Tomi Kuivamäki

MOBIILISOVELLUKSEN TOTEUTUS IONIC-SOVELLUSKEHYKSESSÄ
Case: Aika24

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma

Toukokuu 2015

KUVAILULEHTI

MAMK University of Applied Sciences	**Opinnäytetyön päivämäärä** **8.5.2015**
Tekijä(t) Tomi Kuivamäki	**Koulutusohjelma ja suuntautuminen** Tietojenkäsittelyn koulutusohjelma

Nimeke

Mobiilisovelluksen toteutus Ionic-sovelluskehyksessä - Case: Aika24

Tiivistelmä

Opinnäytetyön tavoitteena on mobiilisovelluksen suunnittelu ja toteutus Aika24-ajanvarauspalvelun yritysasiakkaille. Sovelluksen tarkoitus on tarjota yrittäjille mahdollisuus uutena väylänä lisätä palveluja ja aikoja Aika24-palveluun.

Tutkimusongelmana on löytää toimeksiantajalleni ratkaisu sopivimman kehitysmenetelmään löytämiseen. Tutkimuksessa pitää ottaa huomioon yrityksen resurssit sovelluskehitykseen ja tarve olla rajoittamatta asiakaskuntaa eri käyttöjärjestelmien perusteella. Sovelluksen mahdollisina asiakkaina ovat sekä Aika24-palvelun asiakkaat että sovelluksen kautta palveluun liittyvät uudet asiakkaat. Tutkimuksen on tarkoitus kartoittaa yritykselle tuntemattomia tekniikoita ja menetelmiä, joita omaksumalla toimeksiantajani voi tulevaisuudessa keskittää resurssejaan tehokkaasti mobiilikehitykseen.

Työssä saavutetun tuloksen seurauksena toimeksiantajalle valmistui prototyyppi jatkokehitystä varten. Yritys aikoo jatkaa kehittämistä testiryhmän kautta saaman palautteen perusteella. Se toimii yritykselle myös toimintamallina, jossa pyritään sovelluksen ketterään kehitykseen Suomen johtaviin mobiilikäyttöjärjestelmiin.

Asiasanat (avainsanat)

mobiilisovellukset, JavaScript, sovelluskehykset, AngularJS, Ionic Framework

Sivumäärä 34	**Kieli** **Suomi**

Huomautus (huomautukset liitteistä)

Ohjaavan opettajan nimi Janne Turunen	**Opinnäytetyön toimeksiantaja** Aika24 Oy

DESCRIPTION

MAMK University of Applied Sciences	**Date of the bachelor's thesis** 8 May 2015
Author(s) Tomi Kuivamäki	**Degree programme and option** Business Information Technology

Name of the bachelor's thesis

Development of a mobile application in the Ionic Framework – Case: Aika24

Abstract

The subject of my thesis was to design and build a mobile application for the business customers of the aika24 online booking service. The application was meant to serve as a method for the businesses to add new available appointments to the Aika24 service. Purpose of this study was to search and study new techniques and methods which my client could use to focus their resources on mobile development efficiently.

The research problem was to find the most suitable solution for my client regarding mobile application development. In the study I had to consider the resources of the company for the application development and their wish not to restrict their customers by a specific platform. Potential users of this application were already customers of the Aika24 service and new customers who would join the service via application.

As a result of my thesis a prototype of the application was made for further development. The direction of the future development is determined by the feedback given by the application's test group. The prototype also functions as a template in which the application is developed with agile methods to be one of Finland's leading mobile operating systems.

Subject headings, (keywords)
mobile applications, JavaScript, frameworks, AngularJS, Ionic Framework

Pages 34	**Language** **Finnish**

Remarks, notes on appendices

Tutor Janne Turunen	**Bachelor's thesis assigned by** Aika24 Oy

SISÄLTÖ

1 JOHDANTO

Opinnäytetyöni aiheena on suunnitella ja toteuttaa mobiilisovelluksen prototyyppi Aika24-ajanvarauspalvelun yritysasiakkaiden käyttöön. Sovelluksen tarkoitus on olla keino yrittäjille lisätä varattavia aikoja ajanvarauspalveluun nopeasti ja mahdollisimman yksinkertaisesti. Sovelluksen on oltava myös yhteydessä jo olemassa olevan palvelun kanssa käyttämällä Aika24-rajapintaa, jonka kautta sovellus käy tiedonsiirtoa palvelun tietokantojen kanssa.

Opinnäytetyön toimeksiantaja on ajanvarauspalvelun omistava Aika24 Oy. Ajanvarauspalvelu on toiminut jo noin 10 vuoden ajan Smart Time Oy:n omistamana mutta siirtyi vuoden 2015 alusta samaan aikaan perustetulle Aika 24 Oy:lle. Omistajuusvaihdosta huolimatta palvelun kehittäjät pysyivät samana. Uuden yrityksen alaisena palvelua ollaan uudistamassa ja Aika24 Oy suunnittelee lisäksi uusia tapoja ja menetelmiä asiakkaidensa palvelun parantamiseksi. Opinnäytetyöni aiheena oleva sovellus on niistä ensimmäinen.

Opinnäytetyössä tutkitaan mobiilisovellusten erilaisia kehittämismenetelmiä ja niiden vaatimuksia. Työssäni myös valitsen yhden tavan, jota tarkastelen yksityiskohtaisemmin ja lopulta yritän soveltaa sitä toimeksiantajani tarpeisiin. Opinnäytetyön tavoitteena on suunnitella ja toteuttaa sovelluksen prototyyppi, josta yritys saa ponnahduslaudan jatkokehitykseen ja konkreettisemman käsityksen miten mobiilisovelluksia on yrityksen kannalta luontevin lähteä toteuttamaan.

Luvussa 2 käyn läpi yleisesti mobiililaitteiden käyttöjärjestelmiä ja niiden eroavaisuuksia sovelluskehityksen kannalta. Tutkin käyttöjärjestelmien markkinatilannetta ja mitä vaaditaan sovellusten rakentamiseen ja julkaisemiseen suosituimmilla alustoilla. Esittelen myös erilaiset sovellustyypit ja niiden toimintaperiaatteet.

Luvussa 3 otan lähempään tarkasteluun sovelluskehykset, jotka mahdollistavat hybridisovellusten rakentamisen web-teknologioilla. Etsin eri vaihtoehtoja ja tutkin niiden roolia hybridisovelluskehityksessä. Selvitän miten eri tekniikat toimivat ja mikä on niiden toimintaperiaate ja merkitys mobiilisovelluskehityksessä.

Luvussa 4 rakennan sovelluksen valitsemillani tekniikoilla. Selitän, miten sovellus rakennettiin, sen toiminallisuus ja sen tuomat haasteet. Tuon esille projektin tavoitteet ja miten niitä lähdettiin ratkomaan. Lopuksi käydään vielä läpi työn lopputulos ja sen onnistuminen.

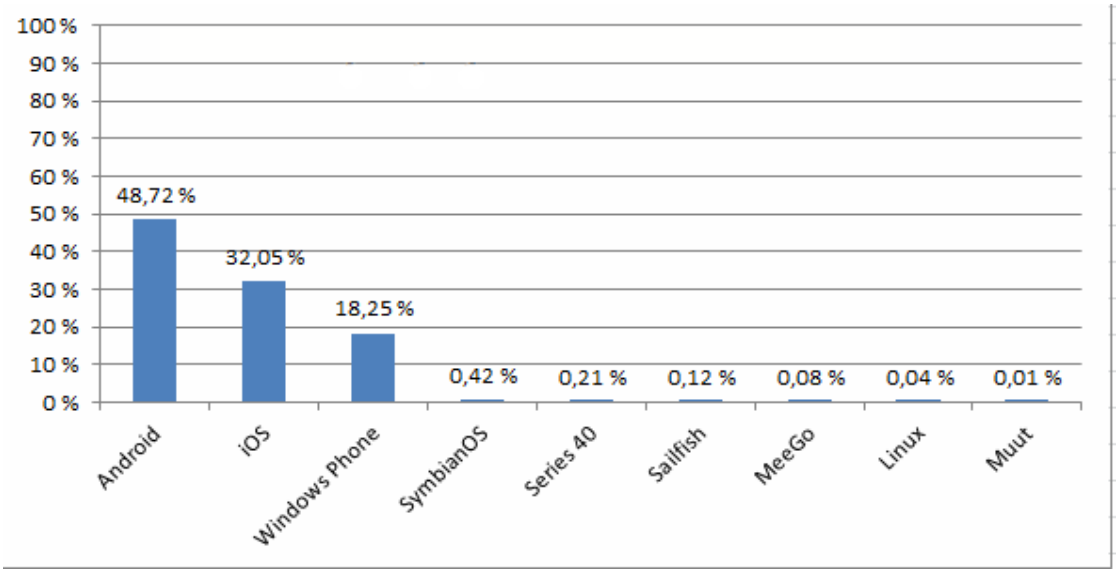
2 MOBIILISOVELLUKSET

Mobiilisovelluksista puhuttaessa tarkoitetaan älypuhelimilla ja tableteilla käytettäviä sovelluksia. Mobiilisovellukset yleensä hankitaan käyttöjärjestelmän sovelluskaupasta ja ne voivat olla joko ilmaisia tai maksullisia. Sovellusten käyttötarkoitus ja kohdeyleisö ovat monipuolisia. Sovellus voi olla ratkaisu yritysmaailman tarpeisiin tai toisaalta olla puhtaasti viihdettä, esim. peli.

Mobiilisovelluskehityksessä on otettava huomioon kohdekäyttöjärjestelmä, sillä jokainen käyttöjärjestelmä tarjoaa omat haasteet ja rajoitteet. Sovellusmarkkinoiden hajotessa eri käyttöjärjestelmien kesken on syntynyt erilaisia kehitystapoja ratkaisuksi kehittäjien tarpeelle saada sovellus mahdollisimman monelle kuluttajalle.

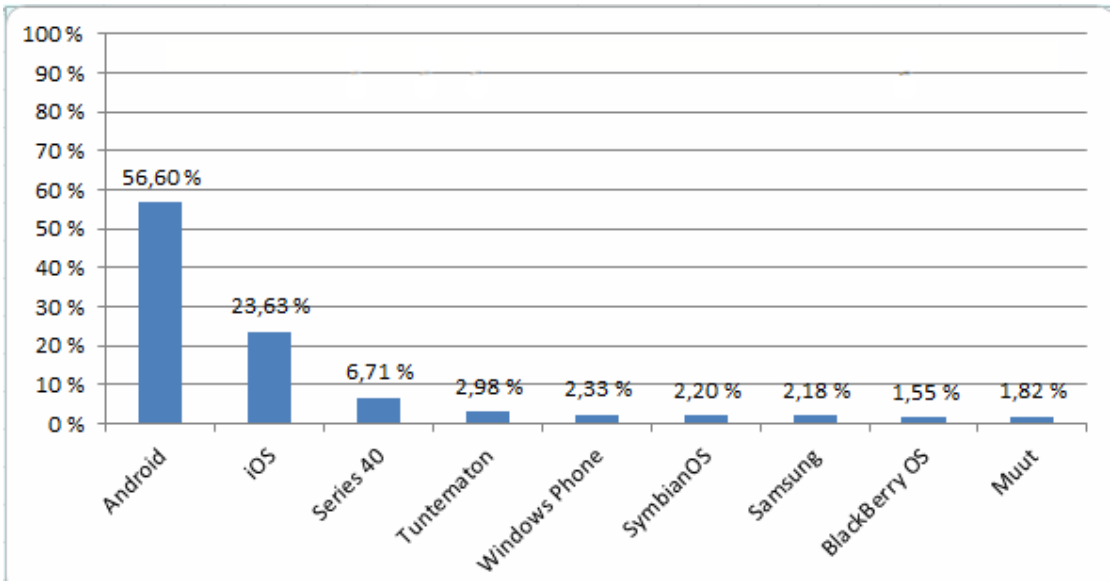
2.1 Käyttöjärjestelmät

Suomen älypuhelinmarkkinoilla on selvästi 3 yleisintä käyttöjärjestelmää. Suositumpana käyttöjärjestelmänä on Googlen Android, jota seuraavat Applen iOS ja Microsoftin Windows Phone. Muut käyttöjärjestelmät, kuten SymbianOs, Meego ja Linux, ovat jäänet marginaaliasemaan pienellä osuudellaan (kuva 1).



KUVA 1. Mobiilikäyttöjärjestelmät Suomessa maaliskuu 2014-huhtikuu 2015 (StatCounter 2015, mukaillen)

Kuvasta 2 käy ilmi että Suomen markkinatilanne heijastaa kansainvälistä tilannetta Androidin ja iOS:n pitävän hallussa kahden suosituimman käyttöjärjestelmän sijaa (IDC 2015). Suomessa kuitenkin Windows Phone on saanut hyvän markkinaosuuden sen ollessa Nokian Lumia-tuoteperheen käyttöjärjestelmä. Lisäksi Lumiat ovat erittäin yleisiä suomalaisessa yritysmaailmassa työpuhelimina (Marketvisio 2015).



KUVA 2. Mobiilikäyttöjärjestelmät maailmanlaajuisesti maaliskuu 2014-huhtikuu 2015 (StatCounter 2015, mukaillen)

Android on avoimeen lähdekoodiin perustuva Linuxiin pohjautuva käyttöjärjestelmä ja markkinaosuudeltaan suurin. Androidin sovelluksien jakelukanava on Google Play-

sovelluskauppa. Android-sovellusten natiivikieli on Java, ja niiden kehitykseen tarvitaan Android SDK. SDK sisältää tarvittavat kehitystyökalut sovellusten rakentamiseen, kuten kirjastot, debuggerin, virallisen dokumentaation ja emulaattorin, jolla voi emuloida puhelimen toimintoja (Android Developer 2015).

Applen iOS on Applen valmistamiin älypuhelimiin ja tabletteihin tarkoitettu käyttöjärjestelmä. iOS julkaistiin 2007 Applen iPhonen yhteydessä ja on vuosien saatossa levinnyt käyttöjärjestelmäksi Applen muihin tuotteisiin kuten iPodiin ja Apple TV:hen (Apple 2015) OS X:n ollessa vielä käyttöjärjestelmä Applen työpöytäkoneissa.

Applen jakelukanava iOS-sovelluksille on App Store, jonka kautta Apple valvoo sovellusten laatua ja tarjoaa niitä loppukäyttäjille. Sovellusten kehittämiseen tarvitaan iOS SDK, jossa on tarvittavat työkalut iOS:lle kehitettäville sovelluksille. Kehittäjien on kuitenkin liityttävä Applen maksulliseen iOS Developer Programiin, joka antaa kehittäjälle oikeuden testata sovellusta Applen laitteella ja laittaa sovelluksen ladattavaksi App Storeen (Apple Developer 2015). iOS-sovelluksissa ohjelmointikielenä on Objective-C ja ohjelmointiympäristönä Xcode.

Windows Phone on Microsoftin kehittämä käyttöjärjestelmä, joka on seuraaja Microsoftin aiemmalle käyttöjärjestelmälle, Microsoft Mobile. Windows Phonen ensijulkaisu oli Windows Phone 7 vuonna 2010 ja sen nykyinen versio on 8.1. Yksi käännekohdista Windows Phonen kehityksessä on 2011 alkanut Microsoftin ja Nokian välinen yhteistyö. (Järvinen 2012, 12.) Yhteistyön myötä Windows Phonesta tuli Nokian älypuhelimien ensisijainen käyttöjärjestelmä syrjäyttäen Symbianin. Käyttöjärjestelmä otettiin ensi kertaa käyttöön Nokian laitteissa Lumia-tuoteperheessä. Yhteistyön seurauksena Microsoft osti Nokian matkapuhelinliiketoiminnan 2014.

Windows Phone SDK sisältää Windows Phonen sovelluskehitystyökalut. Windows Phone-sovelluksien ohjelmointikielenä toimivat Visual C#, Visual Basic tai Visual C++:n ja kehitysympäristönä toimii Visual Studio (MSDN 2015). Saadakseen sovelluksen käyttöjärjestelmän sovelluskauppaan, Windows Phone Storeen, on kehittäjän rekisteröityvä Microsoftin sovelluskehittäjäksi, mikä on maksullista.

2.2 Sovellustyypit

Natiivisovellus on sovellus, joka on kehitetty tietylle käyttöjärjestelmälle ja on ladattavissa sen omasta sovelluskaupasta. Natiivit sovellukset kehitetään käyttöjärjestelmän omalla SDK:lla ja sen vaatimalla ohjelmointikielellä. SDK:t ja kehitysympäristöt ovat hankittavissa käyttöjärjestelmän valmistajalta.

Natiivisovelluksen edut ovat sen nopeus, suorituskyky ja sen yhteensopivuus käytettävän laitteen kanssa. Sovellus pääsee käsiksi kaikkiin laitteen toimintoihin suoraan niiden API:n (application programming interface) eli ohjelmointirajapinnan kautta. (Salesforce Developers 2015.) Esimerkiksi jokaisella käyttöjärjestelmällä on oma API laitteen kameraa varten. Lisäksi natiivisovellus ei tarvitse toimiakseen internetyhteyttä vaan se toimii laitteista käsin.

Sovelluskehityksen kannalta natiivisovellus on aikaa vievää ja kallista. Useammalle käyttöjärjestelmälle tehtävä sovellus on aina kehitettävä uudestaan jokaista järjestelmää kohden (Gambit 2015). Lisäksi sovellusten käyttäjien on aina tarvittaessa ladata sovelluksen päivitykset.

Web-sovellukset ovat laitteen selaimessa avattavia HTML-dokumentteja, jotka on suunniteltu ja optimoitu mobiilikäyttöä varten. Sovellukset kehitetään perinteisillä www-tekniikoilla, HTML5, JavaScript ja CSS3. Web-sovellukset eivät tarvitse käyttöjärjestelmäkohtaisia SDK:ta, vaan ne toimivat mobiililaitteen selaimessa normaalin www-sivun tapaan (Budiu 2013). Tämä mahdollistaa sovelluksen kehittämisen usealle laitteelle yhdellä ohjelmakoodilla ja poistaa tarpeen luoda uutta lähdekoodia toiselle käyttöjärjestelmälle. Sovellusten päivittäminen onnistuu ilman käyttäjän osallistumista, sillä päivitys tehdään sovelluksen käyttämällä palvelimella (Summerfield 2015).

Web-sovellukset tarvitsevat kuitenkin aina toimiakseen internetyhteyden koska niillä ei ole pääsyä laitteen tallennustilaan (Natili 2013, 35). Web-sovelluksilta on yleensä myös puuttunut ennen uusia HTML5-ominaisuuksia oikeus laitteen muihin toimintoihin (Mobile HTML5 2015), joka rajoittaa suuresti sovelluksen toiminnallisuutta.

Hybridisovellus on nimensä mukaan yhdistelmä kahdesta edellisestä tyypistä. Hybriditekniikassa sovellus rakennetaan web-tekniikoilla niin paljon kuin mahdollista natiivisovelluksen kaltaiseksi.

Toisin kuin web-sovellus, hybridisovellusta ei ajeta laitteen selaimessa vaan se ladataan natiivisovelluksen tavoin sovelluskaupasta. Hybridisovellukset käyttävät laitteen selainmoottoria luomalla web-näkymän (web view), jonka kautta sovellus voi ajaa HTML:ää ja JavaScriptiä. (Lehdonvirta & Korpela 2013, 125-126.) Hybridisovellukset voivat myös hyödyntää laitteen natiiveja ominaisuuksia abstraktointikerroksen avulla.

Hybridisovelluskehityksessä abstraktointikerroksen tarjoavat erilaiset JavaScript-sovelluskehykset kuten Cordova/PhoneGap. Nämä toimivat siltana web-pohjaisen sovelluksen ja laitteen välillä. Kehykset antavat sovellukselle pääsyn käyttöjärjestelmän ohjelmointirajapintoihin JavaScriptillä ilman laitekohtaista natiivikieltä (PhoneGap 2015).

Hybridisovellukset ovat yleistyneet viimevuosina niiden tarjoamien mahdollisuuksien ansiosta. Sovelluskehittäjät voivat luoda sovelluksia, jotka ovat toiminnallisuudeltaan lähellä natiivisovelluksia mutta kuitenkin rakennetaan web-tekniikoilla. Tämä nopeuttaa sovelluksen kehittämistä ja testausta ja mahdollistaa sovelluksen toimivuuden usealla alustalla samalla lähdekoodilla (Rudolph 2014).

3 HYBRIDISOVELLUSKEHITYS

Hybridisovelluskehitys yrittää antaa ratkaisuja kehittäjien ongelmiin liittyen usealle käyttöjärjestelmälle rakennettavan sovelluksen kehitystyöhön. Hybridisovelluskehitykseen löytyy useita valmiita kehyksiä, joista monet nojaavat perinteisten web-teknologioiden varaan. Silti niistä löytyy monipuolisuutta ja uusia kehyksiä on tullut mukaan sovelluskehitykseen.

3.1 Cordova

Apache Cordova on avoimen lähdekoodin mobiilisovelluskehys, jolla voi kehittää sovelluksia eri alustoille web-tekniikoita käyttäen. Cordova koostuu rajapinnoista jot-

ka antavat sovelluskehittäjille pääsyn laitteen natiiveihin toimintoihin, kuten kameraan ja kiihtyvyysmittariin JavaScriptin avulla (Apache Cordova 2015). Tämän ansiosta sovellus voidaan rakentaa ilman käyttöjärjestelmän vaatimaa natiivikoodia.

Cordovan kehitys on tiiviisti kytköksissä toiseen sovelluskehykseen nimeltään PhoneGap. PhoneGap on nykyään Adobe Systemsin omistama kehys, joka sai alkunsa Nitobi-nimisen yrityksen projektina luoda keino päästä kiinni laitteen ympäristöön natiivisovellukseen sulautetun web-näkymän kautta, mikä mahdollistaisi sovelluksen kehittämisen puhtailla web-teknologioilla. Adoben ostettua Nitobin vuonna 2011 lahjoitti Nitobi PhoneGapin lähdekoodin Apache Software Foundationille, varmistaakseen sen avoimuuden. (Ionic Blog 2014.) Tämän johdosta Corvova ja PhoneGap ovat lähes identtisiä kehyksiä.

Cordova-pohjaisen sovelluksen toiminnallisuus ja ominaisuudet määritellään projektin config.xml-tiedostossa (kuva 3). Tiedostossa on sovelluksen tiedot, kuten tunniste, nimi, kuvaus ja sisällön lähde (Apache Cordova 2014). Sisällön lähteenä on oletuksena index.html, joka suoritetaan natiivisovelluksen sisällä sulautetun web-näkymän kautta.

```
<?xml version='1.0' encoding='utf-8'?>
<widget id="fi.mamk" version="0.0.1" xmlns="http://www.w3.org/ns/widgets" xmlns:cdv="http://cordova.apache.org/ns/1.0">
    <name>Cordova Sovellus</name>
    <description>
        Esimerkki Cordova-sovellus.
    </description>
    <author email="dev@cordova.apache.org" href="http://cordova.io">
        Apache Cordova Team
    </author>
    <content src="index.html" />
    <access origin="*" />
</widget>
```

KUVA 3. Cordova-projektin config.xml-tiedosto

Toimiakseen natiivin sovelluksen lailla, pitää jokaista käyttöjärjestelmää kohden asentaa sovellukselle tuki Cordova-projektiin. Cordova tukee kaikkia yleisimpiä käyttöjärjestelmiä. Tuen lisäksi tarvitaan käyttöjärjestelmän oma SDK ja kehitysympäristö.

Cordovan liitännäiset (plugin) tarjoavat rajapinnan sovelluksen web-näkymän ja laitteen käyttöjärjestelmän välille, mahdollistaen pääsyn laitteen natiiveihin ominaisuuk-

siin. Cordovaan saatavat liitännäiset löytyvät Cordova Plugin Registry-nimisestä palvelusta, jonka kautta voi hankkia Cordova-projektiin tarvittavat liitännäiset. Palvelusta löytyy liitännäisiä useisiin laitteen ominaisuuksiin, kuten kameran, GPS:n tai kiihtyvyysmittarin hallintaan. Liitännäiset koostuvat JavaScript- ja alustan natiiviohjelmakoodi-tiedostoista. Natiivikoodi piilotetaan JavaScript-käyttöliittymän taakse (MSDN 2015), jonka kautta sovellus voi kutsua laitteen toimintoja (kuva 4).

```
navigator.geolocation.getCurrentPosition(
        function (geoData) {
            //laite vastaa
        },
        function (virhe) {
            // laite ei vastaa
        },
        {
            enableHighAccuracy: true
        }
        );
```

KUVA 4. Paikkatieto-liitännäisen alustaminen

Cordova tarjoaa laiteriippumattoman sovelluskehyksen, jonka avulla HTML5-pohjaiset sovellukset voidaan kääntää natiiveiksi sovelluksiksi, jotka voidaan tarjota levitykseen sovelluskauppaan. Cordovassa itsessään ei ole mukana käyttöliittymäkirjastoa, siihen tarvitaan kolmannen osapuolen tarjoamaa ratkaisua. Näitä ovat esimerkiksi Sencha Touch, Dojo Mobile tai jQuery Mobile.

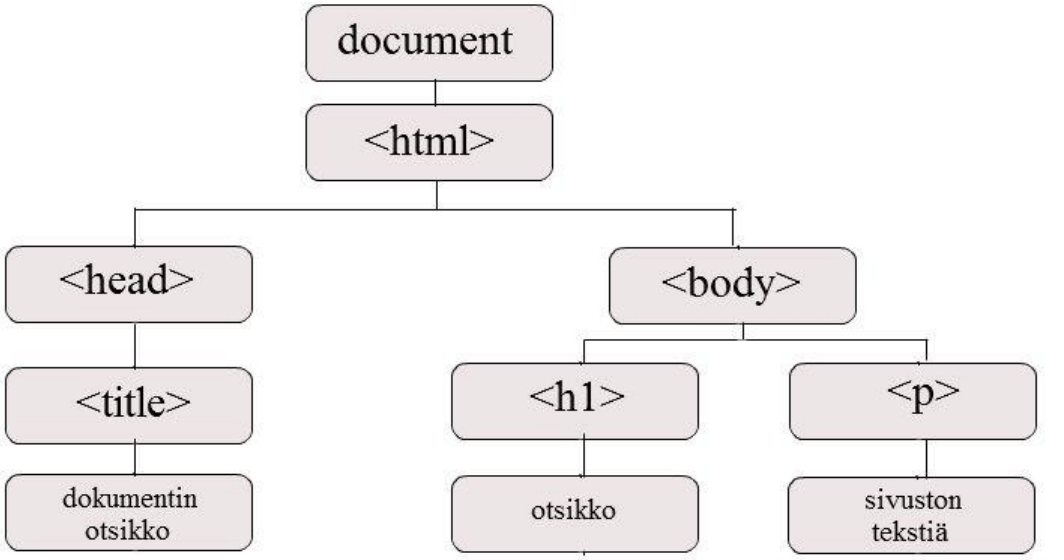
3.2 jQuery Mobile

jQuery Mobile on sovelluskehys suunniteltu mobiilisovelluksien ja sivujen käyttöliittymien kehittämiseen web-tekniikoilla. Se tarjoaa kehittäjille mahdollisuuden luoda käyttöliittymä, joka toimii useilla alustoilla ja laitteilla yhdellä lähdekoodilla (jQuery Mobile 2015). jQuery Mobile on yksi käytetyimmistä kehyksistä hybridisovelluskehityksessä.

jQuery Mobile perustuu vuonna 2006 julkaistuun jQuery JavaScript-kirjastoon. jQueryn tehtävänä on yksinkertaistaa JavaScriptin syntaksia ja varmistaa ohjelman toimivuuden eri selaimilla (Lehdonvirta & Korpela 2013, 59), helpottaa HTML-

dokumentin manipulointia, tapahtumien käsittelyä ja Ajax-kutsuja (jQuery 2015). Yksi oleellisimmista jQueryn ominaisuuksista on DOM-puun lukeminen ja manipulointi.

DOM (Document Object Model) eli dokumenttioliomalli on dokumentin esitystapa, jossa dokumentti ja sen sisältämät elementit esitetään puun kaltaisena rakenteena (Stefanov & Sharma 2013, 227). HTML-dokumentin tapauksessa DOM mahdollistaa vuorovaikutteisten www-sivujen luonnin ilman palvelinyhteyttä ja sivun uudelleen latausta. Selaimessa DOM-manipulaatio suoritetaan JavaScriptillä. Kuvassa 5 näkyy DOM-puun esimerkki.



KUVA 5. Esimerkki DOM-puusta

jQuery helpottaa DOM-puun manipulointia sen JavaScriptiä lyhemmällä syntaksilla ja laajalla rajapinnalla, jonka avulla pystyy helposti lukemaan DOMin elementit. Rajapinta tarjoaa myös kattavan kokoelman funktioita elementtien manipulointiin, kuten elementtien arvon muokkaamiseen (jQuery 2015). Kuvassa 6 näytetään miten jQuerylla asetetaan HTML-dokumentin otsikolle teksti Hello World! normaaliin JavaScriptiin verrattuna. Juuri tämä JavaScriptin syntaksin helpottaminen ja samalla sen tehokkuuden säilyttäminen on yksi jQueryn päätavoitteista.

```
document.getElementsByTagName("h1")[0].innerHTML = "Hello World!";
```

JavaScript

```
$("h1").html("Hello world!");
```

jQuery

KUVA 6. Elementtien manipuloinnin ero JavaScriptillä ja jQuerylla

Halutun DOM-elementin etsiminen ja valitseminen tapahtuu valitsimen (selector) avulla (kuva 7). Elementtejä voidaan etsiä DOM-puusta esimerkiksi elementin id:n, luokan (class) tai tagin perusteella. Syntaksiltaan se muistuttaa CSS:n käyttämiä valitsimia ja varastoi saadun elementin jQuery-objektiin. (Duckett 2014, 302-306.) jQuery-objektin avulla on mahdollista muokata haluttua elementtiä käyttämällä jQueryn metodeja.

```
Id:n perusteella:          $("#id");
Luokan perusteella:        $(".luokka");
Tagin perusteella:         $("button");
Kaikki elementit:          $("*");
```

KUVA 7. Esimerkkejä jQueryn valitsimista

Yksi JavaScriptin tehtävistä on kuunnella ja käsitellä selaimessa tapahtuvia tapahtumia (event). Tapahtumat yleensä käynnistyvät selaimen reagoidessa käyttäjän tekemään valintaan kuten linkin painamiseen, hiiren liikuttamiseen ja sivun lataukseen. Tapahtuman käynnistyttyä suoritetaan kehittäjän haluamaa ohjelmakoodia. (Stefanov & Sharma 2013, 250-251.) Tätä prosessia kutsutaan tapahtumienkäsittelyksi.

jQueryssa on useita metodeja tarkoitettu selaimen tapahtumien käsittelyyn ja ovat osa jQueryn kehittäjille tarjoamista mahdollisuuksista helpottamaan JavaScriptin ominaisuuksien käyttöä. Tapahtumankäsittely on JavaSriptissä ja jQueryssa samanlainen tapahtuman kulun kannalta mutta eroaa jQueryn omien sisäänrakennettujen metodien kohdalta. Haluttuun elementtiin lisätään tapahtumankuuntelija, jonka kautta ohjelma tietää minkälaiseen tapahtumaan elementin pitää reagoida. Kuvassa 8 elementti on valittu id:n kautta ja se reagoi jos elementtiä klikataan.

```
$("#id").click(function () {
    //Ohjelmakoodi, joka ajetaan jos elementtiä klikataan
});
```

KUVA 8. Tapahtumankäsittely jQueryssa

jQuery auttaa kehittäjiä tekemään sivuista vuorovaikutteisimpia käyttämällä Ajax-tekniikkaa. Ajaxin (asynchronous JavaScript and XML) avulla sovellus voi kommunikoida palvelimen kanssa ilman erillistä sivunlatausta. Tiedonvaihto tapahtuu XMLHttpRequest-objektin välityksellä http-metodien avulla ja tiedostomuotona voi olla esim. XML, JSON ja HTML. jQuery tarjoaa valmiita Ajax-metodeja, jotka yksinkertaistavat palvelinpyyntöjä ja ratkaisevat Ajaxin selainkohtaiset eroavaisuudet (Lehdonvirta & Korpela 2013, 55).

```
$.ajax({
      url: "esimerkki.json",
      dataType: "json",
      success: function (data) {
          //Suoritetaan jos Ajax-kutsu onnistuu
      },
      error: function () {
          //Suoritetaan jos Ajax-kutsu epäonnistuu
      }
  });
```

KUVA 9. Esimerkki Ajax-kutsu JSON-tiedostosta

Ajax on luonteeltaan epäsynkroninen, mikä tarkoittaa että Ajax-kutsu palvelimelle ei keskeytä ajettavaa lähdekoodia vaan pyyntö tapahtuu taustalla. Palvelimen palautettua tiedot käynnistyy tapahtuma, jota Ajax-kutsuissa yleensä käytetään funktion kutsumiseen. (Duckett 2014, 371.) Tätä funktiota kutsutaan callback-funktioksi ja sitä käytetään palvelimelta saadun tiedon keräämiseen. Kuvassa 9 esitetään jQuerylla tehty Ajax-kutsu, jossa käytetään metodin omaa sisäänrakennettua callback-funktiota, success tai error riippuen pyynnön onnistumisesta.

jQuerysta on tullut yksi yleisimmistä web-kehityksissä käytetyistä JavaScript-kirjastoista. jQueryn kehityksestä vastaava jQuery Foundation on lisäksi julkaissut jQuery Mobilen, joka on rakennettu jQuery-kirjaston päälle ja vaatii toimiakseen sovelluksessa jQueryn. jQuery Mobile antaa kehittäjille työkalut mobiilikäyttöliittymien

rakentamiseen, pitäen sisällään Ajax-pohjaisen navigaation, tuen kosketusnäytön vaatimiin toimintoihin, sisäänrakennetut ikonit ja widgeteihin (jQuery Mobile 2015).

jQuery Mobilen widgetit ovat valmiita käyttöliittymäkomponentteja, jotka ovat optimoitu mobiilikäyttöä varten. Widgetteihin sisältyy useita DOM-elementtejä, kuten button, checkbox ja listview. Jokaisella widgetillä on sille omat määritetyt metodit ja tapahtumat. Widgettien avulla voidaan luoda responsiivisia ja interaktiivisia elementtejä, jotka ovat optimoitu kosketusnäytöllisiin laitteisiin.

jQuery Mobilen CSS Framework on käyttöliittymän elementeille tarkoitettu tyyliohje, joka yksinkertaistaa CSS-tyylien käyttöä mobiiliympäristössä. Tyyleihin voi vaikuttaa suoraan upottamalla HTML-dokumentin widgetteihin CSS Frameworkin edellyttämää syntaksia. Useisiin widgetteihin voi myös käyttää jQuery Mobilen sisäänrakennettuja ikoneja.

jQuery Mobilella sivunäkymät luodaan data-attribuutin avulla. Attribuutilla määritellään sivu ja erotellaan sivuun kuuluvat osat (Kauppi 2014, 9). Kuvassa 10 näkyy yksittäisen jQuery Mobile-sivun luonti data-attribuutin avulla.

```
<!DOCTYPE html>
<html>
    <head>
        <title>jQuery Mobile-sivu</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <link rel="stylesheet" href="css/jquery.mobile-1.4.5.min.css">
        <script src="js/jquery.js"></script>
        <script src="js/jquery.mobile-1.4.5.min.js"></script>
    </head>
    <body>
        <div data-role="page" id="etusivu">
          <div data-role="header" data-position="fixed">
              jQuery Mobile-sivun yläpalkki
          </div>
          <div data-role="main" class="ui-content">
              jQuery Mobile-sivun sisältö
           </div>
        </div>
    </body>
</html>
```

KUVA 10. jQuery Mobile-sivu

Yhdessä HTML-dokumentissa voi olla useita jQuery Mobile-sivuja, jotka erotetaan toisistaan sivun yksilöllisen tunnisteen avulla. Tunnisteiden kautta voidaan luoda si-

säinen linkitys DOM-puussa sivujen välille. Lisäksi uusia sivuja voidaan hakea DOM-puuhun käyttämällä Ajax-kyselyjä.

3.3 Ionic

Ionic on sovelluskehys tarkoitettu hybridisovelluskehitykseen. Ionic koostuu HTML, CSS ja JavaScript-komponenteista, jotka ovat optimoitu mobiilikäyttöön. Ionic on kehittäjiensä mukaan omistettu nimenomaan hybridisovelluksien rakentamiseen, ei mobiiliwebsivujen kehittämiseen, kuten esimerkiksi jQuery Mobile. Toisin sanoen Ionic-sovellukset ovat tarkoitus ajaa laitteen selainmoottorissa, ei selaimessa (Ionic 2015).

Ionic on rakennettu Cordovan päälle, jota se tarvitsee sovelluksen paketoimiseen toimiakseen natiivin sovelluksen lailla. Tämän seurauksena Ionic-projektin rakenne muistuttaa Cordovaa. Cordovan lailla projektiin pitää lisätä sovelluksen haluttu käyttöjärjestelmä ja sovellukseen on lisättävissä liitännäisiä laitteen erikoisominaisuuksia varten. Tuettuja alustoja Ionicissa ovat iOS ja Android.

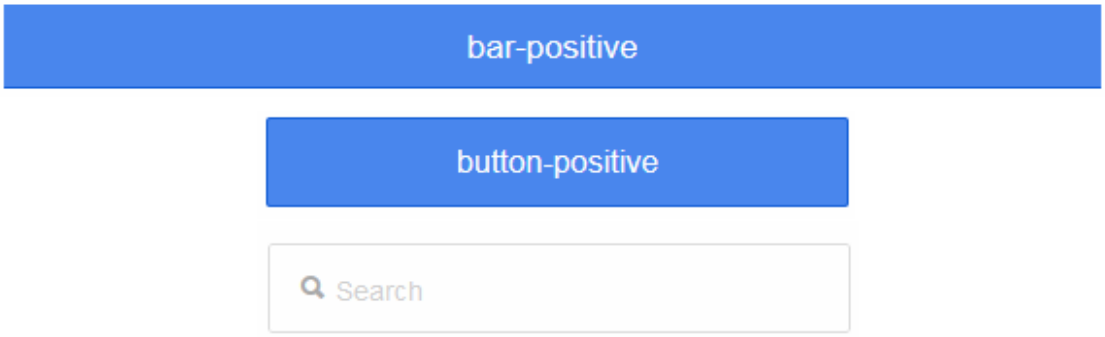
Ionicin sovelluslogiikka on rakennettu AngularJS:llä (Coenraets 2014). JavaScript-rajapinnan avulla voi määritellä käyttöliittymäkomponenttien toimintaa, rakentaa sovelluksen näkymiä ja mahdollistaa navigointi niiden välillä sekä hallinnoida mahdollista tietoliikennettä sovelluksen ja ulkoisen palvelimen välillä. Rajapinta koostuu AngularJS-pohjaisista direktiiveistä ja serviceista.

Ionicin näkymät toimivat yksisivuisen sovelluksen mallin mukaisesti. Sovellus koostuu HTML-dokumentista, joka toimii sovelluksen näkymänä. Näkymän sisältö riippuu missä tilassa sovellus on, eli mihin sovelluksen URL osoittaa. Näkymän sisältö saadaan ulkoisista HTML-tiedostoista, jotka sisällytetään päänäkymään. (Learn Ionic 2014) Ionicin näkymät rakennetaan ionNavView-direktiivillä (kuva 11).

```
<ion-nav-bar></ion-nav-bar>
<ion-nav-view>
  <ion-view view-title="Etusivu">
    <ion-content>
      Sivun sisältö
    </ion-content>
  </ion-view>
</ion-nav-view>
```

KUVA 11. Ionic-näkymä

CSS-komponentit ovat käyttöliittymäelementtejä, jotka ovat muokattavissa ja rakennettu mobiilikäyttöä varten. CSS-komponenttien ydin on rakennettu käyttämällä Sass:ia, laajennuskieltä CSS:ään. Sass tuo uusia ominaisuuksia CSS-kieleen, kuten muuttujat, perintää ja valitsimien sisäistäminen (Lehdonvirta & Korpela 2013, 98; Sass 2015). Ionicin elementteihin sisältyy mm. painikkeita, palkkeja ja erilaisia tekstikenttiä (Kuva 12).



KUVA 12. Ionicin käyttöliittymäelementtejä

Ionicin mukana tulee kokoelma valmiita ikoneja, nimeltään Ionicons (kuva 13). Ikoneja voi käyttää erilaisissa käyttöliittymäelementeissä, esim. painikkeissa, tekstikentissä ja ja tabeissa. Ikoneista löytyy alustariippumattomia ikoneita sekä Androidille ja iOS:lle suunniteltuja ikoneita. Ionic ei kuitenkaan rajoita käyttämään pelkästään sen omia ikoneita, vaan kehittäjä voi lisätä projektiin omia ikoneita.

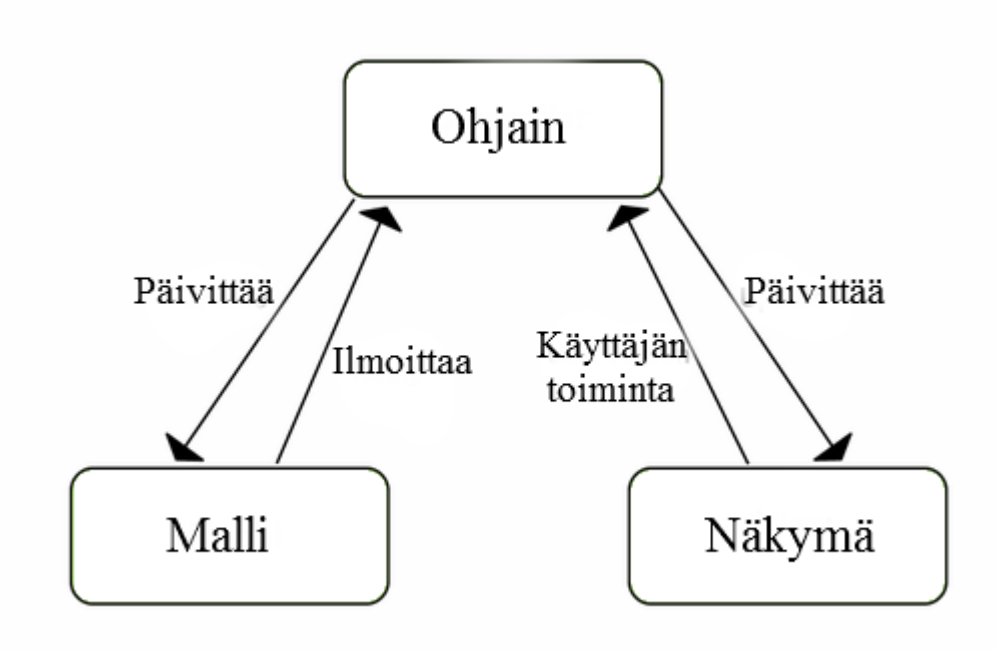
KUVA 13. Esimerkkejä Ionicons-ikoneista

Ioniciin on lisättävissä alustakohtaisia liitännäisiä laitteen erikoisominaisuuksien hallitsemiseksi. Ionic käyttää Cordovan liitännäisiä, jotka asennetaan projektiin ja ne tarvitsevat toimiakseen ngCordovan. ngCordova on Ionicin projekti, joka tarjoaa suosituille Cordova/Phonegap-liitännäisille AngularJS-kääreen (ngCordova 2015). Kääreen avulla liitännäisiä voi hallita Ionic-projektissa käyttämällä AngularJS:ää.

3.4 AngularJS

AngularJS on Googlen ylläpitämä avoimeen lähdekoodiin perustuva JavaScript-ohjelmistokehys, joka on tarkoitettu yksisivuisten web-sovellusten kehittämiseen. AngularJS on viime vuosina noussut suosituksi web-sovellusten ohjelmistokehykseksi muiden perinteisten JavaScript-pohjaisten kehyksien, kuten jQueryn rinnalle. Se on myös omaksuttu mobiilisovelluskehityksessä, kuten Ionic-sovelluskehyksessä.

AngularJS antaa kehittäjälle mahdollisuuden hyödyntää MVC-arkkitehtuuria (Model View Controller), jonka tarkoituksena on erottaa sovellus kolmeen eri osaan (kuva 14). Arkkitehtuuri on yleinen web-sovelluksissa koska se erottaa käyttöliittymän sovelluslogiikasta (Lehdonvirta & Korpela 2013, 66; Raasch 2013, 7). Näin käyttäjä ei pääse suoraan käsiksi sovelluksen kaikkiin tietoihin vaan hänelle rajataan vain oleellisin ja tarvittava tieto, joka näytetään näkymässä.



KUVA 14. MVC-arkkitehtuuri (Chrome Developer 2015, mukaillen)

Malli pitää sisällään sovelluksen toiminnot ja tiedonhallinnan. Näkymä, jonka sovelluksen käyttäjä näkee, riippuu sovelluksen tilasta ja on riippuvainen mallin sisällä pitämistä tiedoista. Ohjain yhdistää käyttäjän ja sovelluksen lukemalla käyttäjän antamia syötteitä. (Chrome Developer 2015.) Ohjain muokkaa mallin sisältämää tietoa, joka puolestaan muuttaa näkymää.

AngularJS toimii ensin lukemalla sovelluksen HTML-dokumentin, josta se etsii AngularJS:n omia attribuutteja. Nämä attribuutit synnyttävät direktiivejä, joita AngularJS:ssä kutsutaan ng-direktiiveiksi (Duckett 2014, 435). Direktiivien avulla DOM-elementtiin sisällytetään ominaisuuksia ja toimintoja, direktiivistä riippuen. AngularJS:ssä voi HTML:ään upottaa ilmaisuja (expressions) käyttämällä syntaksia {{ ilmaisu }}. Niissä voi käsitellä esimerkiksi merkkijonoja, lukuja tai direktiiviin sidottua tietoa (kuva 15).

Nimi: Matti

Hello World!

2

Terve Matti!

```
<!doctype html>
<html ng-app>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.3/angular.min.js"></script>
  </head>
  <body>
    <div>
      <label>Nimi:</label>
      <input type="text" ng-model="sinunNimesi">
      <p>{{ "Hello World!" }}</p>
      <p>{{ 1 + 1 }}</p>
      <p>Terve {{ sinunNimesi }}!</p>
    </div>
  </body>
</html>
```

KUVA 15. AngularJS:n ilmaisuja

Scope, AngularJS:ssä $scope, on JavaScript-objekti, joka sisältää mallin tiedot ja logiikan ja se vaikuttaa suoraan näkymään (ng-newsletter 2014). Scopet alustetaan ja hallinnoidaan ohjaimien avulla ja ne ovat ohjainkohtaisia. Alustus tapahtuu antamalla $scope parametrina ohjaimelle. Scope näkee ohjaimen kautta näkymässä olevat direktiivit ja rekisteröi muutokset, jonka kautta direktiivi pystyy päivittämään tiedot DOM-puuhun. Scopeen voi myös määritellä funktiota (kuva 16).

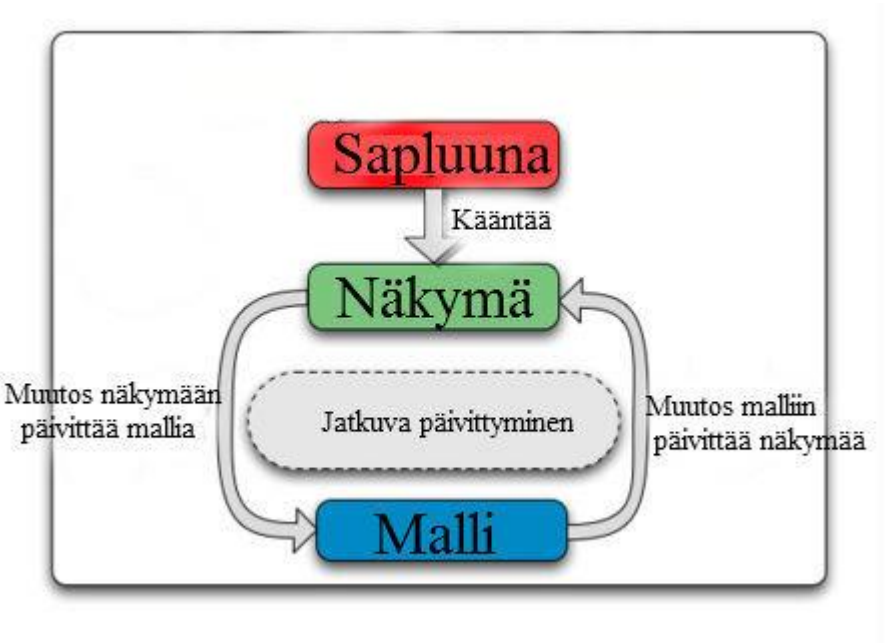
Etunimi: Matti Sukunimi: Meikäläinen

Matti Meikäläinen

```
<!doctype html>
<html ng-app="EsimerkkiApp">
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.3/angular.min.js"></script>
  </head>
  <body>
    <div ng-controller="EsimerkkiController">
        <label>Etunimi:</label>
        <input type="text" ng-model="etunimi">

        <label>Sukunimi:</label>
        <input type="text" ng-model="sukunimi"><hr>
        {{ tulostaNimi() }}
    </div>
  </body>
</html>
<script>
var EsimerkkiApp = angular.module('EsimerkkiApp', []);
EsimerkkiApp.controller('EsimerkkiController', ['$scope', function($scope) {
    $scope.etunimi = "";
    $scope.sukunimi = "";

    $scope.tulostaNimi = function() {
        return $scope.etunimi + " " + $scope.sukunimi;
    }
}]);
</script>
```

KUVA 16. Scope-objektin alustus ja funktion määrittely ohjaimessa

Ohjaimet kontrolloivat sovellusta liittämällä ne sovelluksen näkymiin. Ohjaimia on yleensä useampia ja ne toimivat vain niille määrätyille näkymillä. Kuten edellä mainittiin, ohjaimet alustavat ja hallinnoivat scope-objekteja. Ohjaimilla voidaan lisätä metodeja scopeen, joita voidaan kutsua näkymästä (AngularJS 2015). Näin sovellus voi lukea käyttäjän antamia syötteitä ja päivittää näkymää. Tätä kutsutaan AngularJS:ssä kaksisuuntaiseksi tietojen sitomiseksi (two-way data binding). Siinä malli ja näkymä ovat yhteydessä toisiinsa eli toisin sanoen toista muokattaessa toinen päivittyy (kuva 17).



KUVA 17. AngularJS:n kaksisuuntainen tietojen sitominen (AngularJS 2015, mukaillen)

Servicet ovat AngularJS:ssä objekteja tai funktioita, jotka suorittavat sille määrätyn tehtävän. Niitä voi käyttää sovelluksen HTTP-kutsujen toteuttamiseen. (Patel 2014) AngularJS:ssä on sisäänrakennettuja serviceitä, kuten $http mutta sovellukseen voi rakentaa omia serviceitä. Servicen toimintoja voi kutsua muualta ohjelmasta, kuten ohjaimesta niitä tarvittaessa.

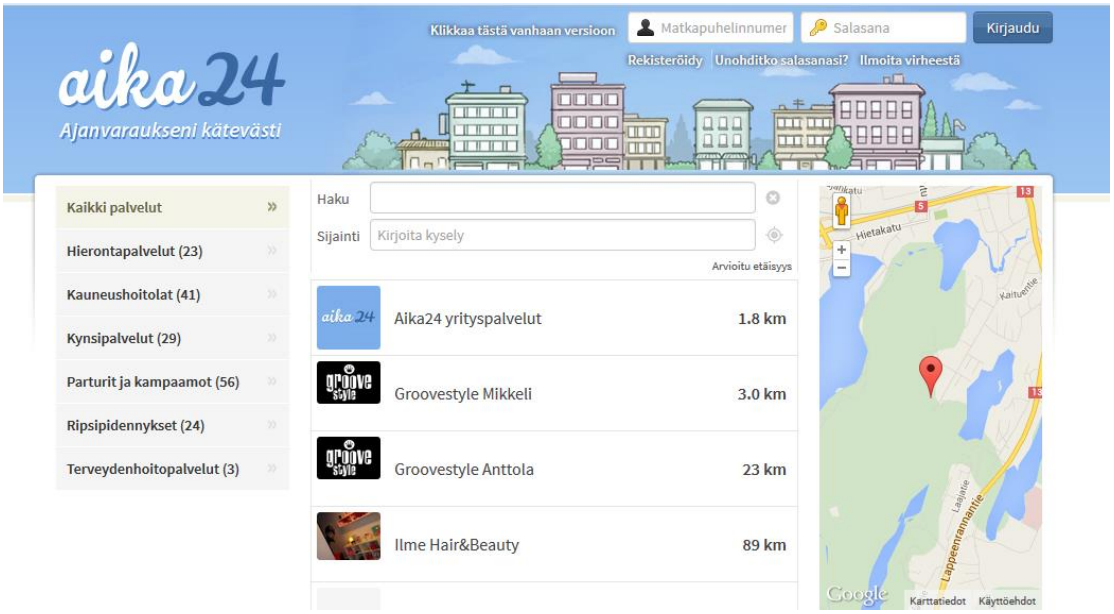
```
<!doctype html>
<html ng-app="EsimerkkiApp">
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.3/angular.min.js"></script>
  </head>
  <body>
    <div ng-controller="EsimerkkiController">
        {{ nimi }}
    </div>
  </body>
</html>
<script>
var EsimerkkiApp = angular.module('EsimerkkiApp', []);
EsimerkkiApp.controller('EsimerkkiController', ['$scope', 'EsimerkkiService', function($scope, EsimerkkiService) {
    $scope.nimi = EsimerkkiService.tulostaNimi();
})

EsimerkkiApp.service('EsimerkkiService', function(){
    this.tulostaNimi = function() {
        return "Matti Meikäläinen";
    }
});
</script>
```

KUVA 18. Riippuvuuden lisääminen

Yksi AngularJS:n periaatteista on modulaarisuus. Sovelluksen eri komponentit, kuten ohjaimet, direktiivit ja servicet liitetään sovellukseen moduuleina. Tämä helpottaa koodin ylläpitoa ja rakennetta sekä tekee komponenteista uudelleen käytettäviä. Komponentit eivät näe toisiaan vaan niiden välinen tiedonkulku mahdollistaan lisäämällä riippuvuus. Riippuvuus lisätään antamalla viittaus komponenttiin toisen komponentin muodostimeen. Näin komponentit voivat kutsua toistensa metodeja (Kuva 18).

4 CASE AIKA 24

Aika24.fi on internetissä toimiva ajanvarauspalvelu, joka tarjoaa kuluttajille mahdollisuuden varata ajan palvelussa olevista palveluntarjoajilta (kuva 19). Palveluita löytyy monilta eri aloilta ja kuluttaja voi hakea niitä alueen, toimialan tai nimen mukaan. Yritykset listaavat ajanvarauspalveluun omia palvelukokonaisuuksia kuluttajien varattavaksi.



KUVA 19. Ruutukaappaus Aika24-ajanvarauspalvelusta

Mobiilisovellus on tarkoitettu Aika24:n yritysasiakkaille mahdollisuudeksi lisätä uusia vapaita aikoja nopeasti ja mahdollisimman vaivattomasti. Sovellus on tarkoitus saada pilottivaiheeseen, jolloin Aika24 Oy pääsee testaamaan sovellusta testiryhmän avulla. Työni päämäärä ei ole rakentaa kaupallisesti valmista sovellusta, vaan prototyyppi joka toimii pohjana lopulliselle tuotteelle sekä ratkaisumallina uusille sovelluksille. Työn tarkoituksena ei ole olla valmis kaupallinen sovellus vaan eräänlainen pohjapiirros, millä tekniikoilla yrityksen on luontevinta rakentaa tulevia tuotteitaan.

4.1 Kehitys- ja testausympäristö

Koska sovellus halutaan rakentaa nopealla aikataululla ja yrityksellä ei ole kokemusta mobiilisovellusten tekemisestä sovellus päätettiin tehdä hybridisovelluksena. Aika24 Oy suunnittelee tekevänsä useampia sovelluksia Aika24-palvelukokonaisuuteen, joista tämä sovellus on ensimmäinen. Yritys haluaa tulevaisuudessa käyttää AngularJS-kirjastoa tulevissa tuotteissaan. Näiden vaatimusten pohjalta sovelluskehykseksi valikoitui Ionic Framework.

Kehitystyö tapahtuu Linux-pohjaisella Xubuntu-käyttöjärjestelmällä ja koodieditoriksi valittiin avoimen lähdekoodin Brackets. Ennen Ionicin asentamista täytyy asentaa Node.js, joka on ajonaikainen ympäristö. Ionic täytyy asentaa Node.js:n Node Package Manageria (npm) käyttämällä. Npm on paketinhallintajärjestelmä, jonka avulla voi asentaa ja päivittää ohjelmia.

Ionic asentuu kirjoittamalla komentopäätteelle *$ npm install -g cordova ionic*, minkä jälkeen npm asentaa Ionic SDK:n. Tämän jälkeen voi jo luoda uuden projektin, joko käyttämällä esiasennettuja malleja käyttämällä tai tyhjän. Tämä sovellus päätettiin tehdä sidemenu-mallia käyttämällä, jonka luonti tapahtuu komennolla *$ ionic start [SOVELLUKSEN NIMI] sidemenu*.

Ioniciin pitää lisätä alusta käyttöjärjestelmälle, riippuen siitä mihin käyttöjärjestelmään sovellusta ollaan rakentamassa. Ionic tukee iOS ja Android – käyttöjärjestelmiä. Tämän sovelluksen ensimmäiseksi kohdekäyttöjärjestelmäksi valittiin Android ja sen lisäys tapahtuu komennolla *$ ionic platform add android*. Android-sovellusten tekemiseen vaaditaan myös Android SDK.

4.2 Sovelluksen rakenne

Sovelluksen toiminnot koostuvat projektissa olevista html- ja JavaScript-tiedostoista, jotka sijaitsevat projektin www-kansiossa. Sovelluksen näkymät ovat templates-kansiossa olevat HTML-dokumentit, joiden sisältö ladataan index.html-tiedostoon riippuen sovelluksen tilasta. Näkymät määritellään app.js-tiedostossa ja niiden näkyvyys riippuu sovelluksen sisäisestä URL:sta. Samalla jokaiselle näkymälle annetaan ohjain, joka vastaa näkymän sisällä käytettävästä toimintalogiikasta (kuva 20).

KUVA 20. Projektin rakenne ja esimerkki app.js-tiedostosta

Sovellus koostuu useista näkymistä, jotka on rakennettu käyttämällä erilaisia Ionicin CSS-komponentteja, kuten listoja, nappeja ja palkkeja. Käyttöliittymässä päätettiin kokeilla joka näkymässä erilaisia komponentteja, jotta prototyypistä kävisi ilmi niiden

toimivuus ja ilmeikkyys mobiiliympäristössä. Näin tuleva testiryhmä pääsee kokeilemaan mahdollisimman vaihtelevia käyttöliittymäratkaisuja ja ryhmän palautteen avulla yritys voi valita mielekkäimmäksi koetun ratkaisun tuotteen jatkokehityksessä.

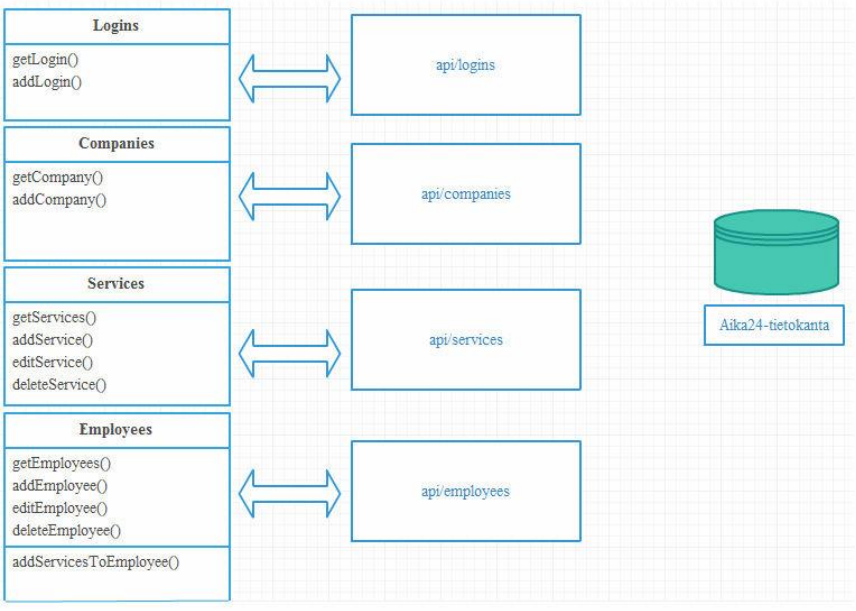
Näkymissä on toistettu samoja toimintatapoja näkymän jaotteluun. Näkymät ovat jaoteltu osioihin käyttämällä ngSwitch-direktiiviä. Direktiivin avulla HTML-dokumentti voidaan jakaa osiin, jotka näytetään käyttäjälle vain niitä tarvittaessa. Näin yhteen dokumenttiin voi koota monta näkymää saman aihealueen mukaan, ja poistaa tarpeen luoda jokaiselle näkymälle oma HTML-dokumentti. Direktiivin käyttö kuitenkin tuo ongelmia navigaation sillä painamalla mobiililaitteen paluu-painiketta sovellus palaa edelliseen HTML-dokumenttiin, eikä edelliseen osioon. Näkymien välillä voi navigoida vasemmalle aukeavan valikko-menun kautta (kuva 21).

KUVA 21. Valikko-menu

AngularJS:n tavan mukaisesti on sovelluksen JavaScript-tiedostot koottu moduuleiksi. Ohjaimet löytyvät omasta moduulistaan controllers.js-tiedostosta ja servicet puolestaan services.js-tiedstosta. Service-moduuliin on koottu kaikki REST-kutsuihin tarvittavat funktiot. Funktioita kutsutaan ohjaimista käsin ja ne yhdistetään toisiinsa lisäämällä niiden välinen riippuvuus.

Tietojen siirtämiseen sovelluksen ja tietokannan välillä käytetään Aika24-rajapintaa. Aika24-rajapinta on Node.js-pohjainen samaan aikaan yrityksen sisällä kehitetty uusi

rajapinta. Rajapinta perustuu REST-arkkitehtuuriin, joka käyttää HTTP-metodeja tiedonsiirtoon sovelluksen pyyntöjen mukaisesti. Käytetyt pyynnöt ovat GET (tietojen pyytämiseen), POST (tietojen lisäämiseen), PUT (tietojen muokkaukseen) ja DELETE (tietojen poistoon) ja niihin käytetään service-moduulista löytyvä aika24Service. Kuvassa 22 näkyy kaavio aika24Servicen ja rajapinnan välisestä kommunikaatiosta.



KUVA 22. Yksinkertaistettu kaavio sovelluksen ja rajapinnan välisestä tiedonsiirrosta

Sovelluksen REST-kutsuissa käyttäjä varmennetaan käyttämällä Basic-autentikointia. Basic-autentikoinnissa käyttäjän tunnus ja salasana enkoodataan base64-menetelmällä. Base64-menetelmässä käyttäjän antama tunnus ja salasana muunnetaan kryptatuksi merkkijonoksi. Tämän jälkeen se lisätään HTTP-otsakkeeseen authorization-kenttään (kuva 23). Näin rajapinta voi todentaa käyttäjän ja sallia REST-kutsut.

```
var userInfo = $localstorage.getObject('userInfo');
var authData = Base64.encode(userInfo.uname + ':' + userInfo.passwd);
$http.defaults.headers.common.Authorization = 'Basic ' + authData;
```

KUVA 23. Basic-autentikoinnin toteutus

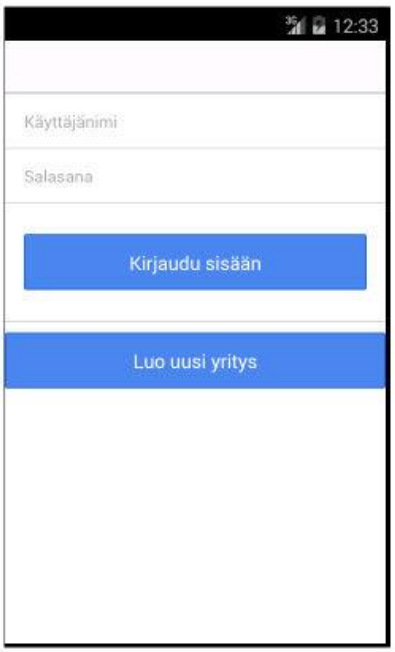
Sovellus pitää tunnuksen ja salasanan muistissa käyttämällä HTML5-ominaisuutta local storage. Services-moduuliin on rakennettu factory, jonka tarkoitus on tallentaa ja

lukea tietoja local storagesta (kuva 24). Factoryn funktioilla tunnus ja salasana laitetaan muistiin ja haetaan sieltä autentikointia varten.

```
.factory('$localstorage', ['$window', function ($window) {
    return {
        set: function (key, value) {
            $window.localStorage[key] = value;
        },
        get: function (key, defaultValue) {
            return $window.localStorage[key] || defaultValue;
        },
        setObject: function (key, value) {
            $window.localStorage[key] = JSON.stringify(value);
        },
        getObject: function (key) {
            return JSON.parse($window.localStorage[key] || '{}');
        },
        remove: function (value) {
            $window.localStorage.removeItem(value);
        }
    };
}])
```

KUVA 24. Factory local storagen hallintaan

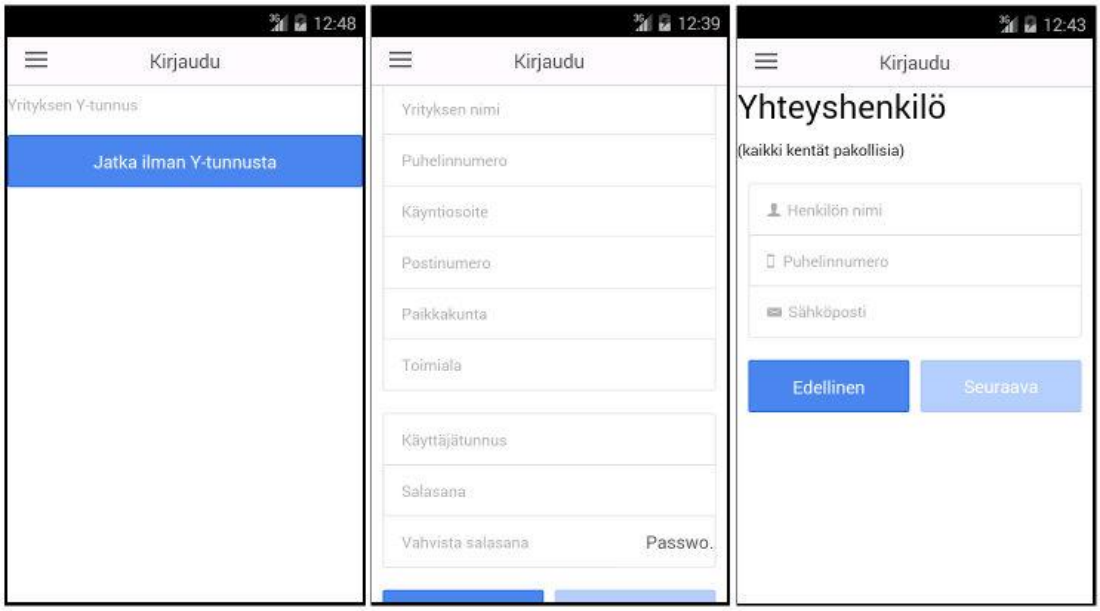
Sovelluksen käynnistyessä tutkitaan löytyykö local storagesta tunnus ja salasana. Niiden puuttuessa ohjataan käyttäjä kirjautumissivulle. Tässä näkymässä käyttäjä voi kirjautua omilla tunnuksillaan tai siirtyä luomaan uuden yhtiön (kuva 25). Kirjautuessaan sovellukseen lähetetään rajapintaan REST-kutsu, jonka seurauksena tarkistetaan löytyykö käyttäjä tietokannasta.



KUVA 25. Kirjautumis-näkymä

Uuden yhtiön luominen koostuu lomakkeesta, jossa täytetään yrityksen tiedot, yhteyshenkilön tiedot sekä käyttäjänimi ja salasana. Lomakkeen voi täyttää osaksi antamalla alussa yrityksen Y-tunnuksen. Antamalla Y-tunnuksen sovellus hakee Patentti- ja rekisterihallituksen Avoin data-palvelusta yrityksen tiedot. Tietojen haku tapahtuu YTJ-tiedot API:n kautta lähettämällä RESTillä GET-pyynnön service-moduulin kautta. API palauttaa halutun yrityksen tiedot JSON-formaatissa. Näistä palautetuista tiedoista sovellus suodattaa tarvitsevat tiedot lomakkeeseen.

Lomakkeen ensimmäisellä näkymällä on yritystietojen lisäksi käyttäjä valittava käyttäjätunnus ja salasana. Käyttäjätunnus on oletuksena sama kuin yrityksen nimi mutta on kuitenkin käyttäjän päätettävissä. Salasana on myös varmistettava kirjoittamalla se uudestaan. Salasanan tarkistus tapahtuu Angular UI.Utils-lisäkirjastolla, joka on liitetty projektiin. UI.Utils sisältää ui-validate-direktiivin, jonka avulla voi vertailla kahta eri syötettä ja varmistaa että ne ovat yhtäläiset merkkijonot.



KUVA 26. Uuden yhtiön luominen

Siirtyessään lomakkeen toiseen näkymään luodaan käyttäjälle tunnus ja yritys lisätään tietokantaan. Tämä tapahtuu ketjuttamalla kaksi funktio-kutsua toisiinsa (kuva 27). Ohjain kutsuu ensin aika24Servicen addLogin-funktiota, jonne lähetetään parametrina objekti jossa on käyttäjän käyttäjätunnus ja salasana. Funktio lähettää POST-pyynnön rajapintaan, jonka kautta tunnus lisätään tietokantaan. Onnistuneen lisäämisen jälkeen

rajapinta palauttaa sovellukselle HTTP-statuksen 200, jonka jälkeen sovellus voi kutsua servicen addCompany-funktiota. Kuvassa 26 esitetään uuden yhtiön luomiseen liittyvät näkymät.

```
$scope.createNewCompany = function () {
    var userInfo = {
        'uname': $scope.loginData.username,
        'passwd': $scope.loginData.password
    };
    $localstorage.setObject('userInfo', userInfo);
    aika24Service.addLogin().then(function (result) {
        var company = {
            'name': $scope.loginData.companyName,
            'street': $scope.loginData.companyAddress,
            'zip': $scope.loginData.companyZip,
            'phone': $scope.loginData.companyPhone,
            'locality': $scope.loginData.companyCity
        };
        aika24Service.addCompany(company).then(function (result) {
            console.log(result);
        }, function (error) {
            console.log(error);
        });
    }, function (error) {
        console.log(error);
    });
}
```

KUVA 27. Ketjutettu funktio-kutsu

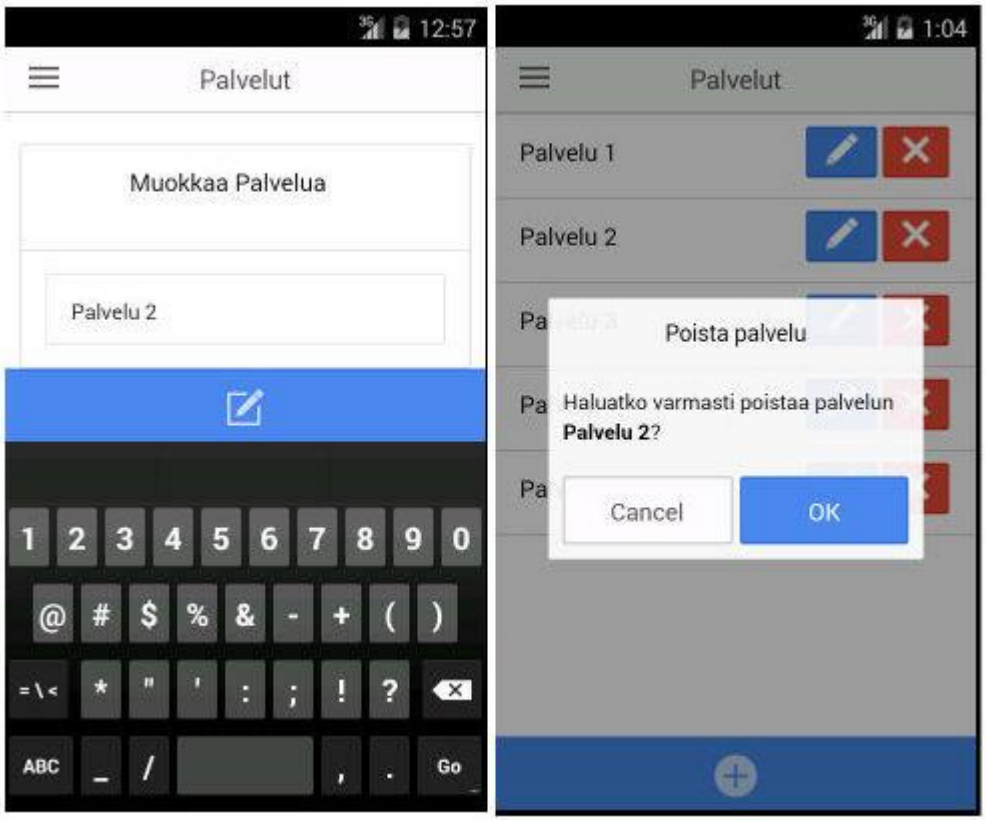
Ketjutetun kutsujen tarkoitus on tehdä samanaikaisesti sekä tunnuksen että yhtiön luominen. Tämä yksinkertaistaa käyttöliittymää ja helpottaa mahdollisten virhetilanteiden käsittelyä epäonnistuneen kutsun seurauksena. Ketjutus on lisäksi tärkeä autentikaation kannalta, sillä se pitää olla mukana yhtiön luomisessa.

Palvelut-näkymässä listataan kaikki yrityksen tarjoamat palvelut (kuva 28). Näkymässä käyttäjän on mahdollista lisätä uusia palveluita, muokata jo olemassa olevia palveluita tai poistaa palvelun. Palvelut saadaan tietokannasta rajapinnan kautta GET-pyynnön avulla. Rajapinnan palauttama tieto varastoidaan taulukkoon, jonka avulla kaikki palvelut voidaan listata AngularJS:n ng-repeat-direktiivin avulla. Lisäksi taulukon läpikäyntiä seurataan $index-ominaisuudella, joka helpottaa taulukon alkioiden ylläpitämistä.

KUVA 28. Palvelut-näkymä

Uuden palvelun lisäämiseksi käyttäjän tarvitsee antaa uudelle palvelulle nimi ja hyväksyä se. Hyväksymisen seurauksena näkymän ohjain kutsuu service-moduulin addService-funktiota. REST-arkkitehtuurin mukaisesti rajapinta palauttaa juuri lisätyn objektin. Näin objekti voidaan lisätä palvelut sisältävään taulukkoon ja palveluiden listaus päivittyy ilman erillistä GET-pyyntöä.

Palvelujen muokkaaminen tapahtuu painamalla halutun palvelun muokkauspainiketta, joka ohjaa käyttäjän muokkausnäkymään. Muokattava palvelu tulostetaan lomakkeelle, josta käyttäjä voi päivittää tietoja. Vahvistaessaan muokkauksen lähetetään rajapintaan PUT-pyynnön ja joka onnistuessaan palauttaa uuden muokatun objektin, joka korvaa taulukossa vanhan tiedon.



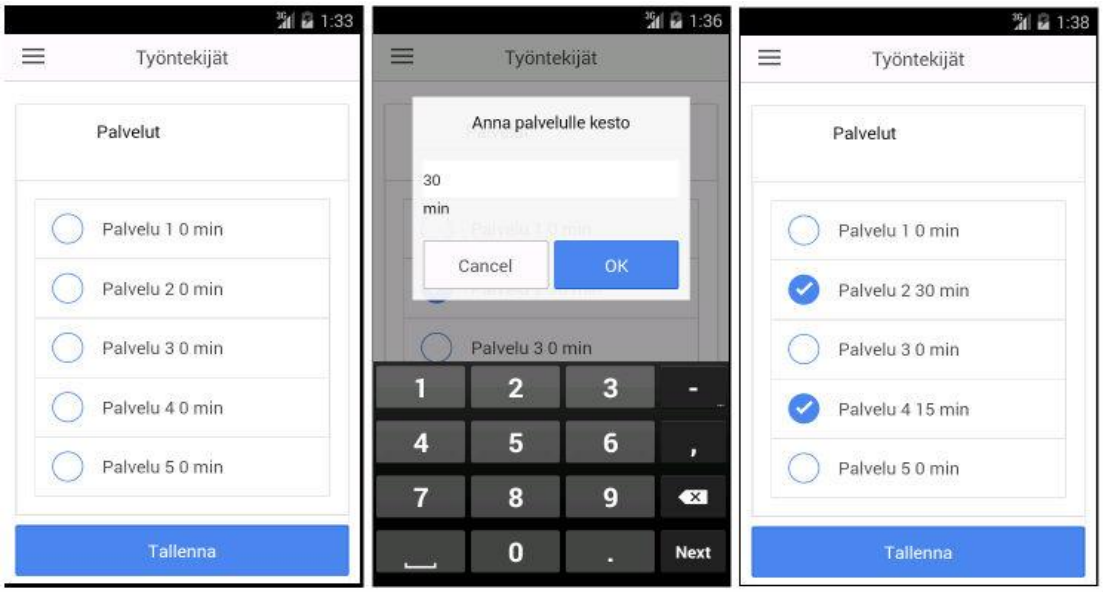
KUVA 29. Palvelun muokkaus – ja poisto-näkymä

Palvelun poistaminen tapahtuu vastaavasti poista-painikkeen kautta. Painiketta painamalla avataan popup-ikkuna, jossa käyttäjä varmistaa poiston. Näin varmistetaan että käyttäjä ei poista palvelua vahingossa. Vahvistettuaan poiston, sovellus lähettää rajapintaan REST-kutsun palvelun poistamiseen. Onnistuneen poiston seurauksena rajapinta palauttaa sovellukselle HTTP-statuksen 200, jonka jälkeen voimme poistaa palvelun näkymästä. Kuvassa 29 näkyy palvelun muokkaus- ja poisto-näkymät.

Sovelluksen työntekijät-näkymässä listataan yrityksen työntekijät (kuva 30). Työntekijät-näkymä vastaa toiminnallisuudeltaan palvelut-näkymää. Näkymässä voi lisätä, muokata tai poistaa työntekijän ja services-moduulissa on funktiot rajapinnan kutsumiseen näille metodeille. Uuden työntekijän luomiseen tarvitaan työntekijän nimi, puhelinnumero ja sähköpostiosoite, lisäksi käyttäjällä on mahdollisuus lisätä työntekijän kuva laitteen valokuva-albumista. Tämä ominaisuus on rakennettu ngCordovan ja Cordovan kamera-liitännäisen avulla.

KUVA 30. Työntekijät-näkymä

Työntekijän lisäyksen jälkeen voi työntekijälle lisätä palveluita. Nämä palvelut valitaan yrityksen palveluista ja lisätään työntekijän tarjoamaksi palveluksi. Palvelut valitaan listalta ja niille annetaan kesto minuuteissa. Valittuaan kaikki haluamansa palvelut, tallennetaan ne lähettämällä rajapintaan objektin, joka sisältää taulukkona valitut palvelut. Tämä prosessi esitetään kuvassa 31.



KUVA 31. Palveluiden lisääminen työntekijälle

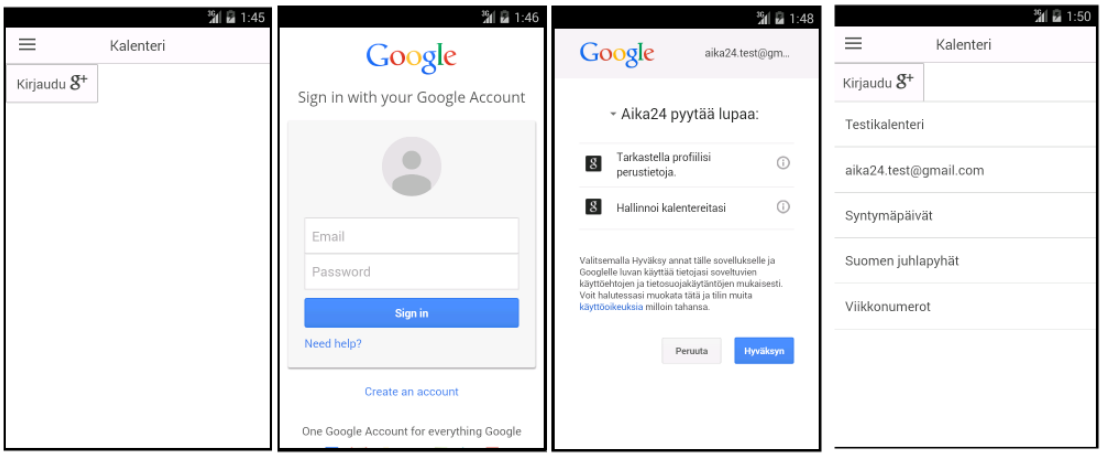
Tämä suoritetaan alustamalla rajapinnan palauttamille palvelut-objekteille ominaisuudet duration ja checked (kuva 32). Käyttäjän valitessaan palvelun ja antaessaan sille keston, päivitetään objektin ominaisuuksien arvoja. Kun käyttäjä tallentaa palvelut, käydään läpi kaikki listatut palvelut ja tutkitaan onko palvelu valittu ja mikä on sille valittu kesto.

```
$scope.services = function (index) {
    $scope.switch = 'addServiceToEmployee';
    $scope.addServicesTo = $scope.companyEmployees[index];
    aika24Service.getServices().then(function (result) {
        $scope.companyServices = result.services;
        for (i = 0; i < $scope.companyServices.length; i++) {
            $scope.companyServices[i].checked = false;
            $scope.companyServices[i].duration = 0;
        }
    }, function (error) {
        console.log(error);
    });
}
```

KUVA 32. Palvelut-objektien uusien ominaisuuksien alustaminen

Sovelluksen kalenteri päätettiin ensin toteuttaa käyttämällä laitteen omaa kalenteria. Tarkoituksena oli käyttäjän lisätessä uuden ajan, sovellus olisi päivittänyt laitteen kalenteria lisäämällä siihen uuden tapahtuman. Tämä toteutus kuitenkin osoittautui hankalaksi testausympäristössä, Android SDK:n emulaattorissa. Android-pohjaisen laitteen kalenteri pitää synkronoida ensin jonkin kalenteripalvelun kanssa, kuten esim. Google Calendar. Synkronointia emulaattorissa ei onnistuttu toteuttamaan joten vaihtoehtoisena toteutuksena päätimme että sovellus tulee käyttämään käyttäjän omaa Google-kalenteria, jonka kautta pääsemme vaikuttamaan laitteen kalenteria.

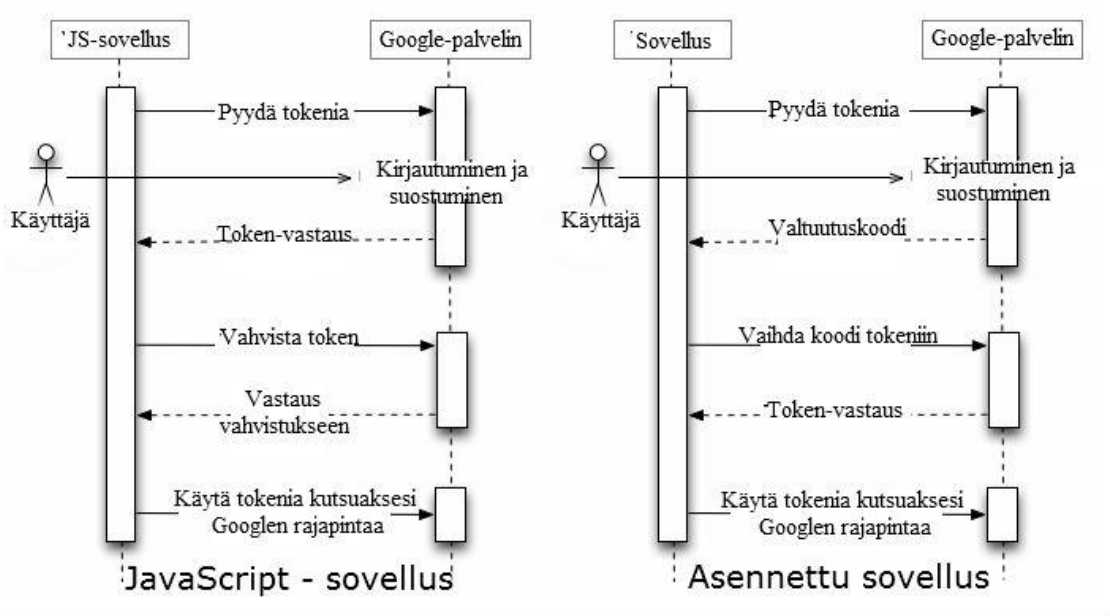
Kalenteri-näkymässä on käyttäjä tunnistautuu ensin Googlen kautta. Käyttäjä syöttää omat Googlen tunnuksen ja salasanan, jos tiedot pitävät paikkansa sovellus kysyy lupaa hallinnoida käyttäjän Google-kalenteria (kuva 33). Tunnistautuminen tapahtuu Oauth 2.0 -protokollan avulla. Protokollassa ohjataan käyttäjä Googlen kirjautumissivulle sovelluksen Google Developers Consolesta saaman Client ID:n avulla. Onnistuneen tunnistautumisen jälkeen sovellus saa Googlelta access tokenin, joka oikeuttaa sovelluksen käyttämään sille myönnettyjä Googlen rajapintoja. Tässä tapauksessa Google Calendar API.



KUVA 33. Kalenterin valitseminen

Oauth-tunnistautuminen päätettiin toteuttaa ngCordovan avulla. ngCordovan mukana tulee Oauth-lisäosa, joten ngCordova asennettiin mukaan projektiin. ngCordovan Oauth tarjoaa kehittäjälle valmiita ratkaisuja Oauth-tunnistaumiselle useisiin sosiaalisen median palveluihin, Googlen lisäksi tuettuina ovat esimerkiksi Facebook, Twitter ja Instagram.

Googlen Oauth -tunnistautumisen prosessi vaihtelee riippuen siitä minkä tyyppiseksi applikaatioksi sovellus on rekisteröity Google Developer Consolessa. Eroavaisuuksia löytyy tiedonkulusta, palveluun lähetettävistä parametreistä ja Googlen antamasta vastauksesta. Aika 24:n sovellus on rekisteröity asennettavaksi sovellukseksi (installed application), kun taas ngCordovan Oauth on suunniteltu client-puolen JavaScript-sovelluksille. Näiden sovellustyyppien Oauth-tunnistautuminen eroavat toisistaan (kuva 34).



KUVA 34. Oauth – tunnistautumisen eroavaisuudet kahden sovellustyyppien välillä (Google Developers 2015, mukaillen)

Tämä johti siihen että ngGordovan Oauth -lisäosaa piti muokata sovelluksen sopivaksi (kuva 35). Ensimmäinen eroavaisuus on kyselyyn lähetettävässä request_type-parametrissä, asennetulla sovelluksella sen tyyppi on code, kun taas JavaScript-sovelluksessa tyyppi on token. Sovellukset odottavat saavansa niille sopivan vastauksen, jotta tiedonkulku ei katkea. JavaScript-sovellus saa suoraan access tokenin, asennetun sovelluksen pitää vaihtaa saamansa code siihen.

```
browserRef.addEventListener("loadstart", function (event) {
    if ((event.url).indexOf("http://localhost") === 0) {
        var callbackResponse = (event.url).split("?")[1];
        var responseParameters = (callbackResponse).split("&");
        var parameterMap = [];
        for (var i = 0; i < responseParameters.length; i++) {
            parameterMap[responseParameters[i].split("=")[0]] = responseParameters[i].split("=")[1];
        }
        if (parameterMap.code !== undefined && parameterMap.code !== null) {
            $http.defaults.headers.post['Content-Type'] = 'application/x-www-form-urlencoded';
            $http.defaults.headers.post.accept = 'application/json';
            $http({
                method: "post",
                url: "https://www.googleapis.com/oauth2/v3/token",
                data: "code=" + parameterMap.code + "&client_id=" + clientId + "&redirect_uri=http://localhost&grant_type=authorization_code"
            })
            .success(function (data) {
                deferred.resolve(data);
            })
```

KUVA 35. Osa muokattua ngCordovan Oauth-ohjelmakoodia

Asennetulle sovellukselle myönnetään vastauksen mukana access tokenin lisäksi myös refresh token. Access tokenit ovat lyhytikäisiä tietoturvan vuoksi, joten refresh token otetaan talteen myöhempää käyttöä varten. Refresh tokenin avulla sovellus voi saada Googlelta uuden access tokenin ilman käyttäjän tunnistautumista.

```
$scope.token = {};
$scope.chosenCalendar = {};

$scope.getGoogleOauth = function () {
    testService.googleOauth().then(function (result) {
        $scope.token = result;
        $localstorage.set('googleOauthToken', result.access_token);
        testService.getGoogleCalendars().then(function (res) {
            $scope.calendars = res.data.items;
        }, function (error) {
            console.log(error);
        });
    }, function (error) {
        console.log(error);
    });
}
```

KUVA 36. Kalenteri-näkymän ohjaimen service-pyynnöt

Access token liitetään parametriksi osana GET /users/me/calendarList-pyyntöä Google Calendar-rajapintaan. Kysely palauttaa JSON-muodossa kaikki käyttäjän Google-tiliin kytketyt kalenterit. Käyttäjän valittua halutun kalenterin, lähetetään sen tunniste ja refresh token Aika24-rajapintaan. Kuvassa 36 näkyy ohjaimen kutsut service-moduuliin, tunnistautumisen onnistuttua haetaan käyttäjän kalenterit.

5 PÄÄTÄNTÖ

Opinnäytetyön tekeminen osoittautui vaativaksi ja monipuoliseksi projektiksi. Haastavuutta lisäsi yrityksen kokemattomuus mobiilikehityksessä. Tämän vuoksi työstä tuli eräänlainen tekniikkademo. Työllä haluttiin kokeilla perustoimintoja, kuten REST-kutsuja, kuvien lisäystä ja yleistä rakennetta uusilla tekniikoilla. Lisäksi sovellukselle ei ollut yleiskuvausta tarkempaa suunnitelmaa vaan sovellusta rakennettiin suoraan kokeilemalla eri ratkaisuja. Tämä johti joskus tarpeettomaan työskentelyyn ja aiheutti ongelmia aikataulun kanssa.

Työ kuitenkin saavutti sille asetetut tavoitteet ja löysi Aika24 Oy:lle ratkaisun usealle alustalle ja pienillä resursseilla kehitettävän sovelluksen tekemiseen. Ionicin myötä

täyttyi myös yrityksen toive saada kokemusta ja ymmärrystä AngularJS:n käytöstä. Sovelluksesta löytyy vaaditut ominaisuudet rajapinnan kanssa keskusteluun mutta kalenterin hallinta jäi vain synkronoinnin tasolle. Lopullinen tiedonsiirto vapaiden aikojen lisäämiseksi tulee tapahtumaan Aika24:n ja Googlen rajapintojen välillä. Sovellukselta puuttuu myös vielä tuki Windows Phoneen, sillä se ei ole Ionicin tukema alusta.

Olen myös tyytyväinen omaan suoritukseeni työssäni, sillä AngularJS ja Ionic olivat minulle uusia tekniikoita. Jouduin koko ajan tutkimaan uutta tietoa ja kehitysmenetelmiä, joita sovelsin käytännössä sovelluksen kehitystyössä. Lisäksi opinnoissani oli samaan aikaan mobiiliohjelmointi-kurssi, jonka pääpaino oli jQuery Mobilessa. Näin pääsin vertailemaan kahden erilaisen sovelluskehyksen eroavaisuutta ja näin kuinka olennaista käytettävän tekniikan valitseminen on kehitystyön kannalta.

Aika24 Oy aikoo jatkaa prototyypin kehitystyötä valmiiksi tuotteeksi. Tärkeimpänä jatkokehityksessä on kalenterin hallinnan jatkaminen. Tämä kuitenkin tulee tapahtumaan suurimmaksi osaksi uudessa rajapinnassa. Google-kalenterin valinta oli projektiin sopiva sillä se tarjoaa valmiin kalenteripohjan uusien aikojen lisäämiseen ja kalenteri on jaettavissa muiden Google-tilin omaavien kanssa. Näin jatkossa sovellusta käyttävä yritys voi käyttää yhteistä kalenteria tai työntekijät näkisivät toistensa työvuorot. Lisäksi sovellukseen tulee mahdollisesti muita keinoja kalenterin synkronointiin, sillä ngCordova tarjoaa Oauth-tunnistautumista muiden palveluiden kautta, kuten Facebook.

Tulevan kehityksen suuntaan vaikuttaa suuresti prototyypin saama testiryhmän antama palaute. Siitä selviävät ovatko Aika24:n yritysasiakkaat mahdollisesti kiinnostuneet lopullisesta tuotteesta. Lisäksi täyttyi myös tutkimusongelma ja toimeksiantajani sai ymmärrystä ja tietoa miten heidän on järkevintä lähestyä mobiilisovelluskehitystä. Uskon että näitä ratkaisuja tullaan yrityksen puolesta käyttämään heidän tulevissa kehitystyössä liittyen mobiilisovelluksiin.

LÄHTEET

All About Ionic. 2015. Ionic. WWW- dokumentti. http://ionicframework.com/docs/guide/preface.html. Päivitetty 14.4.2015. Luettu 15.4.2015.

About. 2015. jQuery Mobile. WWW-dokumentti. http://jquerymobile.com/about/. Ei päivitystietoja. Luettu 15.4.2015.

Apache Cordova. 2015. Apache Cordova. WWW-dokumentti. https://cordova.apache.org/. Ei päivitystietoja. Luettu 15.4.2015.

Apple – iOS – Accessibility. 2015. Apple. WWW-dokumentti. https://www.apple.com/accessibility/ios/. Ei päivitystietoja. Luettu 15.4.2015.

Budiu, Raluca 2013. Mobile: Native Apps, Web Apps, and Hybrid Apps. WWW-dokumentti. http://www.nngroup.com/articles/mobile-native-apps/. Päivitetty 14.9.2013. Luettu 8.4.2015.

Build an App with Navigation and Routing - Part 1. 2014. Learn Ionic. WWW-dokumentti. http://learn.ionicframework.com/formulas/navigation-and-routing-part-1/. Päivitetty 30.9.2014. Luettu 7.4.2015.

Coenraets, Christophe 2014. Sample Mobile Application with Ionic and AngularJS. Blogi. http://coenraets.org/blog/2014/02/sample-mobile-application-with-ionic-and-angularjs/. Päivitetty 5.2.2014. Luettu 29.3.2015.

Data Binding. 2015. AngularJS. WWW-dokumentti. https://docs.angularjs.org/guide/databinding. Päivitetty 10.4.2015. Luettu 14.4.2015

Download Android Studio and SDK Tools 2015. Android Developers. WWW-dokumentti. https://developer.android.com/sdk/index.html. Ei päivitystietoja. Luettu 10.4.2015.

Duckett, Jon 2014. JavaScript and JQuery: Interactive Front-End Web Development. Indianapolis: John Wiley & Sons, Inc.

FAQs. 2015. PhoneGap. WWW-dokumentti. http://phonegap.com/about/faq/. Päivitetty 1.1.2015. Luettu 12.4.2015.

HTML5 compatibility on mobile and tablet browsers with testing on real devices. 2015. Mobile HTML 5. WWW-dokumentti. http://mobilehtml5.org/. Ei päivitystietoja. Luettu 16.4.2015.

iOS Developer Program 2015. Apple Developer. WWW-dokumentti. https://developer.apple.com/programs/ios/. Ei päivitystietoja. Luettu 16.3.2015.

jQuery. 2015. jQuery. WWW-dokumentti. http://jquery.com/. Ei päivitystietoja. Luettu 1.4.2015.

jQuery Mobile. 2015. jQuery Mobile. WWW-dokumentti. https://jquerymobile.com/. Ei päivitystietoja. Luettu 1.4.2015.

Järvinen, Jani 2012. Windows Phone sovelluskehitys. Jyväskylä: Docendo.

Kauppi, Juho 2014. jQuery Mobile -pohjainen mobiiliportaali. Metropolia Ammattikorkeakoulu. Tietotekniikan koulutusohjelma. Insinöörityö. PDF-dokumentti. https://www.theseus.fi/bitstream/handle/10024/71658/Juho_Kauppi_jQuery_Mobile_pohjainen_mobiiliportaali.pdf?sequence=1. Päivitetty 26.3.2014. Luettu 16.4.2015.

Lehdonvirta, Pyry & Korpela, Jukka K. 2013. HTML5 sovellusalustana. Helsinki: RPS-yhtiöt.

Manage Plugins for Apps Built with Visual Studio Tools for Apache Cordova. 2015. MSDN. WWW-dokumentti. https://msdn.microsoft.com/en-us/library/dn757051.aspx. Ei päivitystietoja. Luettu 11.4.2015.

MVC Architecture. 2015. Chrome Developer. WWW-dokumentti. https://developer.chrome.com/apps/app_frameworks. Ei päivitystietoja. Luettu 12.4.2015.

Manipulation | jQuery API Documentation. 2015. jQuery. WWW-dokumentti. https://api.jquery.com/category/manipulation/. Ei päivitystietoja. Luettu 1.4.2015.

Natiivisovellus. 2015. Gambit. WWW-dokumentti. http://www.gambitgroup.fi/meista/natiivisovellus/. Ei päivitystietoja. Luettu 3.4.2015.

Natili, Giorgio 2013. PhoneGap 3 Beginner's Guide. Birmingham: Packt Publishing Ltd.

Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options. 2015. Salesforce Developers. WWW-dokumentti. https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options. Päivitetty 14.4.2015. Luettu 16.4.2015.

ngCordova. 2015. ngCordova. WWW-dokumentti. http://ngcordova.com/docs/. Päivitetty 15.4.2014. Luettu 17.4.2015.

Nokia Lumia suosituin yritysten puhelinmalli. 2015. Marketvisio. WWW-dokumentti. http://www.marketvisio.fi/fi/ajankohtaista/uutiset-marketvisio/1838-nokia-lumia-suosituin-yritysten-puhelinmalli. Ei päivitystietoja. Luettu 18.3.2015.

Patel, Viral. 2014. Introduction to AngularJS Services. Blogi. http://viralpatel.net/blogs/angularjs-service-factory-tutorial/. Päivitetty 7.1.2014. Luettu 12.4.2015.

Raasch, Jon 2013. Pushing the Limits : JavaScript Programming : Pushing the Limits. Chichester: John Wiley & Sons, Ltd.

Rudolph, Patrick 2014. Hybrid Mobile Apps: Providing A Native Experience With Web Technologies. WWW-dokumentti. http://www.smashingmagazine.com/2014/10/21/providing-a-native-experience-with-web-technologies/. Päivitetty 21.10.2014. Luettu 18.3.2015.

Sass Basics. 2015. Sass. WWW-dokumentti. http://sass-lang.com/guide. Päivitetty 2.4.2015. Luettu 7.4.2015.

Scopes. 2014. ng-newsletter. WWW-dokumentti. http://www.ng-newsletter.com/posts/beginner2expert-scopes.html. Päivitetty 5.12.2014. Luettu 16.2.2015.

Smartphone OS Market Share, Q4 2014. IDC. WWW-dokumentti. http://www.idc.com/prodserv/smartphone-os-market-share.jsp. Ei päivitystietoja. Luettu 18.3.2015.

Stefanov, Stoyan & Sharma, Kumar Chetan 2013. Object Oriented JavaScript-Second Edition. Birmingham: Packt Publishing Ltd.

Summerfield, Jason 2015. Mobile Website vs. Mobile App (Application): Which is Best for Your Organization?. WWW-dokumentti. http://www.hswsolutions.com/services/mobile-web-development/mobile-website-vs-apps/. Ei päivitystietoja. Luettu 15.4.2015.

The config.xml File. 2014. Apache Cordova. WWW-dokumentti. https://cordova.apache.org/docs/en/4.0.0/config_ref_index.md.html. Päivitetty 28.10.2014. Luettu 9.4.2015.

The Last Word on Cordova and PhoneGap. 2014. Ionic Blog. Blogi. http://blog.ionic.io/what-is-cordova-phonegap/. Päivitetty 6.3.2014. Luettu 12.4.2015.

Top 8 Mobile Operating Systems in Finland from Mar 2014 to Apr 2015. 2015 StatCounter Global Stats. WWW-dokumentti. http://gs.statcounter.com/#mobile_os-ww-monthly-201403-201504. Ei päivitystietoja. Luettu 18.3.2015.

Top 8 Mobile Operating Systems from Mar 2014 to Apr 2015. 2015 StatCounter Global Stats. WWW-dokumentti. http://gs.statcounter.com/#mobile_os-FI-monthly-201403-201504-bar. Ei päivitystietoja. Luettu 18.3.2015.

Understanding Controllers. 2015. AngularJS. WWW-dokumentti. https://docs.angularjs.org/guide/controller. Päivitetty 23.1.2015. Luettu 17.1.2015.

Using OAuth 2.0 to Access Google APIs. 2015. Google Developers. WWW-dokumentti. https://developers.google.com/identity/protocols/OAuth2. Päivitetty 8.4.2015. Luettu 17.1.2015.

What's a Windows Runtime app?. 2015. MSDN. WWW-dokumentti. https://msdn.microsoft.com/library/windows/apps/dn726767.aspx. Ei päivitystietoja. Luettu 12.4.2015.