisi ollut React Native, jolla olisi saanut tuotettua lopputuloksena natiiveja komponentteja käyttävän mobiilisovelluksen. Käytännössä se kariutui pois Ionicin tieltä, koska React eroaa huomattavasti muista web-kielistä ja se olisi vaatinut omaa opetteluaan. Se ei myöskään tarjoa yhtä monta valmista graafista komponenttia kuin Ionic, mikä olisi pidentänyt projektia entisestään, koska iso osa komponenteista olisi pitänyt tehdä itse. Tässä heräsi myös ajatus, jos kerran React-koodaaminen olisi opeteltava ja loppullinen sovellus käyttäisi natiiveja komponentteja, miksei vain suoraan käyttäisi aikaa natiivin koodin opetteluun ilman kiertoteitä? (6.)

Ionic-sovellusta tehtäessä oli myös mahdollisuus käyttää tyylien määrittelyyn SASS:isa eli Syntactically Aweseome Style Sheetsiä CSS3:n sijaan. CSS3 päädyttiin kuitenkin käyttämään vanhasta tottumuksesta, ja tässä vaiheessa tarpeen muokata sovelluksen tyylejä arviotiin paljon todellisuutta pienemmäksi, jolloin tyylitiedosto pysyisi hallittavan kokoisena.

4.3 Työympäristön asennus

Sovelluksen kehitys alkoi heti kehittäjän saatua Macintosh-tietokoneen El Kapitan käyttöjärjestelmällä ja Ipod Touchin laitteella testausta varten. Myöhemmin käyttöön saatiin myös Iphone 5. Saatu tietokone oli käytännössä tyhjä, joten siihen täytyi asentaa Ionicilla kehitykseen tarvittavat ohjelmat. Kovinkaan montaa eri ohjelmaa ei tarvittu ja asennusprosessi oli suhteellisen mutkaton, yhdessä päivässä ohi ollut urakka.

Tavalliseksi tekstieditoriksi valikoitui aiemmassa käytössä hyväksi todettu Sublime 3, jonka avulla pystyi muun muassa liikkumaan sovelluksen kansiorakenteessa ja avaamaan sieltä tiedostoja muokattavaksi. Versionhallintaan käytettiin ilmaista Bitbucketin palvelinta, johon siirrettiin koodit Source Tree -ohjelmalla aina päivän lopuksi. Ionicin asentamiseen tarvittiin npm eli Node package manager, jota käytettiin Macintoshin terminaalista. Ajamalla asennuskomentoja npm asentaa automaattisesti kaikki asennettavan paketin riippuvuudet. Ionicin työtilaa luotaessa riitti vain muutaman komennon ajaminen. (9)

```
npm install cordova –g
npm install ionic
ionic start villiruoka tabs
```

Koodiesimerkki 7. Npm:llä ajetut komennot Ionicin työtilaa luotaessa sekä Ionic kehittämisen käynnistäminen kolmella Ionicin komennolla

Koodiesimerkissä 7 asennetaan kaikki Ionicin npm:stä tarvitsemat osat eli Cordova ja itse Ionic. Tämän jälkeen käynnistetään sovelluksen kehittäminen luomalla projekti valmiilta ”tabs”-pohjalla, joka oli välilehtiä esittelevä esimerkkisovellus. Ennen kuin Ionicilla voitiin aloittaa ios-sovelluksen luonti, täytyi tietokoneelle asentaa ios-sovellusten rakentamiseen käytetty Xcode, jonka käyttäminen vaati Apple-tilin luomisen. Itse Xcode on ilmainen kehitysympäristö, mutta jos sillä tehtyjä sovelluksia haluaa julkaista App Storessa, on Apple tilistä maksettava vuosittainen maksu. Kun Xcode oli asennettu, Ionic-sovellukselle oli mahdollista lisätä iOS-alusta komennolla ”ionic platform add ios”. Tämä generoi sovellukselle useita ikonitiedostoja eri puhelinmalleille sekä liittää Ionicin paikalliseen Xcodeen, jonka kautta sovellus rakennetaan ja emuloidaan.

Ohjelmien ja riippuvuuksien jälkeen Ionic-sovelluksen kehittäminen voitiin aloittaa. Sovellusta oli siis mahdollista testata kolmella eri tavalla: joko selaimessa, suoraan laitteella tai emulaattorilla. Järkevintä oli testata sovelluksen tavallinen toimivuus selaimessa,

koska koodimuutosten näkyminen tapahtui lähes heti Ionicin mukana tulleen Gulp-työkalun avulla. Sovelluksen rakentaminen laitetta tai emulaattoria varten vei taas kymmenisen sekuntia, mikä olisi jatkuvassa käytössä vienyt paljon aikaa. Lopullinen ulkonäön hiominen oli siis hyvä tehdä vasta itse laitteella ajan säästämiseksi. Sovelluksen testaamiseen käytetyt komennot näkyvät koodiesimerkissä 8.

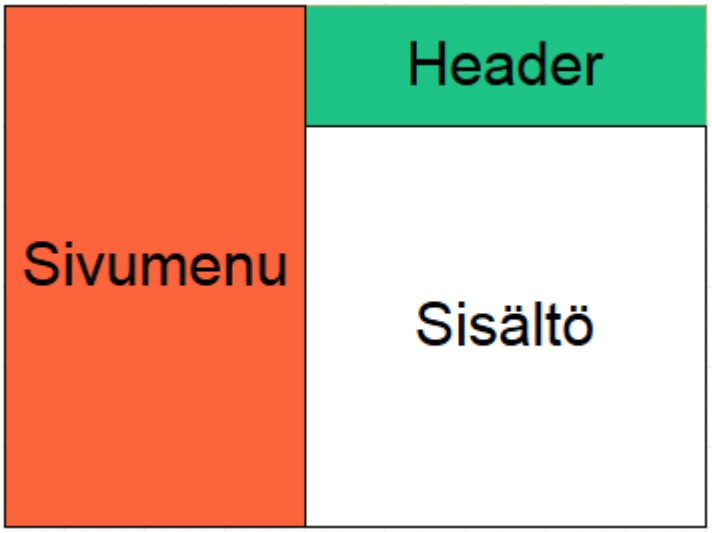
```
Ionic serve
Ionic build ios
Ionic emulate ios
```

Koodiesimerkki 8. Ionic-sovelluksen rakentamiseen testausta varten käytetyt komennot.

Ionic serve oli päällelaitettava prosessi, joka aloitti sovelluksen kansiorakenteen kuuntelemisen ja julkaisi koodimuutokset suoraan paikalliselle sivulle, josta sovellusta voitiin testata. Tämä oli mahdollista Gulp:in avulla. ”Ionic build ios” rakensi koodin puhelinta ja emulaattoria varten ja ”Ionic emulate ios” oli mahdollista käynnistää emulaattori sovelluksen testaamista varten. Emulointi ja laitteella testaamisen käynnistäminen oli myös mahdollista ja ehkä jopa helpompaa Xcoden graafisen käyttöliittymän kautta, koska Xcoden kautta pystyi vaihtamaan emuloitua iPhonea sekä se ilmoitti suoraan sovelluksen rakentamiseen liittyvistä ongelmista, jotka joka tapauksessa piti usein ratkaista Xcoden kautta.

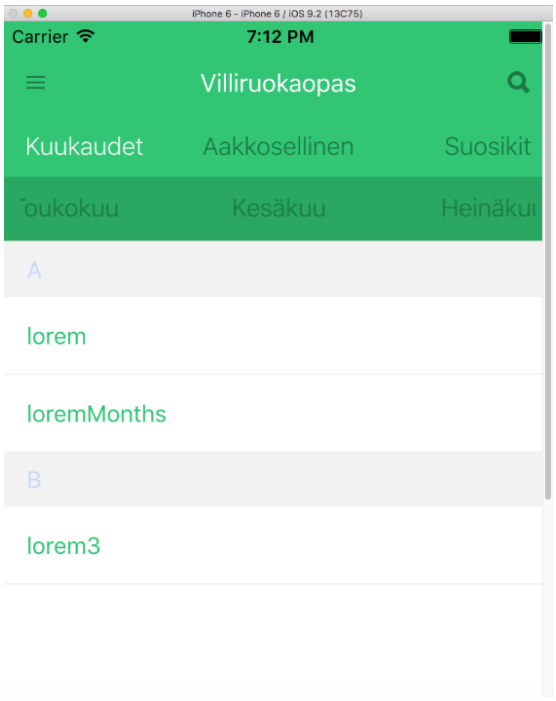
4.4 Toteutuneet ominaisuudet

Ionic toi sovelluksen kehittämiseen omat vahvuutensa ja heikkoutensa. Vahvana puolena oli kehityksen nopea käynnistyminen ja tulosten synty: jo muutaman ensimmäisen päivän jälkeen saatiin toteutettua suurin osa hakemisto-osion ominaisuuksista käyttäen pelkästään Ionicin omia komponentteja. Tarkoitus oli tässä vaiheessa käyttää Ionicin tarjoamia osia ja muokata niistä myöhemmin tyyliltään enemmän suunnitelman mukaisia. Erittäin hyödyllinen oli ainakin Ionicin yläpalkin toteutus, johon pystyi kirjoittamaan nopeasti otsikkotekstin ja lisäämään kaksi ikonia Ionicin omasta Ionicons-ikonikirjastosta.



Kuva 8. Villiruokasovelluksen tekninen rakenne

Villiruokasovellus päädyttiin jakamaan kolmeen osaan kuvan 8 mukaisesti: headeriin, sisältöön ja sivumenuun. Tämä käytännössä toimi niin, että header ja sivumenu toimivat omana erillisenä osana sisällöstä. Tieto käyttäjän liikkumisista välittyi molemmille osille, jolloin header eli ylävalikko pystyi muuttamaan esimerkiksi yläikoneitaan tai otsikkoaan erillaiseen muotoon. Tähän ratkaisuun päädyttiin, koska sivujen ylämenu oli jokaisella sivulla lähes samanlainen. Lisäksi header- ja sisältö-osat tarvitsivat erillaiset siirymäanimaatiot. Suunnitteluvaiheessa päätettiin, että ylämenu ei saisi siirtyä muun sivun mukaan, minkä takia header ja sisältö pilkottiin omiksi kokonaisuuksiksiin, jolloin kummallekkin osalle voitiin tehdä omat animaationsa. Teknisesti tämä tehtiin käyttämällä Angularin templateja: header ja sivumenu kuuluivat ylimpään template-tiedostoon, jonka sisälle sijoitettiin aina nykyisen sivun template-tiedosto ion-nav-view-elementtiin.



Kuva 9. Villiruokasovelluksen alkuvaiheilta oleva kuva, jossa käytetään Ionicin omia komponentteja vain pienillä muokkauksilla.

Kehitystyö vaikeutui kuitenkin vaiheessa, jossa aloitettiin sovelluksen ulkonäön muokkaaminen enemmän suunnitelman mukaiseksi. Tämä tarkoitti käytännössä Ionicin omien tyylien ylikirjoittamista CSS:llä, mikä oli ajoittain haastavaa Ionicin oman laajan tyylitiedoston takia. Muutamaan kertaan jouduttiin käyttämään huonona käytäntönä pidettyä ”!important”-sääntöä, joka pakottaa elementin käyttämään tärkeäksi merkittyä tyylin osaa. Sääntöä oli pakko käyttää, koska Ionicin oma tyylitiedosto antoi sen alkuperäisille tyyleille CSS:ssä liian korkean prioriteetin, joten ainoaksi vaihtoehdoksi jäi merkitä tarvittavat tyylien osat tärkeiksi kuten koodiesimerkissä 9.

```
.cancel-icon::before {
  font-size: 17px !important;
}
```

Koodiesimerkki 9. Hakusivulla olevan peruutusnapin koon muokkaaminen pienemmäksi !important-säännöllä.

Yllä olevassa koodiesimerkissä pienennetään hakusivulla olevan peruutusikonin kokoa !important-säännöllä. Ikonit toimivat Ionicissa fontteina, joten niiden kokoa säädettiin fonttikokoa käyttämällä. Ionicin oma tyylitiedosto antoi ikoneille oman kokonsa käyttäen ”before”-pseudoelementtiä, minkä takia ikonin tyyliä ei pystytty muokkaamaan käyttä-

mättä myös samaa valitsinta sovelluksen omassa tyylitiedostossa. Tyyli oli vielä asetettava tärkeäksi, koska Ionicin oma joukko tyylejä button-icon:ille sai CSS:n sääntöjen mukaan liian korkean prioriteetin. Tämän kaltaisia tapauksia oli useita, mitkä haittasivat tyylitiedoston koodin ylläpidettävyyttä.

Kehityksen aikana suunnitelmiin tehtiin myös muutamia muutoksia. Alunperin sovelluksessa suunniteltiin käytettävän fonttia, joka paljastui kehityksen aikana maksulliseksi. Fontit muutettiin tämän jälkeen Googlen Material designin Open sansiksi ja otsikoissa Volkorniksi. Lisäksi teknisen toiminnan määrittelyn puute aiheutti epäselvyyksiä kasvihaun toiminnasta: pitäisikö kasveja hakea kaikilla kasvista saatavilla olevan datan perusteella vai vain nimen? Tässä päädyttiin vain nimeen, vaikka käyttämällä haussa myös kasvin kuvausta olisi saanut tarkempia hakutuloksia. Tästä syystä jatkuva yhteistyö asiakkaan kanssa on arvokasta ja Skrum-tyylinen kommunikointi toimi ainakin tämän projektin aikana.

Seuraavissa kappaleissa käydään läpi toteutuneen sovelluksen ominaisuudet sekä tekniset ratkaisut niiden takana.

4.4.1 Kasvit-JSON

Villiruokasovelluksen sisältämät kasvit päädyttiin sijoittamaan JSON-tiedostoon eli JavaScript Object Notation-tiedostoon. JSON-tiedostot koostuvat joukosta avain- ja arvo pareja, joista avainten avulla voidaan hakea tiettyä osaa, kuten koodiesimerkin 10 tapauksessa voitaisiin esimerkiksi hekea kasvin nimi ”name” avaimella. Tällöin saataisiin JSONia käsittelevässä Javascript-koodissa arvo ”Test Plant”. Arvojen ei tarvitse olla pelkkää tekstiä, niihin voi myös sijoittaa toisia taulukoita kuten months-avaimen kanssa on tehty.

```
{
“id”: “0”,
“name”: “Test Plant”,
“latinName”: “Plantus Ipsumus”,
“months": [5,6,7],
“description": “Lorem ipsum dolor sit amet”,
“images": “img/flower.jpg,img/flower2.jpg,img/flower3.jpg”
}
```

Koodiesimerkki 10. Ote JSON-tiedostossa käytetystä testikasvista.

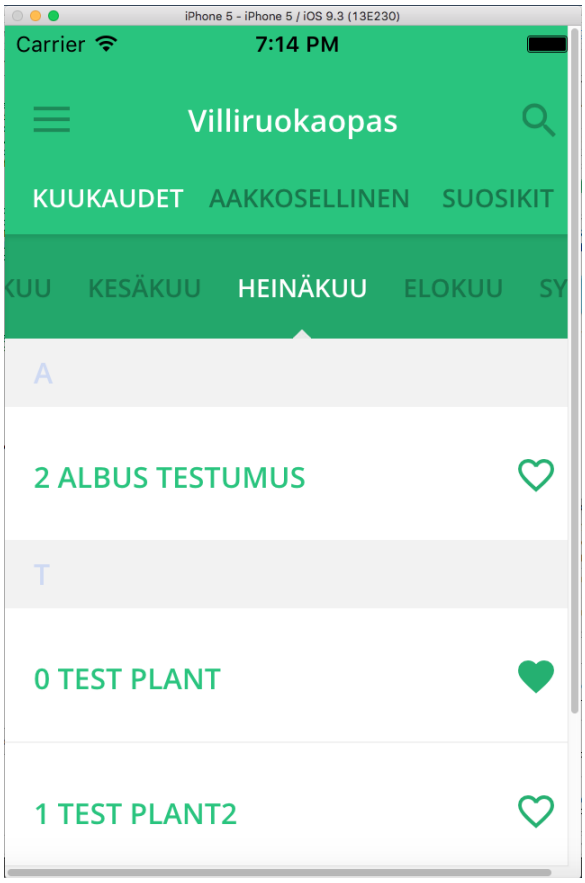
Jokainen kasvi oli JSON-tiedostossa oma osansa yhtä isoa kasvitaulukkoa, joka käytiin läpi hakemisto-osioiden listauksissa ja tulostettiin sinne kaikki JSON-tiedostosta löydetyt kasvit. Lisäksi kasvien omien sivujen sisältö generoitiin JSON:ista löytyvien tietojen perusteella. Tämän rakenteen avulla uusien kasvien lisääminen vaati vain lisäyksen JSON tiedostoon. JSON:ista löytyvät tiedot ja niiden selitykset ovat taulukossa 2.

Taulukko 2. JSON-tiedoston avaimet ja niiden tarkoitus

Avain	**Selite**
Id	Kasvin tunnistinnumero, jonka on oltava uniikki jokaisella kasvilla.
name	Kasvin nimi, joka tulostetaan kasvilistauksissa sekä kasvin omalla sivulla.
latinName	Kasvin latinalainen nimi, joka tulostetaan kasvin suomenkielisen nimen alle.
months	Taulukko, johon laitetaan kaikkien kuukausien numero, jolloin kasvi on käytettävissä. Kuukaudet näytetään kasvisivulla ylätunnisteen alla .
description	Kasvin kuvausteksti, joka tulostetaan kasvisivulle. Kuvaustekstissä on mahdollista käyttää h5-html tagia, joka luo kuvaustekstiin väliotsikon.
images	Tiedostopolut kasvisivulle sijoitettaviin kuviin. Useampia tiedostopolkuja on mahdollista antaa pilkulla erotettuna.

4.4.2 Hakemisto-osio

Suunnitelmien mukaisesti hakemisto-osio sisälsi kolme välilehteä: kuukaudet, aakkosellinen ja suosikit. Kaikilla kolmella osiolla oli hyvin samankaltainen periaate: näytetään kasvilista, jota filtteröidään tietyllä tavalla. Jokaisessa välilehdessä oli myös mahdollista asettaa kasveja suosikeiksi, jolloin ne näkyvät suosikit-välilehdessä.



Kuva 10. Hakemisto-osion kuukaudet-välilehti.

Kasvit olivat jokaisella sivulla aakkosjärjestyksessä, mikä toteutettiin Javascriptin sort-funktiolla, joka asetettiin käyttämään kirjaimia vertailevaa funktiota. Järjestämisen jälkeen uusilla alkukirjaimilla alkavien kasvien eteen lisättiin harmaita palkkeja, joissa oli uusilla alkukirjaimilla alkavien kasvien alkukirjain. Alkukirjainpalkit lisättiin samaan taulukkoon, jossa kasvitkin olivat, mutta niihin asetettiin ”divider”-niminen muuttuja todeksi, joka kasveja tulostettaessa erotti harmaan palkin kasvista. Alkukirjainpalkit ja kasvit sisältävä taulukko sijoitetaan lopuksi Angularin tarjoamaan ”scope”-muuttujaan, josta sitä voidaan käyttää myös template-sivuilta sen nimeä viittaamalla. Listaussivujen kasvilistoille oli rakennettu oma logiikansa, jolloin kasvitaulukosta kaikki objektit, joiden divider muuttuja oli tosi, muutettiin alkukirjainpalkeiksi. Lisäksi aina listaussivustoille mentäessä sovellus tarkasti, mitkä kasvit oli asetettu suosikeiksi, asettti ne Angularin scopeen ja niiden listoissa näkyvät sydämmet täytettiin. Suosikkikasvien tunnistinnumeroja säilytettiin paikallisessa varastossa eli localStoragessa, missä niitä oli myös mahdollista säilyttää sovelluksen ollessa kiinni.

```
$scope.favorites = getFavorites();
var sortedPlants = response.plants;
sortedPlants.sort(alphabeticalSort);
sortedPlants = addDividers(sortedPlants);
$scope.plants = sortedPlants;
```

Koodiesimerkki 11. Kasvilistoille tehtävä alustus kun kaikki kasvit on haettu JSON-tiedostosta.

Aakkossivun alustus toimi täysin koodiesimerkki 11 mukaisesti, mutta kuukausisivun ja suosikkisivun alustuksessa filtteröitiin JSON:ista saatavaa kasvilistaa. Suosikkisivuilla käytännössä tarkistettiin vain, onko kasvi asetettu suosikiksi localStoragessa. Jos ei, sitä ei näytetä sivun listauksessa. Lisäksi suosikkisivuilla ei käytetty alkukirjainpalkkeja.

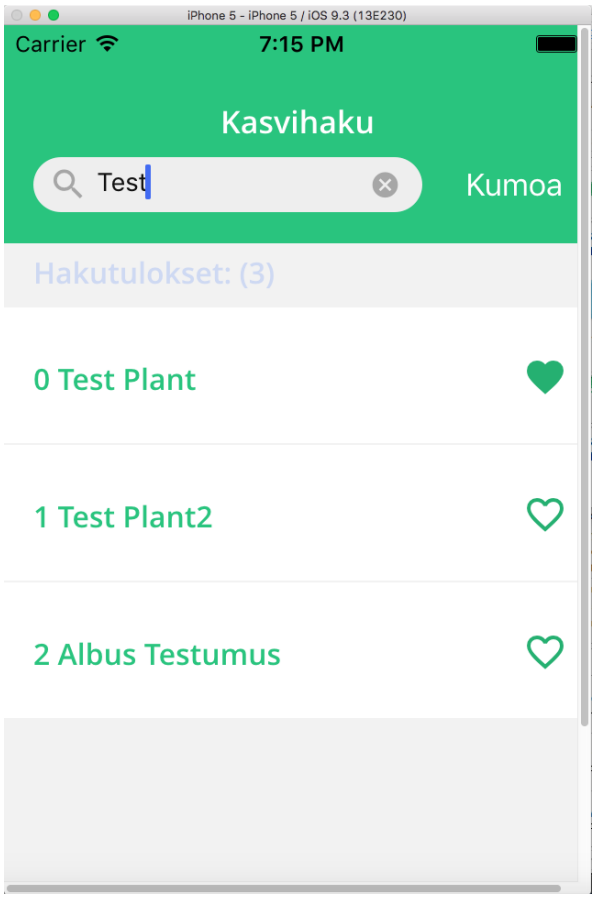
Kuukausisivu oli toiminnaltaan monimutkaisin. Siellä kasveja oli mahdollista filtteröidä pois kaikki muu paitsi valittu kuukausi, jolloin kasvi on käytettävissä. Tähän käytettiin JSON-tiedoston kasveista löytyvää ”months”-taulukkoa ja näytettävä kuukausi oli mahdollista valita vieritettävästä kuukausipalkista, joka näkyy kuvassa 10. Uutta kuukautta painettaessa kasvilistaan asetetaan vain painetun kuukauden aikana käytettävissä olevat kasvit.

Kuukausivun kuukausipalkin alkuperäinen suunnitelma aiheutti kuitenkin Ionicissa ongelmia. Alunperin siitä haluttiin Ios-sovelluksissa usein käytetty valitsin, jota on mahdollista vierittää, mutta vierittäminen voi pysähtyä vain kohteiden keskelle. Lisäksi kohteita painettaessa valitsin siirtyy painetun kohteen keskelle. Jälkimmäinen ominaisuus oli mahdollista tehdä Ionicin ScrollDelegaten metodin ”scrollTo” avulla, mutta palkin lukkiutuminen vain tiettyihin kohteisiin vieritettäessä tuotti ongelmia ja lopulta siitä luovuttiin. Se olisi voinut olla mahdollista kuuntelemalla vierityksen nopeutta, mutta se todettiin muutamien kokeilujen jälkeen liian epävakaaksi ja raskaaksi tavaksi, koska kuuntelun ollessa päällä sovellus pysähtyi useaan kertaan. Tämä oli ominaisuus, mikä olisi ollut mahdollista toteuttaa natiivissa sovelluksessa esimerkiksi ACTabScrollView’nä. (18.) Tämä jäi hybridisovelluksessa ainoaksi isoksi puutteeksi, mikä oltaisiin voitu paikata tekemällä sovellus perinteisellä natiivilla tavalla. Loppujen lopuksi kuukausien valitsinpalkista päädyttiin tekemään ilman lukkiutumista kohteisiin palkkia vieritettäessä.

4.4.3 Haku-osio

Koska sovellus päätettiin jakaa sekä sisältö että header osiin, hakusivulle täytyi järjestää jokin tapa, jolla headerin hakupalkki ja sisällön kasvilista voisivat kommunikoida keskenään kuten kuvassa 11, jossa hakupalkin tekstiä käytetään sisällön etsimisessä. Tätä

varten headerin käyttämälle controllerille tehtiin oma funktionsa, joka näkyy koodiesimerkissä 12.



Kuva 11. Valmis kasvihaku.

```
$scope.searchChange = function(input) {
    $rootScope.$broadcast('searchContent', input);
  }
```

Koodiesimerkki 12. Menu Controllerin käyttämä funktio hakulauseen lähettämiselle

Koodiesimerkin 12 funktio käyttää Angularin rootScopea, mikä on ylin scope koko sovelluksessa ja jonka alaisia template-sivujen paikalliset scopet ovat. Sinne on mahdollista lisätä arvoja ja funktioita, jotka ovat saatavilla sovelluksen jokaisessa templatessa. Sen käyttöä on kuitenkin hyvien käytäntöjen mukaisesti vältettävä, koska se tekee liian paljon käytettynä sovelluksen tapahtumien seurannasta vaikeaa. Tässä sitä käytetään lähettämään sanoma, joka vastaanotetaan koodiesimerkin 13 koodissa.

```
$scope.$on('searchContent', function (event, arg) {
    $scope.searchQuery = arg;
  });
```

Koodiesimerkki 13. Hakusivun ohjaimeen luotu kuuntelija, joka vastaanottaa koodiesimerkin 12 sanoman.

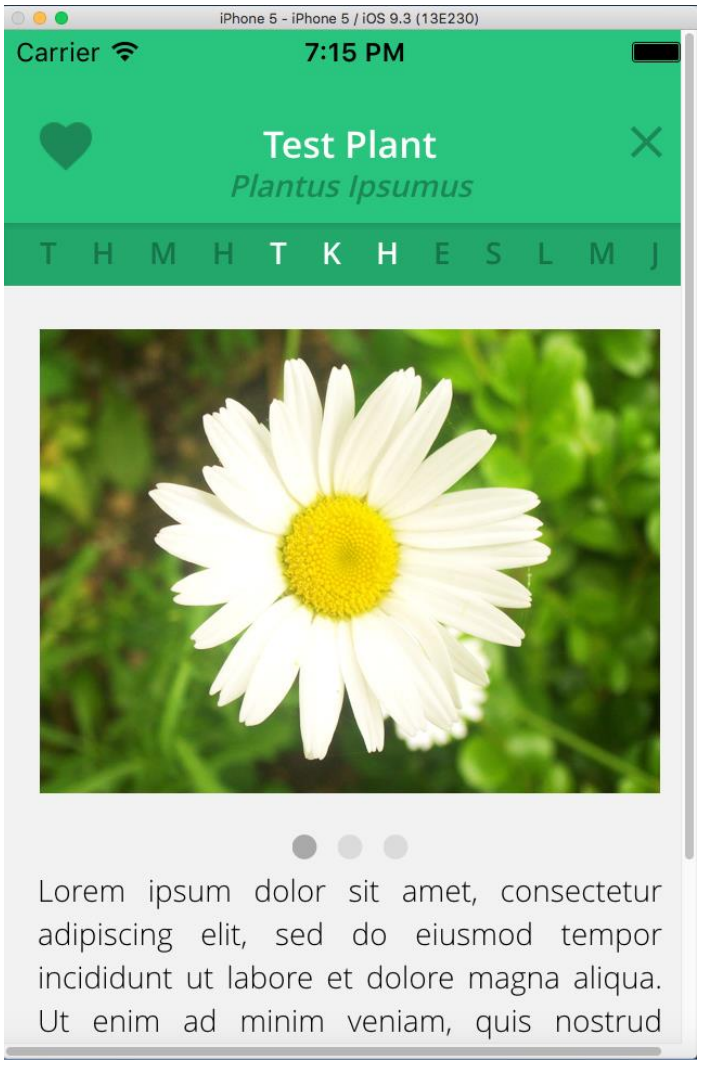
Koodiesimerkin 13 koodissa hakusivulle asetetaan kuuntelija, joka odottaa ”searchContent”-sanomaa ylämenun controllerilta. Sanoma lähetetään rootScopen kautta. Kun hakuviesti saadaan, se sijoitetaan hakusivun omaan scopeen ja siellä sitä käytetään kasvilistan filtteröimisessä. Jotta kasvi löydetään, hakuviestin on oltava osa yhden tai useamman kasvin nimeä. Koodiesimerkissä 14 käytetään Angularin filtteröintiä annetulla hakusanalla. Kasvilista toimii muuten samankaltaisesti kuin muissakin hakemiston listauksissa, filtteröintikriteerinä käytetään hakusivulla vain käyttäjän antamaa hakusanaa.

```
<ion-item ng-repeat="data in filteredPlants =
        (plants | filter:{name:searchQuery})">
```

Koodiesimerkki 14. Hakusivun listaus, jossa tehdään filtteröinti headerista saadun hakutekstin mukaan.

4.4.4 Kasvisivu

Jokaisen kasvin omat sivut luodaan Kasvit-JSON:ista saadun tiedon perusteella: kasvin nimi sekä latinalainen nimi sijoitetaan yläpalkkiin, months-taulukon kuukaudet värjätään yläpalkin alle valkoiseksi, images-tekstistä haetaan pilkulla erotetuista poluista kuvat kasvista ja sijoitetaan ne Ionicin ion-slide-box:iin, jossa kuvia voidaan vaihtaa vierittämällä. Lopuksi kuvien alle sijoitetaan description-kentän teksti kasvin kuvaukseksi, johon on myös mahdollista tehdä väliotsikoita käyttämällä HTML:n h5-tagia. HTML-tagien käyttö onnistui käyttämällä Angularin ng-bind-html-direktiivillä, joka mahdollistaa myös muiden tagien käytön, mutta vain h5-tagille luotiin oikeanlainen tyyli.



Kuva 12. Test Plantille luotu kasvisivu.

Haasteita kasvisivulla olivat latinalaisen nimen ja kuukausien saanti yläpalkkiin, koska sovelluksen yläpalkki ja sisältöosa toimivat erillisinä osinaan ja kaikken kasvien tiedot olivat sisältöosassa. Tässä päädyttiin lähettämään kasvin latinalainen nimi ja kuukaudet rootScopen kautta menun ohjaimeen aina, kun käyttäjä painaa kasvilistalta kasvia, kuten koodiesimerkissä 15. Menun ohjaimessa rootScopen arvot siirrettiin paikalliseen scopeen, josta niitä lopuksi käytettiin kasvisivun yläpalkissa. Tämä oli huono, nopea ratkaisu tähän ongelmaan. RootScopessa oleva siirtymäfunktio oli kuitenkin yksinkertaisuudessaan kaikkein monimutkaisin funktio, joka sinne asetettiin.

```
$rootScope.go = function(url, latinName, months){
    if (latinName != null) {
      $rootScope.latinNameRoot = latinName;
      $rootScope.monthsRoot = months
    }
    $location.path(url);
  };
```

Koodiesimerkki 15. Rootscopeen asetettu funktio, jolla siirrytään kasvisivulle. Samalla siirretään rootScopen kautta kasvin latinalainen nimi sekä kuukaudet.

4.4.5 Hyvä tietää ja info -osiot

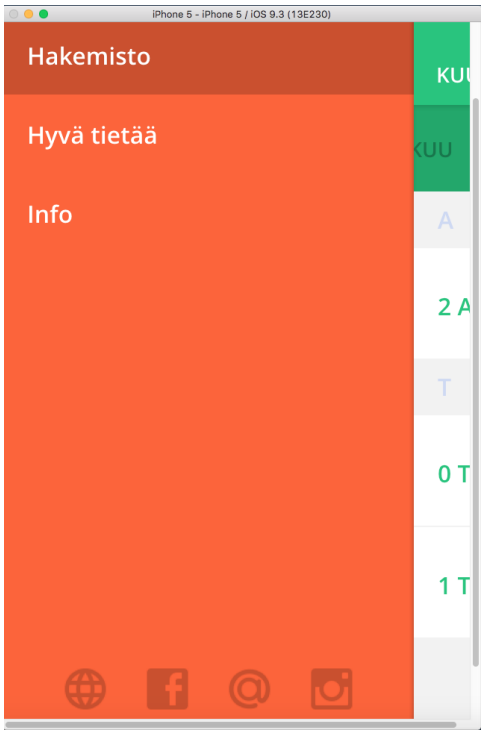
Hyvä tietää- ja info-osiot päätyivät olemaan kuin kasvisivuja ilman kuvia: ne käyttivät samoja tyylejä kuin kasvisivun teksteissä ja tärkeää-välilehden puhelinnumero onnistui helposti HTML:n linkki-tagilla, jonka sisältämän puhelinnumeron eteen oli lisätty puhelinnumeron tunniste ”tel” koodiesimerkin 16 mukaisesti. Kyseinen tunniste on HTML5:n tarjoama, jonka avulla avataan puhelimen soittamisvalikko. Info-sivuille ei määritetty sisältöä, joten sovelluksen ylläpitäjän on tulevaisuudessa asetettava sinne tekstiä, kuten tekijäluettelo.

```
<a href="tel:+3589471977">p. 09 471 977</a>
```

Koodiesimerkki 16. HTML:n linkki-tagi, jossa on soitettava puhelinnumero.

4.4.6 Sivuvalikko

Sivuvalikko kuului teknisesti samaan osaan kuin yläpalkit, jolloin se käytti myös samaa ohjainta ja template-sivua. Ionic tarjosi oman sivuvalikkonsa sekä animoinnin sille, joiden pohjalta rakennettiin Villiruokasovelluksen toteutus. Sivuvalikko ei itsessään ollut teknisesti yllättävä, mutta sovelluksessa liikkumisen kannalta sillä oli tärkeä rooli. Siellä oli kolme tekstimuotoista nappia, josta pystyi siirtymään kasvihakemistoon, hyvä tietää- ja infosivuille. Alempana oli sosiaalisen median nappeja, joiden on tulevaisuudessa tarkoitus linkittää sovelluksen kotisivuille, Facebook-sivuille, omistajan sähköpostiin sekä Instagramiin. Koska sivuja ei ole vielä tehty, napit eivät kehityksen päättyessä tehneet mitään. Niihin on kuitenkin mahdollista lisätä helposti tarvittavat osoitteet, paitsi Instagram-nappi, joka todennäköisesti vaatisi jonkin Cordovan pluginin, koska sovelluksen täytyisi avata Instagramin sovellus, mikä ei onnistu ilman puhelimen rajapintaa.



Kuva 13. Sovelluksen sivuvalikko. Kuvasta näkyvät eri osioihin vievät napit sekä alempana sosiaalisen median ikonit.

4.5 Jatkokehitysideat

Sovelluksen suunnitelman kaikki suunnitellut ominaisuudet saatiin toteutettua lukuunottamatta kuukausi-sivun kuukausivalitsinpalkkia. Kyseinen valitsin oli hyvin vaikeaa toteuttaa sulavasti hybridisovellusten keinoilla, joten jos valitsinpalkista haluttaisiin enemmän iOS:n tyylinen, sovellus kannattaisi varmuuden vuoksi tehdä natiivina. Hybridisovellukset ovat kuitenkin vielä kehittyvä älypuhelinsovellusten muoto, mikä voi tarjota tulevaisuudessa parempia mahdollisuuksia toteuttaa alunperin kaavailtu kuukausivalitsin. Nopea kehitys näkyikin sovelluksen rakentamisen aikana, jolloin Ionic 2 julkaistiin kehityksessä ollessa vielä käytössä Ionic 1. Ionic 2 olisi vaatinut siirtymisen myös Angularin ensimmäisestä versiosta toiseen, mikä olisi vaatinut koodin uudelleenkirjoittamista suurimmassa osassa sovellusta, koska Angular 2:n rakenne on reilusti erillainen Angular 1:een verrattuna. Siirtyminen Ionic 2:een olisi kuitenkin sovellukselle luonnollinen seuraava askel, koska se tuo joukon uusia ominaisuuksia, kuten mahdollisuuden käyttää iOS:illa parempaa selainnäkymää, WKWebView:iä, mikä olisi voinut tehdä sovelluksesta sulavemman. (12.) Lisäksi Ionicin 2 Ionicons-kirjastossa siirryttiin käyttämään Googlen Material Designin ikoneja, joita Villiruokasovellukseen haluttiin. Ionic 1 Ionicons ei niitä

sisältänyt, joten helppokäyttöisen kirjaston sijasta joudutiin käyttämään käsin asetettuja kuvia. Uuden Iconiconssin käyttäminen olisi siis selkeyttänyt koodia. Lisäksi sovellukseen ei tarjottu omia käynnistysikoneita tai latausruutuja eri ruutukoille, jotka sovittiin asiakkaan toteutettavaksi. Lopputuotteessa sovellus käytti Ionicin tarjoamia oletusikoneita sekä latausruutua

5 Yhteenveto

Insinöörityön tavoitteena oli kertoa lukijalle Ionic-sovelluskehyksellä kehitetystä sovelluksesta ja sen kehitysprosessista, kertoa sen hyvistä ja huonoista puolista sekä siitä milloin hybridisovelluksen kehittäminen on varteenotettava vaihtoehto. Näihin annettiin sekä teoreettisia vastauksia että käytännön työn aikana saatuja tapausesimerkkejä.

Insinöörityön teoreettinen osa aloitettiin kertomalla mobiilisovelluksista ja kolmesta eri tavasta tehdä niitä. Niistä kerrottiin niiden hyvät ja huonot puolet, sekä milloin niiden käyttäminen olisi järkevintä. Tämän jälkeen keskityttiin hybridisovelluksien kehittämiseen käytettyyn Ionic-sovelluskehykseen. Ionicin tärkeimmät riippuvuudet, Cordova ja Webview esiteltiin, jonka jälkeen käytiin läpi Ionicilla luodun verkkosivun käyttämät web-teknologiat, joista tärkein oli Angular.

Insinöörityön käytännön osassa toteutettiin asiakkaan tarjoamasta Photoshop-tiedostosta Ionic-sovelluskehyksellä tehty Villiruoka-niminen hybridisovellus. Aluksi käytiin läpi sovelluksen suunnitelma ja sen asettamat vaatimukset ja perusteltiin sen pohjalta päätös toteuttaa sovellus hybridisovelluksena. Tämän jälkeen käytiin läpi kehityksen aikana huomattuja Ionicin hyviä sekä huonoja puolia ja esiteltiin lopputuloksena syntynyttä sovellusta, johon saatiin lähes kaikki suunnitellut ominaisuudet toteutettua. Lopuksi käytiin läpi valmiin Ionic-sovelluksen mahdollisia jatkokehityskohteita.

Lähteet

1 Mobile App. Verkkodokumentti. Wikipedia. 2016. <https://en.wikipedia.org/wiki/Mobile_app/>. Luettu 1.10.2016.

2 What is a Hybrid Moblie App? Verkkodokumentti. John Bristowe. 2015. <http://developer.telerik.com/featured/what-is-a-hybrid-mobile-app/>. Luettu 25.9.2016.

3 Native, Web or Hybrid Apps? What's The Difference? Verkkodokumentti. Mobiloud. 2016. <https://www.mobiloud.com/blog/native-web-or-hybrid-apps/>. Luettu 25.9.2016.

4 Record-Breaking Holiday Season for the App Store. Verkkodokumentti. Apple. 2016. <http://www.apple.com/pr/library/2016/01/06Record-Breaking-Holiday-Season-for-the-App-Store.html/>. Luettu 1.10.2016.

5 Native vs Hybrid - Demystifying the technology dilemma. Verkkodokumentti. Krzysztof Marszałek. 2016. <http://howwedostartups.com/articles/Native-vs-Hybrid/> Luettu 1.10.2016.

6 Getting Started. Verkkodokumentti. Facebook inc. 2016. <https://facebook.github.io/react-native/docs/getting-started.html/> Luettu 8.10.2016.

7 The Ionic Book. Verkkodokumentti. <Drifty Co. 2016. http://ionicframework.com/docs/guide/>. Luettu 2.10.2016.

8 Overview. Verkkodokumentti. Apache Foundation. 2015. <https://cordova.apache.org/docs/en/latest/guide/overview/index.html/>. Luettu 2.10.2016.

9 What is npm? Verkkodokumentti. NPM. 2016. <https://docs.npmjs.com/getting-started/what-is-npm/> Luettu 9.10.2016.

10 XCode on Windows: How to Develop for Mac or iOS on a PC. Verkkodokumentti. Nick Gibson. 2014 <https://blog.udemy.com/xcode-on-windows/> Luettu 9.10.2016.

11 What is WebView? Verkkodokumentti. Jen Looper. 2015. <http://developer.telerik.com/featured/what-is-a-webview//>. Luettu 2.10.2016.

12 Cordova iOS Performance Improvements: Drop-in Speed with WKWebView. Verkkodokumentti. Ionic Team. 2016. <http://blog.ionic.io/cordova-ios-performance-improvements-drop-in-speed-with-wkwebview//>. Luettu 8.10.2016.

13 HTML5. Verkkodokumentti. Wikipedia. 2016. <https://en.wikipedia.org/wiki/HTML5/>. Luettu 9.10.2016.

14 Cascading Style Sheets. Verkkodokumentti. Wikipedia. 2016. <https://en.wikipedia.org/wiki/Cascading_Style_Sheets/>. Luettu 15.10.2016.

15 AngularJS. Verkkodokumentti. Wikipedia. 2016. <https://en.wikipedia.org/wiki/AngularJS/>. Luettu 15.10.2016.

16 Guide. Verkkodokumentti. Google. 2016. <https://docs.angularjs.org/guide/>. Luettu 15.10.2016.

17 Angular-ui-router. Verkkodokumentti. Samuel Lijin. 2016.. <https://github.com/angular-ui/ui-router/wiki/> Luettu 16.10.2016.

18 ACTabScrollView. Verkkodokumentti. Azurechen. 2016. <https://github.com/azurechen/ACTabScrollView/>. Luettu 5.11.2016.