Haaga-Helia
ammattikorkeakoulu Oy

DinnerChatter-sovelluksen kehittäminen Androidille hybridimenetelmin

Olli Nyholm

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2017

Tiivistelmä

Haaga-Helia ammattikorkeakoulu Oy

Tekijä(t)
Olli Nyholm

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön otsikko
DinnerChatter-sovelluksen kehittäminen Androidille hybridimenetelmin

Sivu- ja liitesivumäärä
30 + 1

Projektin tavoitteena oli luoda ruokailusuunnitteluun tarkoitettu Android-sovellus käyttäen Cordova-hybridisovelluskehystä. Sovellus auttaa kaveriporukoita pääsemään yhteisymmärrykseen syömäpaikasta tarjoten mahdollisuuden lisätä ravintoloita listaa, näyttää ne kartalla ja keskustella niistä. Sovellus toteutettiin parityönä, ja tämä opinnäytetyö keskittyy puhelimeen asennettavaan ohjelmistoon.

Teoriaosuudessa käsitellään sovelluksen kehittämisessä käytettyjä teknologioita kuten Cordova, AngularJS ja OnsenUi, sekä miten ne soveltuvat mobiilikehittämiseen. Luvussa käydään läpi AngularJS:n perusteet, jotka ovat olennainen osa kehitystä.

Toiminnallisessa osuudessa perustellaan valitut teknologiat ja kerrotaan, miten niitä hyödynnettiin sovelluksen kehittämisessä. Luvussa käydään läpi sovelluksen käyttöliittymä näkymä kerrallaan ja nostetaan esille mielenkiintoisimpia ominaisuuksia, sekä miten ne toteutettiin. Lopuksi käsitellään palvelinpuolen integraatiota ja sovelluksen julkaisua Google Play-palvelussa.

Viimeinen luku käy läpi projektin aikana nousseita mietteitä sovelluksen kehittämisestä ja käytetyistä teknologiosta sekä pohditaan myös sovelluksen tulevaisuutta ja jatkokehitys-suunnitelmia. Sovellus saatiin valmiiksi juuri ennen opinnäytetyön päättämistä, minkä takia testaus laajemmassa mittakaavassa jäi toteuttamatta. Jatkokehitykseen jäi sovelluksen hienosäätämistä ja runsaasti kehityksen aikana nousseita ideoita uusista toiminnallisuuksista, joita ei vielä ehditty toteuttaa.

Asiasanat
Android, Cordova, hybridikehitys, mobiilisovellus

Sisällys

1 Johdanto

Projektin tarkoituksena oli toteuttaa ruokailusuunnitteluun tarkoitettu DinnerChatter-sovellus ja julkaista se Android-käyttöjärjestelmälle Google Play-palvelussa. Järjestelmä tehtiin parityönä. Tässä työssä keskitytään puhelimelle asennettavaan ohjelmistoon ja sen toteuttamiseen hybridimenetelmillä, kun puolestaan toinen työ kuvaa palvelinpuolen ohjelmiston kehittämistä. Järjestelmä toteutettiin, koska kumpikin kehittäjä on kiinnostunut sovelluskehityksestä ja valituista teknologioista.

Sovelluksen tarkoitus on auttaa käyttäjäryhmiä pääsemään yhteisymmärrykseen syömäpaikasta tarjoamalla ominaisuuksia muun muassa ravintoloiden listaamiseen ja niiden näyttämiseen kartalla. Käyttäjät voivat keskustella ravintoloista sovelluksen sisäänrakennetulla chat-toiminnallisuudella.

Sovellus kehitettiin käyttäen Cordova-hybridisovelluskehystä, jonka avulla mobiilisovellukset voidaan toteuttaa standardeilla Web-teknologioilla. Hybridisovelluksissa JavaScript on suuressa roolissa, ja sitä käytettiin tässä projektissa AngularJS-sovelluskehyksen avulla. Jotta sovellus pääsee käsiksi puhelimen omiin sisäänrakennettuihin toimintoihin, sovelluskehys tarjoaa kirjaston lisäosia, joita kehittäjä pääsee hyödyntämään Cordovan omien API:en kautta. Vaikka Cordova onkin hyvin helppokäyttöinen ja selkeä, se ei ole ilman haittapuolia suhteessa natiiveihin mobiilisovelluksiin.

Opinnäytetyön tavoite on tutkia käytettyjä teknologiota ja pohtia, miten ne sopivat käytettäväksi mobiilikehitysprojekteihin. Työssä käydään läpi sovelluksen kehityksen kulku, tutustutaan valittuihin aihealueisiin tarkemmin ja osoitetaan, kuinka niistä on päästy hyötymään projektin aikana. Työssä nostetaan myös kehityksessä esille nousseet teknologioihin, työvälineisiin ja työskentelytapoihin liittyvät ongelmat.

1.1 Käsitteet

JavaScript = Skriptikieli, joka on olennainen osa nykyaikaista Web-kehitystä. Käytetään dynaamisuuden lisäämiseksi käyttäjäpuolen ohjelmistoihin.

HTML5 eli hypertext markup language = Verkkosivujen elementtien luomiseen käytetty standardi kieli.

CSS eli cascading style sheets = verkkosivujen elementtien tyylien muuttamiseen käytetty standardi ohjeistus.

Android = Googlen kehittämä Linux-pohjainen avoimen lähdekoodin mobiilikäyttöjärjestelmä.

Palvelinpuoli = Palvelimella oleva ohjelmisto, jonka kanssa puhelinsovellus kommunikoi tiedonsiirron tarkoituksissa.

Hybridisovellus = Web-teknologioilla toteutettu sovellus, joka toimii useilla alustoilla ilman ylimääräistä työtä. Mobiililaitteilla hybridisovellukset käyttävät hyväkseen WebView-ominaisuutta, joka mahdollistaa verkkosisällön näyttämisen käyttäjälle.

Natiivisovellus = Kehitetty varta vasten yhdelle mobiilialustalle alustan omia teknologioita ja palveluita käyttäen.

2 Tietoperusta

Tässä luvussa esitellään sovelluksessa käytettävät teknologiat ja niiden olennaisimmat ominaisuudet kehitettävän sovelluksen suhteen. Kaikki kehityksessä käytetyt teknologiat ovat ilmaisia ja vapaasti saatavilla sekä pitkälti riippumattomia kehitysympäristöstä. Kappaleen tarkoitus ei ole kuvata jokaista teknologiaa todella tarkasti, vaan antaa lukijalle yleinen kuva siitä, mistä kehityksessä on kyseessä.

2.1 Apache Cordova hybridiohjelmistokehys

Apache Cordova on sovelluskehys, jota käytetään hybridisovelluksien kehittämiseen mobiilialustoille. Cordova tunnettiin aiemmin nimellä PhoneGap, kunnes Adobe Systems osti sitä kehittäneen yrityksen ja julkaisi avoimeen lähdekoodiin perustuvan version Apache Cordova nimellä. PhoneGap on vielä käytössä ja Adobe jatkaa sen kehittämistä. Sen suurin ero Cordovaan on Adoben palvelut, jotka auttavat sovelluksien kehittämisessä ja julkaisussa (Wikipedia 2017.)

Cordovalla kehitys tapahtuu käyttämällä yleisiä Web-teknologioita kuten HTML5, CSS ja JavaScript. Toisin kuin natiiveissa sovelluksissa, käytetyt teknologiat ovat aina samat kehitettävästä alustasta riippumatta. Sovelluksen käyttöliittymänä toimii puhelimen WebView-ominaisuus, jota hyödyntäen Cordova skaalaa näkymän koko ruudun kokoiseksi ja näyttää käyttäjälle luodut HTML-näkymät. Näin ollen Cordova ei millään tavalla muokkaa luotua koodia (Code.tutplus 2017.) Cordovaan kuuluu lista liitännäisiä, jotka mahdollistavat kommunikoinnin puhelimen ominaisuuksien kanssa, kuten kameran, GPS:n ja tiedostojärjestelmän. Cordovaa käytetään komentoliittymän kautta. Tätä kautta kehittäjä pystyy hallinnoimaan kehitystä projektien luomisesta liitännäisien ja alustojen lisäämiseen.

Monet muut hybridikehykset käyttäjät pohjana Cordovaa tai PhoneGapia. Kehykset kuten Monaca ja Ionic ottavat Cordovan perustoiminnallisuudet ja laajentavat niitä. Esimerkiksi Ionic on rakennettu AngularJS-kehyksen päälle (Ionicframework 2017.) Android-sovelluksissa käytettävistä sovelluskehyksistä Cordova on tällä hetkellä käytössä 6,44% sovelluksista (AppBrain 2017.)

Kuvassa 1 esitetään Cordovan tukemat alustat ja niillä käytettävissä olevat ominaisuudet, Cordova tukee kokonaisuudessaan seitsemää kehitysalustaa. Kunkin alustan tuetut ominaisuuden kuitenkin vaihtelevat, varsinkin liitännäisten käytettävyyksissä ilmenee alustakohtaisia eroja (Cordova 2017a.) Tällä hetkellä Android ja iOS tukevat ainoina jokaista

Cordovan tarjoamaa ominaisuutta. Alustoille on saatavilla kolmannen puolen liitännäisiä, jotka laajentavat kehitysmahdollisuuksia entisestään.

	android	blackberry10	ios	Ubuntu	wp8 (Windows Phone 8)	windows (8.1, 10, Phone 8.1)	OS X
cordova CLI	✓ Mac, Windows, Linux	✓ Mac, Windows, Linux	✓ Mac	✓ Ubuntu	✓ Windows	✓	✓ Mac
Embedded WebView	✓ (see details)	✗	✓ (see details)	✓	✗	✗	✓
Plugin Interface	✓ (see details)	✓ (see details)	✓ (see details)	✓	✓ (see details)	✓	✓
	Core Plugin APIs						
Accelerometer	✓	✓	✓	✓	✓	✓	✗
BatteryStatus	✓	✓	✓	✓	✓	✓ * Windows Phone 8.1 only	✗
Camera	✓	✓	✓	✓	✓	✓	✗
Capture	✓	✓	✓	✓	✓	✓	✗
Compass	✓	✓	✓ (3GS+)	✓	✓	✓	✗
Connection	✓	✓	✓	✓	✓	✓	✗
Contacts	✓	✓	✓	desktop only	✓	partially	✗
Device	✓	✓	✓	✓	✓	✓	✓
Events	✓	✓	✓	✓	✓	✓	✗
File	✓	✓	✓	✓	✓	✓	✓
File Transfer	✓	✓ * Do not support onprogress nor abort	✓	✗	✓ * Do not support onprogress nor abort	✓ * Do not support onprogress nor abort	✗
Geolocation	✓	✓	✓	✓	✓	✓	✗
Globalization	✓	✓	✓	✓	✓	✓	✗
InAppBrowser	✓	✓	✓	✓	✓	uses iframe	✗
Media	✓	✓	✓	✓	✓	✓	
Notification	✓	✓	✓	✓	✓	✓	✗
Splashscreen	✓	✓	✓	✓	✓	✓	✗
Status Bar	✓	✗	✓	✗	✓	✓ Windows Phone 8.1 only	✗
Storage	✓	✓	✓	✓	✓ localStorage & indexedDB	✓ localStorage & indexedDB	✗
Vibration	✓	✓	✓	✓	✓	✓ * Windows Phone 8.1 only	✗

Kuva 1. Cordovan tukemat ominaisuudet laitekohtaisesti (Cordova 2017b.)

2.1.1 Cordovan hyödyt ja haitat

Suurin Cordovan tarjoama etu natiiveihin sovelluksiin verrattuna on kehittämisen nopeus, varsinkin, jos tarkoituksena on kehittää usealle alustalla samanaikaisesti, koska käytettävät teknologiat ovat aina samat. Kehittämisen perustuminen yleisiin Web-teknologioihin mahdollistaa nopean oppimisen kaikille, joille Web-kehityksen perusteet ovat tutut. Cordovan suuren yhteisön ansiosta avun löytäminen tarvittaessa on helppoa. Esimerkiksi kehittäjien ohjelmointiin liittyvien ongelmien ratkaisuun erikoistuneella, Stack overflow-sivustolta löytyy yli 50,000 hakusanaan ”Cordova” liittyvää kysymystä (Stack overflow 2017.) Cordova on täysin ilmainen ja perustuu kokonaan avoimeen lähdekoodiin, joten kuka tahansa voi kehittää liitännäisiä ja julkaista niitä (Code.tutplus 2016.)

Hybridisovelluksien suorituskyky on natiivisovelluksia heikompi, koska hybridisovellukset käyttävät WebView-ominaisuutta käyttöliittymän esittämiseksi käyttäjälle. Pienemmissä sovelluksissa suorituskykyerot eivät ole merkittäviä, mutta suuret tai muuten enemmän prosessoria ja näytönohjanta rasittavat kokonaisuudet kärsivät huomattavasti. Tällaisissa tilanteissa natiivikehitys on suositeltavaa. Toisin kuin natiivissa kehittämisessä, Cordovan mukana ei tule valmiita käyttöliittymäkomponentteja. Natiivin ulkoasun saavuttamiseksi kehittäjällä on useita vaihtoehtoja erilaisten käyttöliittymäkehysten käyttöön, jotka imitoivat kunkin käyttöjärjestelmän komponentteja (Code.tutplus 2016.) Usealla alustalle kehittäessä on huomioitava, että hybridisovellus käyttää laitteen selainta. Eroja sovelluksen ulkoasussa ja yleisessä käyttäytymisessä saattaa ilmetä, kaikki selaimet eivät tue samoja CSS, HTML ja JavaScript-komponentteja (Developer.salesforce 2016.) Käyttäjien luomien Cordova liitännäisten käytössä saattaa ilmetä ongelmia, jos liitännäistä ei ole päivitetty uusimman Cordova-version mukaan. Esimerkiksi tarvittavien lupien pyytämisessä käyttäjältä voi olla ongelmia, koska liitännäisen käyttämät metodit voivat olla vanhentuneet.

2.1.2 Cordovan liitännäiset

Olennainen osa Cordova-kehitystä ovat pluginit eli liitännäiset, jotka mahdollistavat sovelluksen kommunikoinnin puhelimen sisäänrakennettujen toimintojen kanssa, kuten kamera, tiedostot ja GPS. Liitännäiset on kirjoitettu kunkin kehitysalustan natiivilla kielellä, ja ne käyttävät JavaScript API:a yhteyden muodostamiseksi liitännäisten ja käyttöliittymän välille, joten niiden käyttö on luontaista ja sopii jo käytettyihin teknologioihin. Cordova tarjoaa kattavan listan liitännäisiä kehittäjien käyttöön. Niiden toiminnassa on jonkin verran

alustakohtaisia eroja, mutta ne pääsiassa ovat käytettävissä yleisimmillä alustoilla. Käyttäjät voivat luoda ja julkaista itse tekemiään liitännäisiä, jotka saattavat olla hyödyksi, jos ydinliitännäisistä ei kaivattua ominaisuutta löydy. Liitännäisten asennus tapahtuu yleisesti npm-paketinhallintajärjestelmän kautta, jota kautta kolmannen osapuolen liitännäisiä on myös helppo löytää (Cordova 2017a.)

2.1.3 Cordovan vaatimukset

Cordova on hyvin helppo ja nopea asentaa, eikä sillä ole erityisiä vaatimuksia käytetyn laitteiston suhteen. Ainoa välttämätön ohjelmisto, jonka Cordova tarvitsee, on Node.js-kehitysympäristö, koska asennus tapahtuu käyttäen NPM-paketinhallintapalvelua. Myös Git-versionhallintajärjestelmän asentaminen on suositeltavaa, sillä Cordova käyttää sitä joidenkin tiedostojen lataamiseen (Cordova 2017c.)

Laitteistovaatimukset on huomioitava, jos tarkoituksena on rakentaa ja emuloida sovellusta kehitysympäristössä. Kehitettävästä alustasta riippuen on asennettava tarvittavat työkalut, jotta sovellusta voidaan testata. Käyttöjärjestelmien emulointi on kohtalaisen raskasta, joten heikoimmat tietokoneet eivät välttämättä suoriudu siitä (Developer.android 2017a.) Sovellukset voidaan testata myös suoraan puhelimella tarkemman testituloksen saavuttamiseksi.

2.1.4 Onsen UI käyttöliittymäkirjasto

Onsen UI on avoimeen lähdekoodiin perustuva käyttöliittymäkehys HTML5 hybridisovelluksien kehittämiseen Cordova ja PhoneGap sovelluskehyksillä. Onsen UI sisältää CSS ja JavaScript-kirjastot, jotka mahdollistavat natiivilta näyttävien HTML-komponenttien ja animaatioiden tuottamisen helposti. Kehys tukee sekä Android- että iOS-alustaa, joten se sopii hyvin samanaikaiseen kehittämiseen usealle alustalle. Elementtien tyyli muuttuu automaattisesti kehitettävän alustan mukaan, joka vähentää vaadittavan työn määrää edelleen. Yksi haittapuoli, johon Onsen UI:lla kehittämisessä voi törmätä on yhteisön koko, joten avun löytäminen saattaa osoittautua vaikeaksi. Onsen UI tukee useita JavaScript-kirjastoja ja kehyksiä, ja integroi niiden valmiita toiminnallisuuksia omiensa kanssa. Kehys on näin ollen helppo sisällyttää projekteihin ilman merkittäviä muutoksia (OnsenUI 2017.)

2.2 AngularJS JavaScript-ohjelmistokehys

AngularJS on Googlen kehittämä ja vuonna 2010 julkaisema, avoimeen lähdekoodiin perustuva JavaScript-ohjelmistokehys, joka laajentaa HTML-elementtejä ja mahdollistaa dy-

naamisten sovellusten luomisen helposti. AngularJS loistaa varsinkin yhdensivun sovelluksien kehittämisessä, jonka ansoista kehys on hyvin potentiaalinen hybridisovelluksien kehittämiseksi. Ohjelmistokehyksen avulla pystyy luomaan sovelluksia, palvelinpuolen ohjelmistoissa tutuksi tulleella, MVC- eli mode-view-controller-periaatteella, joka jakaa sovelluksen arkkitehtuurin kolmeen osaan, ja poistaa osien välisiä riippuvuuksia (Tutorialspoint 2017.)

AngularJS:n yksi varteenotettavimmista ominaisuuksista on two-way data binding, joka tarkoittaa mallin (model) ja näkymän (view) synkronoitua yhteyttä. Tämä merkitsee sitä, että kaikki joko mallissa tai näkymässä tehdyt muutokset päivitetään välittömästi ja automaattisesti myös toisessa osapuolessa (AngularJS 2017a.) Tällaisen tiedon sitomisen suurin hyöty on sovelluksen arkkitehtuurin yksinkertaistaminen tiedon keräämisen ja käsittelemisen kannalta.

2.2.1 AngularJS direktiivit

AngularJS:n direktiivit laajentavat valmiita HTML-elementtejä attribuuteilla, ja lisäävät niihin kustomoituja toimintoja. Sivua ladattaessa AngularJS:n HTML-kääntäjä yhdistää lisätyt attribuutit niiden viittaamiin toimintoihin. Ohjelmistokehyksen JavaScript-kirjaston mukana tulee joukko direktiivejä, jotka ovat keskeisiä kehityksen kannalta (AngularJS 2017b.)

AngularJS:n sisäänrakennetut direktiivit tunnistaa ng-liitteestä. Kuvassa 2 luodaan yksinkertainen AngularJS sovellus, joka tulostaa käyttäjän syötteen reaaliajassa input-kentän alle. Ng-app direktiivi rakentaa ja määrittää sovelluksen juurielementin, ng-model sitoo käyttäjän input-kenttään kirjoittaman syötteen ”nimi”-nimiseen muuttujaan, ja ng-bind sitoo muuttujan arvon <p>-elementtiin.

```
<div ng-app>
  <input type="text" ng-model="nimi">
  <p ng-bind="nimi"></p>
</div>
```

Kuva 2. AngularJS:n direktiivien käyttäminen.

AngularJS:n sisäänrakennetut direktiivit eivät aina riitä sovelluksen tarpeisiin, joten kehittäjä voi myös luoda omia direktiivejä. Kuvassa 3 luodaan direktiivi, joka lisää template-valinnan määrittämän sisällön sivulle, kun nimi-directive-attribuutti lisätään HTML-elementtiin. Tämän tapaiset direktiivit sopivat hyvin käyttötarkoituksiin, joissa sama sisältö toistuu usein koodin sisällä, koska kun sisällön tarpeet muuttuvat, tarvittavat muutokset on tehtävä vain yhdessä paikassa (AngularJS 2017b.)

```
<div ng-app="my-app" nimi-directive>
</div>

<script>
angular.module('my-app', [])
.directive('nimiDirective', function(){
  return {
    template: '<input type="text" ng-model="nimi">' +
              '<p ng-bind="nimi"></p>'
  };
});
</script>
```

Kuva 3. Angular direktiivien luominen.

2.2.2 AngularJS Controller ja $scope

AngularJS:n Controller on JavaScript objekti, jota käytetään sovelluksen datanhallintaan. Controller sidotaan HTML-elementteihin ng-controller-direktiivillä, ja se on käytettävissä vain sidotun elementin sisällä. Controllereihin tulisi soveltaa samaa logiikkaa kuin MVC-arkkitehtuurissa; yhden Controllerin tulisi hallinnoida vain yhtä näkymää, pitäen ne mahdollisimman yksinkertaisina (AngularJS 2017c.)

Controllerin olennainen osa on $scope-objekti, joka toimii sovelluksen mallina (model). $scope on välikäsi näkymän (view) ja Controllerin välillä. Kaikki Controllerin määrittämän elementin sisällä tapahtuvat tiedonsitomiset liitetään automaattisesti $scope-objektiin, ja vastaavasti kaikki Controllerissa $scope-objektiin sidottu data on vapaasti käytettävissä näkymässä direktiivien avulla. $scope kuuntelee kaikkia siihen sidottuja muuttujia, kun muutos havaitaan, kaikille direktiiveille ilmoitetaan arvon muuttuneen, jolloin se on välittömästi käytettävissä näkymässä. Jokainen sovellus sisältää aina $rootScope-objektin, josta kaikki $scope-objektit johdetaan. $rootScope rakentaa sovellukset sen sisältämien direktiivien mukaan, ja toisin kuin $scope, se on käytettävissä koko sovelluksessa, ja näin ollen sopien hyvin tiedonsiirtoon Controllerien välille (AngularJS 2017d.)

Kuvan 4. sovelluksessa käyttäjän painaessa lisää-nappia ng-click-direktiivi kutsuu Controller-objektin lisaaFunc-funktiota, joka lisää input-kentän syötteen nimet-listaan. Funktiot

voidaan sitoa $scope-objektiin samalla tavalla kuin muuttujatkin, jolloin ne ovat kutsuttavissa näkymästä. Esimerkissä käytetty {{nimet}}-syntaksi vastaa ng-bind-direktiiviä, ja sitoo listan elementtiin.

```
<div ng-app="my-app" ng-controller="myController">
  <input type="text" ng-model="nimi">
  <button ng-click="lisaaFunc()">lisää</button>
  <p>{{nimet}}</p>
</div>

<script>
  angular.module('my-app', [])
  .controller('myController', function($scope){
    $scope.nimi = "";
    $scope.nimet = [];
    $scope.lisaaFunc = function(){
      $scope.nimet.push($scope.nimi);
    }
  });
</script>
```

Kuva 4. AngularJS Controller ja $scope käytössä.

2.2.3 AngularJS $http-palvelu

AngularJS:n $http-palvelu mahdollistaa joko XMLHttpRequest- tai JSONP-pyyntöjen (request) tekemisen ulkoiselle palvelimelle tiedonsiirtotarkoituksessa, $http-pyynnöt ovat rakenteeltaan asynkronisia, eli ne ovat täysin riippumattomia muusta koodista. Pyyntö jää odottamaan palvelimen vastausta (response), minkä jälkeen sille voidaan tehdä määritetyt toimenpiteet (AngularJS 2017e.)

Kuvassa 5 on yksinkertainen $http-pyyntö, .post()-metodi kertoo, että olemme lähettämässä tietoa palvelimelle, ja sen sisällä määritetään pyynnön vastaanotto-osoite ja lähetettävä tieto. Pyynnön jälkeen .then()-metodi alustaa jatkotoimenpiteet, metodin ensimmäinen funktio ajetaan aina pyynnön onnistuessa ja jälkimmäinen epäonnistuessa. Kummankin funktion parametreissa voidaan ottaa vastaan palvelimelta tuleva vastaus. AngularJS:n määrityksen mukaan kaikki vastaukset, joiden status-koodit ovat 200 ja 299 välillä ovat onnistuneita (AngularJS 2017e.)

```
var data = {nimi: olli}

$http.post('http://localhost:8080/', data).then(function(success){
  console.log("Onnistui :)")

},function(error){
  console.log("Epäonnistui :(")

})
```

Kuva 5. AngularJS $http-kutsu

3 Sovelluksen kehittäminen

Tässä luvussa käydään läpi hybridisovelluksen kehitys Android-käyttöjärjestelmälle, kuvataan valittuja teknologioita, sekä kerrotaan miten ja miksi tiettyihin ratkaisuihin päädyttiin. Luvussa tuodaan esille kehityksessä ilmenneitä ongelmia ja haasteita sekä se, miten ne ratkaistiin. Tarkoituksena on antaa lukijalle kuva kehitysprosessista kokonaisuudessaan ja siitä minkälaiseen lopputulokseen päädyttiin.

3.1 Työvälineet

Kehityksessä käytetyt työvälineet valittiin niiden sopivuuden ja ennestään olemassa olleiden taitojen perusteella. Koska tekniset taidot ovat Web-kehittämisen puolella ja natiivien mobiilisovelluksien kehittämisestä ei ole kokemusta, Cordovan valinta sovelluksen kehittämiseksi oli luontaista, varsinkin koska se oli tullut tutuksi koulussa tehdyssä ohjelmistoprojektissa.

Cordovan mukana ei tule käyttöliittymäkomponentteja, joten natiivin ulkoasun saavuttamiseksi valittiin Onsen Ui, jonka valmiit komponentit pienettävät selvästi kehitysaikaa. Myös Onsen Ui oli tuttu edellisistä projekteista, vaikkakaan sitä ei tullut käytettyä samassa määrin.

Koska kehitys tapahtuu Web-teknologioita käyttäen, rinnalle haluttiin sopiva JavaScript-kehys. AngularJS:ään päädyttiin, sen yhteensopivuuden ja koska sen vahvuus on juuri yksisivuisten sovelluksien kehittämisessä. Myös runsas kokemus sen käytössä auttoi valinnan tekemisessä. Toisena vaihtoehtona oli AngularJS 2, mutta sen oli vielä uusi eikä siitä ollut kokemusta, mikä olisi tarkoittanut kehityksen hidasta alkua.

Sovelluksen kehittäminen Web-ympäristössä mahdollisti kehityksen tietokoneen selainta testialustana hyödyntäen. Jotta paikallista testausta voitiin suorittaa, tietokoneelle asennettiin palvelinpuolen ohjelmiston kehittämisessä käytetty Node.js ja MongoDB. Laajempaa testausta varten oli kuitenkin asennettava Android Studio, jotta sovellusta voitiin rakentaa ja emuloida Androidin natiivia ympäristöä. Tarkemmat testit suoritettiin Android-puhelimella käyttämällä kehittäjätyökalujen USB debugging-ominaisuutta, jolla sovellus pystyttiin rakentamaan suoraan puhelimeen. Chrome-kehittäjätyökalujen kanssa sovelluksen konsolia voitiin tarkkailla tietokoneen selaimen avulla.

3.2 Sovelluksen luominen

Sovellus luotiin Cordovalla komentosarjan kautta kuvan 6 komennolla. Komento luo sovellukselle yksinkertaisen pohjan kansiorakenteineen, jonka voi välittömästi avata selaimessa. Sovelluksen luomisessa käytettävä id ja nimi eivät ole lopulliset ja ne voidaan vaihtaa myöhemmin. Id:n pitää kuitenkin seurata nimeämiskäytäntöjä, joiden mukaan sen pitää olla vähintään kaksiosainen ja osat pitää erottaa pisteellä (Developer.android 2017b.) Esimerkiksi kehitettävän DinnerChatter sovelluksen id on com.app.dinnerchatter.

```
cordova create {{KANSION NIMI}} {{SOVELLUKSEN ID}} {{SOVELLUKSEN NIMI}}
```

Kuva 6. Cordova-sovelluksen luonti.

Luomisen jälkeen sovellukseen voitiin lisätä Android kehitettäväksi alustaksi, tämä tapahtui kuvan 7 komennolla. Komento lisää sovellukseen kaikki alustakohtaiset tiedostot.

```
cordova platform add android
```

Kuva 7. Android-alusta lisäys sovellukseen.

Jotta sovellusta voitiin rakentaa ja ajaa Android-ympäristössä oli asennettava Java Development Kit sekä Android Studio. Tämän jälkeen sovellus voitiin rakentaa kuvan 8 komennolla. Kehitys tehtiin pääasiassa sovelluksen www-kansiossa, josta Cordova rakentaa sovelluksen build-komennon ajettaessa.

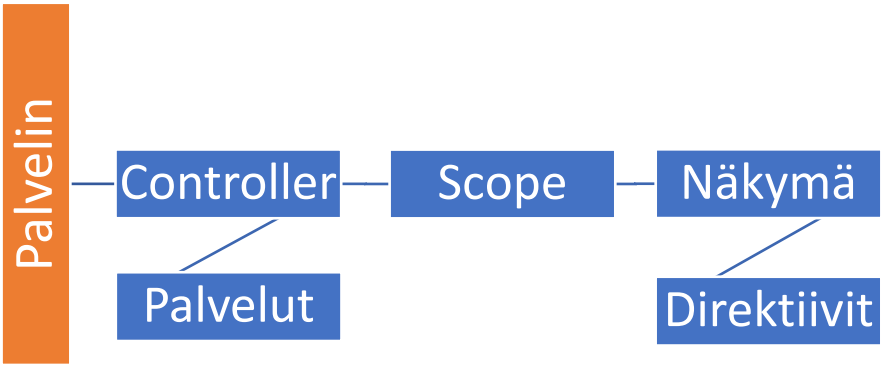
```
cordova build android
```

Kuva 8. Sovelluksen rakentaminen.

3.3 Sovelluksen rakenne

Sovellus on yksisivuinen (single page application), tarkoittaen kaiken sisällön lataamista kerralla ilman tarvetta päivittää näkymää käytön aikana. Kehitetyssä sovelluksessa yksisivuisuus saavutettiin käyttöliittymäkehys, Onsen UI:n, avulla. Kehys sisältää kustomoituja HTML-elementtejä, jotka mahdollistavat näkymien määrittämisen yhden HTML-tiedoston sisälle sekä näkymien vaihtamisen natiivilta vaikuttavia animaatioita käyttäen. Jokainen näkymä olisi vaihtoehtoisesti voitu määrittää omiin tiedostoihin, johon sovelluksen kokoon nähden ei ollut tarvetta.

Kuviossa 1 kuvataan sovelluksen rakennetta. MVC-arkkitehtuuria seuraten jokaiselle näkymällä määritettiin oma Controller rakenteen selkeyttämiseksi ja yhden Controllerin taakan keventämiseksi. Sovelluksen koon vuoksi suurin osa tiedon siirrosta ja käsittelystä tapahtui kunkin näkymän Controller-tiedostossa. Usein sovelluksen koodissa esiintyvät funktiot ja useiden näkyminen välinen tiedonjakaminen siirrettiin erillisiin palvelu-objekteihin, näin välttyen liialliselta toistamiselta. Näkymissä usein esiintyvät elementit tai muuten dynaamisesti lisättävä sisältö luodaan direktiiveissä, jotka ovat käytettävissä kaikissa näkymissä.



Kuvio 1. Sovelluksen rakenne.

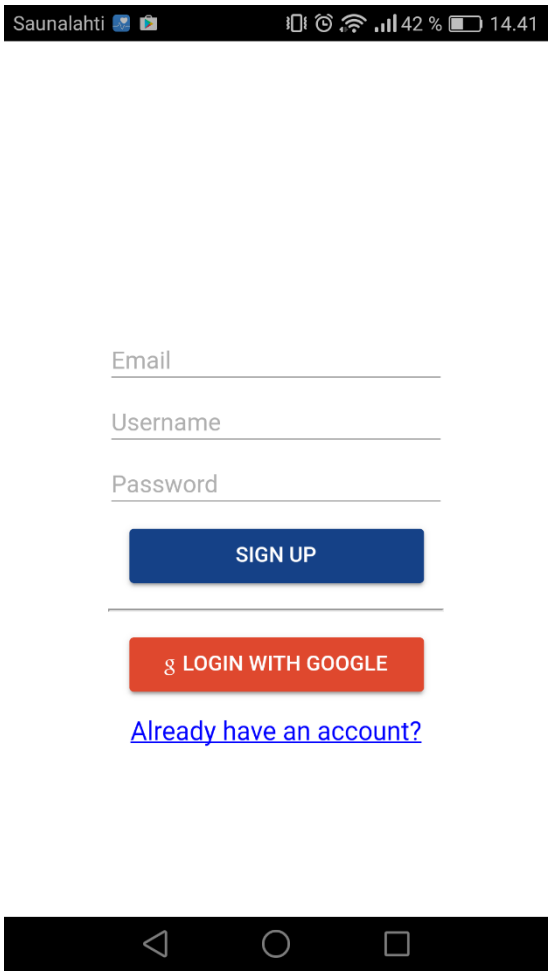
3.4 Käyttöliittymän kehittäminen

Sovelluksen käyttöliittymää lähdettiin rakentamaan yksinkertaisuuden näkökulmasta. Koska projektilla ei ollut erillistä asiakasta, kädet olivat vapaat suunnittelun kannalta. Kehityksessä hyödynnettiin paljon Onsen Ui:n valmiita käyttöliittymäkomponentteja, joita muokattiin omien tarpeiden mukaan. Haasteeksi ilmeni edellä mainittujen komponenttien muokkaaminen, koska ne ovat rakenteeltaan monimutkaisia ja koska !important-liitteen käyttö hankaloitti CSS-määrityksien yliajamista. Komponenttien muokkaamisessa tavoitteena oli kuitenkin säilyttää sovelluksen natiivi tuntuma. Varsinkin animaatiot ja navigaatio-elementit jätettiin pääosin ennalleen joitakin värimuutoksia lukuun ottamatta.

3.4.1 Kirjautumisnäkymä

Kuvassa 7 on sovelluksen kirjautumisnäkymä, joka tarjoaa käyttäjälle kaksi vaihtoehtoa, joko kirjautumisen sovelluksen sisäistä palvelua käyttäen tai tunnistautumisen Google-tilin avulla. Google-kirjautuminen on tehty käyttäen Cordovan cordova-plugin-googleplus kolmannen osapuolen liitännäistä, joka tarjoaa JavaScript API:n kirjautumisen suorittamiseksi. Käyttäjän käynnistäessä sovelluksen aina yritetään niin kutsuttua hiljaista kirjau-

tumista, jotta käyttäjä voidaan siirtää suoraan sovellukseen ilman ylimääräistä kanssakäymistä (GitHub 2017.) Jos käyttäjä luo tilin sovelluksen sisäisellä toiminnolla, syötetty sähköposti on vahvistettava ennen sisäänkirjautumista.



Kuva 9. Sovelluksen kirjautumisnäkymä.

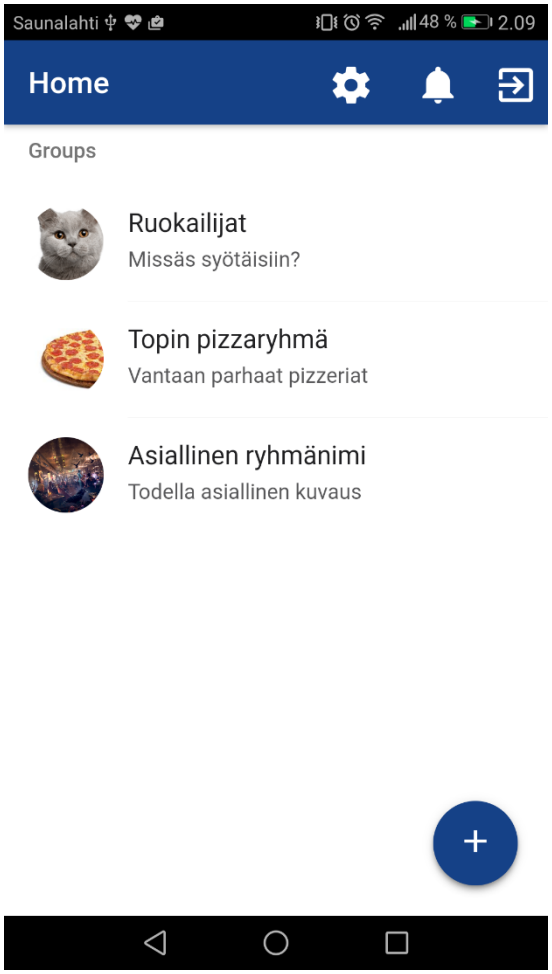
Käyttäjän kirjautuessa sovellukseen avataan yhteys Socket.IO-palveluun, jolla suoritetaan reaaliaikainen tiedonsiirto, jota käytetään pääasiassa chat-palvelussa. Koska kirjautumistapoja on useita, palveluun yhdistäminen siirrettiin erilliseen AngularJs:n service-objektiin, jota voidaan kutsua tarvittaessa sovelluksen jokaisesta Controller-objektista. Kuvan 10 objekti sisältää 3 funktiota. Ensimmäisessä avataan uusi Socket.IO yhteys ja tallennetaan sen palauttama objekti, toista funktiota kutsuttaessa palautetaan yhteysobjekti esimerkiksi silloin, kun tarkoituksena on liittyä chat-ryhmää, ja kolmas funktio lopettaa yhteyden, jota käytetään käyttäjän uloskirjautumisessa tai tunnistautumisvirheen ilmetessä. Sovelluksen yksisivuisuuden ansiosta AngularJs:n palvelu sopii väliaikaisen tiedon ylläpitämiseen käytön aikana, ja se mahdollisti edellä mainitussa esimerkissä yhteysobjektin säilyttämiseen.

```
app.service('socket', function(address){
    var address = address.getAddress();
    var socket = "";
    this.connectUser = function(){
        socket = io.connect(address, {
            query: {token: localStorage.token}
        });
        console.log("user connected")
    }

    this.getConnection = function(){
        return socket;
    }

    this.disconnectUser = function(){
        socket.disconnect();
        console.log("disconnected")
    }

});
```

Kuva 10. AngularJs service Socket.IO:n käytöön

3.4.2 Kotinäkymä

Onnistuneen kirjautumisen jälkeen käyttäjä ohjataan kuvassa 11 näkyvään sovelluksen kotinäkymään, jonka kautta käyttäjällä on pääsy kaikkiin omaisuuksiin. Näkymässä on listattu jokainen ryhmä, joissa käyttäjä on jäsenenä. Sovelluksen yläpalkista löytyvät ryhmäkutsut, käyttäjän asetukset ja uloskirjautuminen. Uloskirjautuminen on ainoa yläpalkin valinta, joka on käytettävissä jokaisessa näkymässä. Jos käyttäjä on kirjautunut aiemmin sisään ja hänet pystytään tunnistamaan, sovellus siirretään suoraan kotinäkymään.

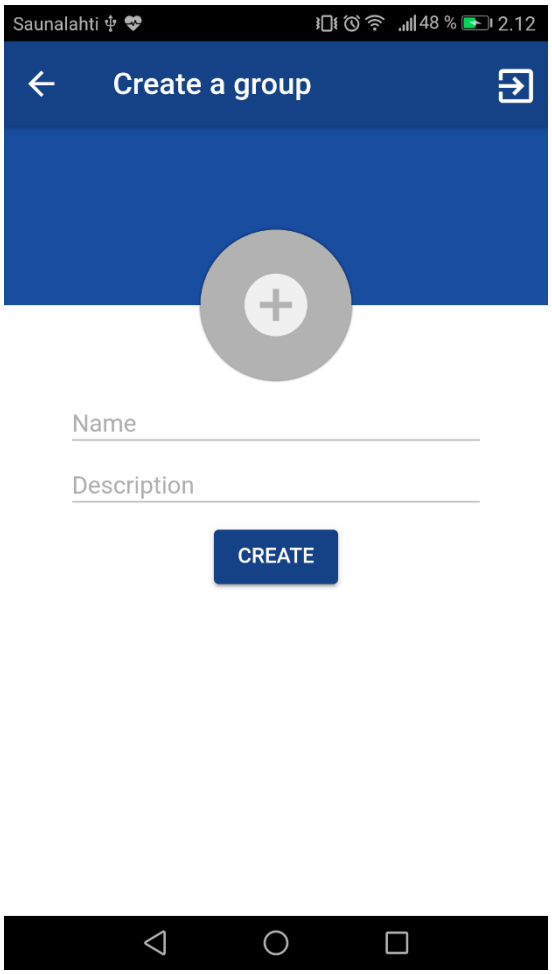


Kuva 11. Sovelluksen kotinäkymä.

Jotta sovelluksesta uloskirjautuminen voitiin sijoittaa jokaiseen näkymään nopeasti ilman ylimääräistä työtä, luotiin kustomoitu AngularJS direktiivi. Direktiivi lisää haluttuun kohtaan uloskirjautumiseen viittaavan ikonin, jota painamalla käyttäjälle näytetään vahvistusviesti. Vahvistusviesti luodaan ons.notification-metodia käyttäen, ja käyttäjän hyväksyessä sen kaikki kirjautumistiedot poistetaan ja hänet siirretään takaisin kirjautumisnäkymään. Kirjautumisen yhteydessä avattu Socket.IO-yhteys katkaistaan, jotta käyttäjä ei pysty vastaanottamaan viestejä sovelluksen ulkopuolella.

3.4.3 Ryhmän luontinäkymä

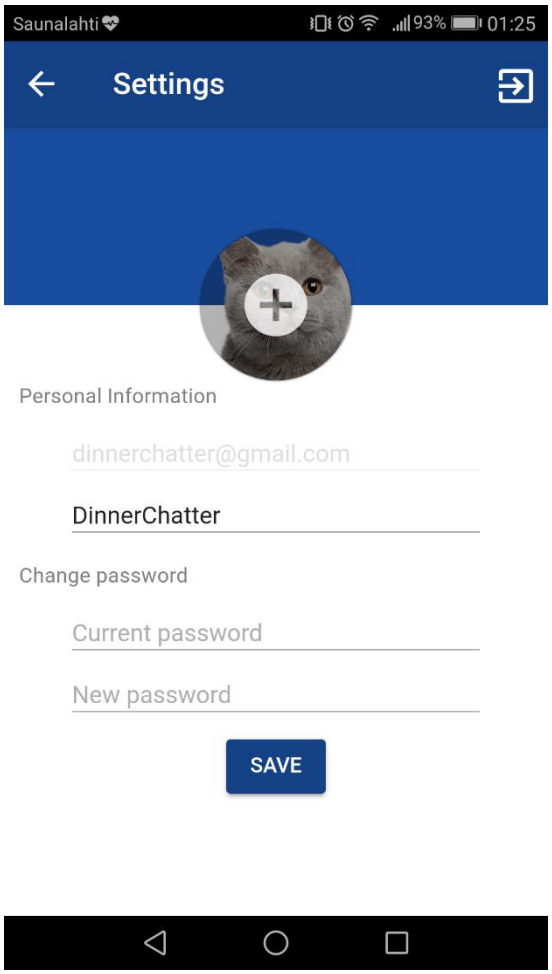
Kuvan 12 näkymässä käyttäjä voi luoda uuden ryhmän. Pakollisina kenttinä on ainoastaan nimi, mutta listan yhtenäisyyden kannalta on suositeltavaa, että kuvaus ja kuva lisätään, koska niille ei ole väliaikaisia arvoja. Kuvien koossa ei ole rajoitteita Android-sovelluksen puolella. Sovellus rajoittaa lisättävien tiedostojen tyypin vain kuviksi, tämä tehdään sekä paikallisesti, että palvelimella. Rajoitteiden puutteen vuoksi kuvien lataamisessa saattaa kestää, mikä hidastaa ryhmän luomista. Käyttäjä ei pysty vielä luontinäkymässä lisäämään ryhmään uusia käyttäjiä, vaan tämä tehdään ryhmän asetuksissa.



Kuva 12. Ryhmän luontinäkymä.

3.4.4 Käyttäjän asetusnäkymä

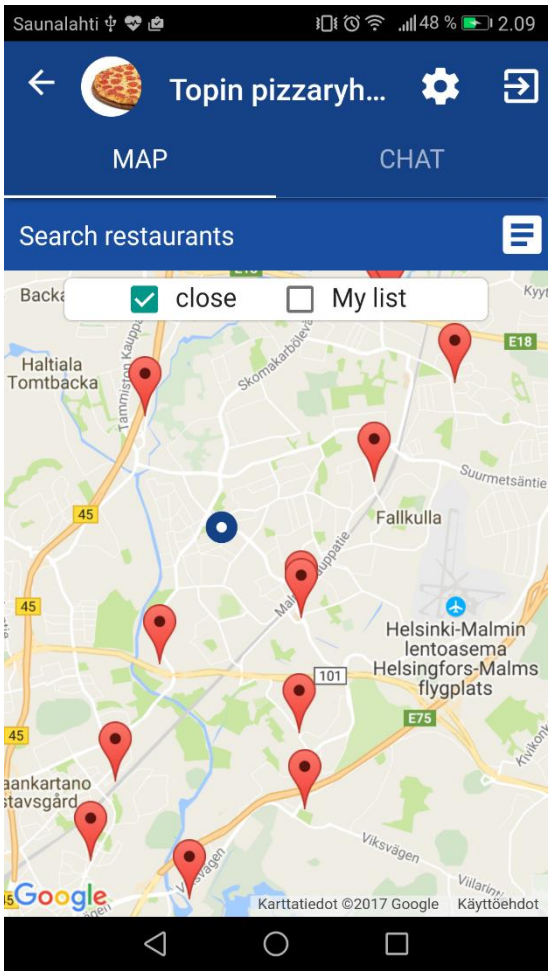
Kuvan 13 käyttäjän asetusnäkymä on pyritty pitämään mahdollisimman yksinkertaisena. Käyttäjä pystyy lisäämään itselleen profiilikuvan sekä vaihtamaan näkyvän käyttäjänimen ja salasanan. Sähköpostin vaihto ei ole nykyisessä versiossa mahdollista. Profiilikuvaa painamalla aukeaa käyttäjän tiedostojenhallinta, jossa valinta on rajattu vain kuviin. Kaikki muutokset tehdään vasta save-nappia painamalla, joten esimerkiksi kuva ei tallennu välittömästi valinnan jälkeen. Jos käyttäjä on kirjautunut Google-tiliään käyttäen niin salasanan vaihto ei ole luontaisesti mahdollista.



Kuva 13. Käyttäjän asetusnäkymä.

3.4.5 Ryhmänäkymä - kartta

Kun käyttäjä valitsee ryhmän listasta, hänet viedään kuvan 14 näkymään. Tässä näkymässä käyttäjä voi etsiä ravintoloita, jotka on mahdollista lisätä ryhmäkohtaiseen listaan. Käyttäjän lisäämät ravintolat esitetään kartassa ja erillisessä listanäkymässä, jossa tarkastellaan ravintoloiden tarkempia tietoja. Alun perin lista ja kartta olivat erillisillä sivuilla mutta ne yhdistettiin käytön yksinkertaistamiseksi. Kartalla voi vaihtoehtoisesti näyttää käyttäjän läheltä löytyviä ravintoloita. Kartta ja ravintoloiden etsiminen toteutettiin Google Mapsin ja Places API:n avulla. Niiden rajoitteet toivat omat haasteensa, esimerkiksi Googlen käyttöehtojen mukaan ravintolahauista sai tallentaa ainoastaan jokaisen paikan uniikin id:n (developers.google, 2017). Tämä lisäsi ylimäärisiä askelia paikkojen näyttämisessä kaikille ryhmän käyttäjille, sillä jokainen id joudutaan muuttamaan paikaksi kunkin käyttäjän mobiililaitteessa.



Kuva 14. Ryhmän karttanäkymä.

Kuvassa 15 on käyttäjän sijainnin hakua ja palauttamista varten luotu AngularJS:n palvelu. Objektin ensimmäinen funktio paikantaa käyttäjän Cordovan cordova-plugin-geolocation-liitännäisen avulla, käyttämällä puhelimen omaa GPS-ominaisuutta. Funktiossa

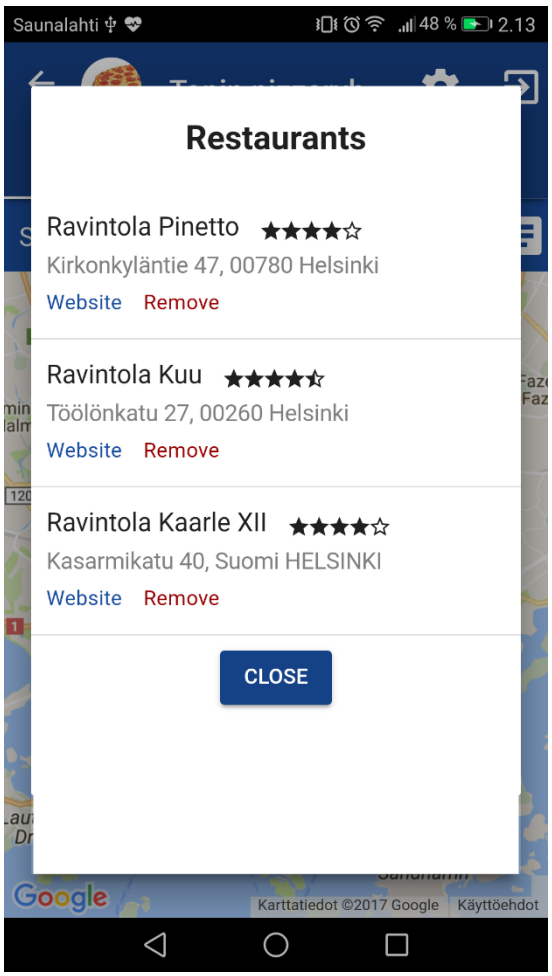
hyödynnetään $q-palvelua, joka auttaa tiedonkäsittelyssä asynkronisesti, ja varmistaa palautetun tiedon käsittelyn välittömästi, kun se vastaanotetaan. Käyttäjän sijainnin hakemisen jälkeen koordinaatit palautetaan. Jos syystä tai toisesta haku epäonnistuu, virheiden välttämiseksi sijainnin vakioarvoksi on asetettu Helsingin keskusta. Palvelun toista funktiota käytetään, kun tarvitaan käyttäjän koordinaatit ja tiedetään paikantamisen tapahtuneen vain hetkeä enne. Karttanäkymän kartta keskitetään haettujen koordinaattien avulla, ja käyttäjän sijainnin kohdalle asetetaan merkki.

```
app.service('geolocation', function($q){
    var userLocation = {lat: 60.1699, lng: 24.9384};

    this.getPosition = function(){
        document.addEventListener("deviceready", onDeviceReady, false);
        function onDeviceReady() {
            var deferred = $q.defer();
            var onSuccess = function(position){
                userLocation = {lat: position.coords.latitude, lng: position.coords.longitude};
                deferred.resolve({lat: position.coords.latitude, lng: position.coords.longitude});
            }
            var onError = function(error){
                deferred.reject(error);
            }
            navigator.geolocation.getCurrentPosition(onSuccess, onError);

            return deferred.promise;
        }
    }

    this.returnPosition = function(){
        return userLocation;
    }
});
```

Kuva 15. Käyttäjän sijainnin haku.

3.4.6 Ryhmänäkymä - ravintolalista

Kuvassa 16 on käyttäjien luoma ravintolalista. Listan avulla käyttäjät pääsevät käsiksi ravintoloiden oleellisimpaan tietoon, kuten nimeen, osoitteeseen, ravintoloiden saamaan arvioon ja linkkiin ravinolan verkkosivuille. Lisäksi käyttäjä voi poistaa ravintoloita listasta. Listan tarkoituksena on tarjota käyttäjille vaihtoehtoinen tapa ravintoloiden tarkasteluun ja selkeyttää niiden hahmottamista. Esitettyjen tietojen suhteen on hyvä muistaa, että ne tulevat suoraan Googlelta, joten eroavaisuuksia saattaa ilmetä, esimerkiksi katujen nimet ovat joissakin tapauksissa ruotsiksi tai postinumeroa ei ole listattu.



Kuva 16. Karttanäkymän ravintolalista.

Ravintolalistassa näytetään kunkin ravintolan saamat arvostelut tähtinä. Jotta tähden pystytään luomaan dynaamisesti riippuen jokaisen ravintolan saamista arvosteluista, oli luotava kustomoitu direktiivi. Kun direktiivi liitetään HTML-elementtiin, siihen on lisättävä ravintolan saama arvio, joka on luku yhdestä viiteen. Kuvan 17 koodi ottaa vastaan ravintolan arvion ja muuttaa sen näytettäväksi tähtinä näkymän puolella. Jokaisen ravintolan kohdalle asetetaan yhteensä viisi tähteä, ja yksi tähti voi olla joko tyhjä, puolikas tai täysi.

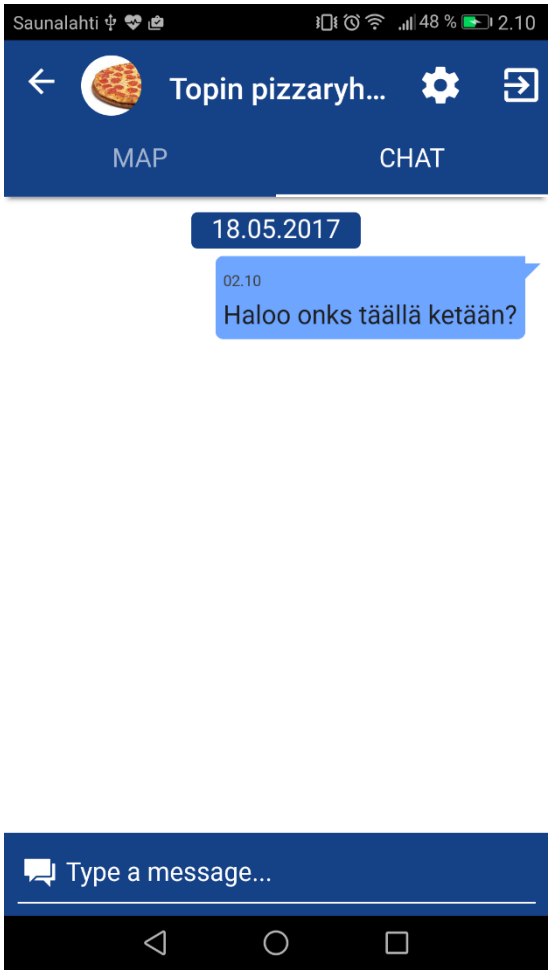
Ravintolan saama arvio käydään läpi ja muuttujaan asetetaan tarvittavat tähti-ikonit, minkä jälkeen muuttuja liitetään näkymään lisättävään sisältöön. AngularJS:n normaali ng-bind-direktiivi ei mahdollista HTML-elementtien lisäämistä sivulle $scope-objektiin sidottujen muuttujien kautta turvallisuussyistä. Ng-bind-hml sallii tämän mutta riisuu elementistä kaiken JavaScript-koodin.

```
app.directive('resRating', function() {
  return {
    restrict: 'E',
    scope: {
        rating: '@'
    },
    controller: function ($scope, $sce) {
        $scope.x = "";
        var stars = (Math.round($scope.rating * 2) / 2).toFixed(1);
        for(i = 0; i < 5 ; i++){
            if(stars>=1){
                $scope.x=$scope.x + '<ons-icon icon="ion-android-star"></ons-icon>';
                stars=stars-1;
            }
            else if(stars==0.5){
                $scope.x=$scope.x + '<ons-icon icon="ion-android-star-half"></ons-icon>';
                stars=stars-0.5;
            }
            else if(stars==0){
                $scope.x=$scope.x + '<ons-icon icon="ion-android-star-outline"></ons-icon>';
            }
        }
        $scope.x = $sce.trustAsHtml($scope.x);
    },
    template: '<span ng-bind-html="x"></span>'
  };
});
```

Kuva 17. Ravintolan arvostelun muuttaminen tähdiksi.

3.4.7 Ryhmänäkymä - chat

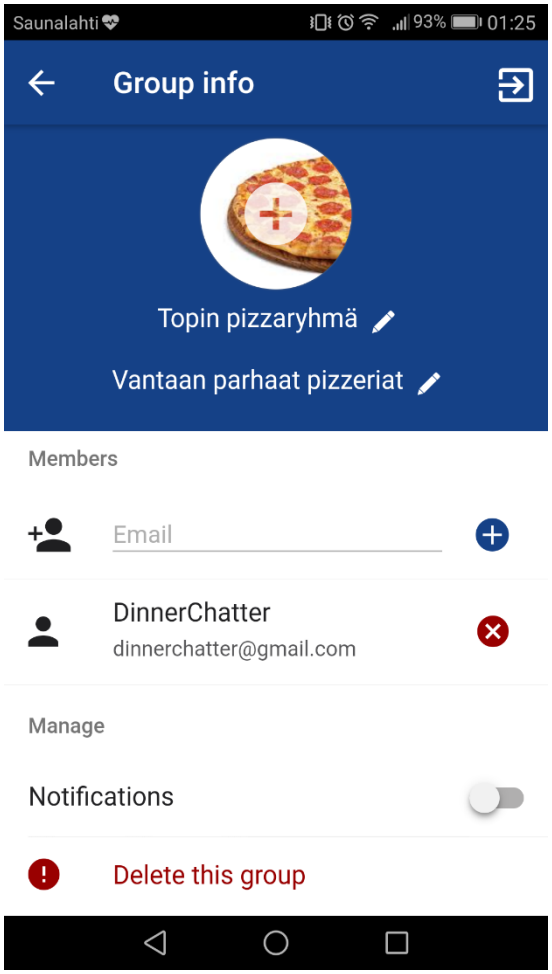
Toinen ryhmän päänäkymistä on chat, joka on kuvassa 18. Chat mahdollistaa reaaliaikaisen keskustelun ryhmän jäsenien välillä, jotta käyttäjät voivat tulla paremmin yhteisymmärrykseen ruokapaikasta. Kaikki viestit tallennetaan tietokantaan, joten ne ovat myös saatavilla uusia käyttäjiä lisättäessä tai käyttäjien palatessa sovellukseen. Chatin reaaliaikainen tiedonsiirto on toteutettu Node.js:n Socket.IO JavaScript-kirjastolla.



Kuva 18. Toinen ryhmän päänäkymistä.

3.4.8 Ryhmän hallintanäkymä

Kuvan 19. Ryhmänhallintanäkymässä hallinnoidaan ryhmän tietoja ja käyttäjiä. Sellaiset ominaisuudet kuten ryhmän ja sen jäsenien poistaminen, sekä nimen, kuvauksen ja profiilikuvan muuttaminen ovat käytettävissä vain ryhmän luoneella käyttäjällä. Ryhmän muut jäsenet pystyvät lisäämään käyttäjiä ja poistumaan itse ryhmästä. Lisääminen tapahtuu syöttämällä käyttäjän sähköpostiosoite, jonka jälkeen lisätyn käyttäjän on hyväksyttävä kutsu, ryhmään liittyäkseen.



Kuva 19. Ryhmän hallintanäkymä.

3.5 Palvelinpuolen integraatio

Chat-palvelua lukuun ottamatta kommunikaatio palvelimen kanssa toteutettiin AngularJS:n $http-palvelulla, joka käyttää XMLHttpRequest API:a tiedon viemiseen ja hakemiseen. Näkymiin liittyvät pyynnöt tehdään niiden Controller-objekteissa, joissa palvelimelta palautettava tieto myös käsitellään. Tilin luomisessa ja sisäänkirjautumisessa palvelin palauttaa todennus tokenin, joka tallennetaan paikallisesti käyttämällä HTML Local Storage-palvelua. Tämä token lisätään kirjautumisen jälkeen jokaiseen $http-kutsuun, jotta palvelin

pystyy varmistamaan käyttäjän oikeudet. Kuvassa 20 on osa AngularJS palvelua, joka kuuntelee kaikkia palvelimelle lähetettyjä kutsuja ja sieltä tulevia vastauksia. Jos käyttäjältä ei löydy tarvittavia oikeuksia kutsujen tekemiseen, kuvan koodi kaappaa palvelimelta tulevat 401-virhekoodin sisältävät vastaukset, minkä jälkeen käyttäjälle ilmoitetaan tapahtuneesta virheestä ja hänet ohjataan takaisin kirjautumisnäkymään. Myös token poistetaan, jotta sama virhe ei enää toistu ja käyttäjä pakotetaan kirjautumaan uudestaan.

```
'responseError': function(rejection) {
    if(rejection.status == 401&&check==false){
     console.log("forbidden");
     check=true;
     localStorage.removeItem("token");
     localStorage.removeItem("userid");

     ons.notification.alert({
         message: "Authentication error",
         callback: function(){
             myNavigator.resetToPage("signup.html");
             check=false;
         }
     })
   }
   return $q.reject(rejection);
 },
```

Kuva 20. Kaappaa palvelimelta tulevat virheet

Socket.IO JavaScript-kirjastoa käytetään reaaliaikaisen tiedon välittämiseen. Suurin osa tästä tapahtuu palvelimella, mutta sovelluksen on ensin avattava yhteys, jonka jälkeen palvelin voi puskea tietoa. Yhteys avataan aina käyttäjän kirjautuessa sisään sovellukseen, ja ryhmäkohtaiseen chat-huoneeseen liitytään käyttäjän valitessa ryhmän.

3.6 Google-palvelut

Googlen tarjoamat palvelut ovat olennainen osa sovelluksen toimintoja. Googlen käyttöehtojen tarkkuuden takia alun perin suunniteltuja toimintoja jouduttiin jonkin verran muokkaamaan. Sovellukseen käyttäjät voivat vaihtoehtoisesti kirjautua sisään Google-tilillään, ja koska Android sekä Google Play-palvelu ovat vahvasti sidoksissa Googlen kanssa, on oletettavaa, että suurin osa käyttäjistä valitsee tämän vaihtoehdon sisäisen kirjautumisen sijaan. Google Sign-In toteutettiin kolmannen osapuolen Cordova-liitännäisellä, Googlen lopettaen tuen WebView-kirjautumisissa (developers.googleblog 2016). Tämä kolmannen osapuolen liitännäinen käyttää Androidin natiivia kirjautumista JavaScript API:n kautta. Kirjautumisesta palautettu käyttäjän token vahvistetaan vielä palvelimen puolella henkilöllisyyden varmistamiseksi.

Muita merkittäviä sovelluksen osia, jotka hyödyntävät Googlen palveluita, on kartta ja ravintoloiden etsiminen. Nämä toiminnallisuudet pystyttiin toteuttamaan Googlen JavaScript API:lla, mutta silti ne tuottivat päänvaivaa muun muassa käyttöehtojen asettamien rajoitteiden takia. Googlen kartan näyttämisessä ongelmana oli sen luominen näkymän ollessa piilotettuna, jolloin kartta ei latautunut. Kartta piti manuaalisesti keskittää sen jälkeen, kun käyttäjä oli siirtynyt näkymään, jotta ongelmalta vältyttiin. Ravintolahakujen tuloksista sai Googlen käyttöehtojen mukaan tallentaa vain kunkin paikan uniikin id:n. Kun haluttiin päästä käsiksi ravintoloiden tietoihin, ne oli haettava Googlelta kyseisen id:n avulla. Google on asettanut paikkatietojen hakemiseen käyttörajoitteita. Nämä rajoitteet tuskin tulevat vaikuttamaan sovelluksen käyttöön, mutta ne on hyvä kuitenkin pitää mielessä, jos käyttö ylittää odotukset.

3.7 Sovelluksen julkaisu

Sovellus julkaistiin Google Play palvelussa, joka on tarkoitettu sovelluksien jakeluun Android-alustalla. Palvelussa oli ensin rekisteröidyttävä kehittäjäksi, mikä vaati 25 dollarin kertamaksun. Maksun jälkeen sovelluksia voi julkaista rajattomasti palvelun kautta.

Rekisteröitymisen jälkeen Google Play konsolissa voitiin luoda uusi sovellus, jolle annettiin nimeksi DinnerChatter. Sovelluksesta täytettiin pakolliset tiedot, jotka sisälsivät muun muassa kuvauksen, sovelluksen tyypin ja kuvankaappauksia sovelluksesta. Sovelluksen tietoihin liitettiin käyttöehdot, jotka kertovat käyttäjälle, mitä käytöltä tulee odottaa ja kuinka hänen tietonsa tallennetaan, sekä sen, miten niitä käytetään. Käyttöehdot myös suojaavat kehittäjiä mahdollisilta ongelmilta julkaisun jälkeen.

Sovelluksen tiedostot oli muunnettava APK (Android application package) -tiedostomuotoon, ja allekirjoitettava ennen kuin se voitiin julkaista. Allekirjoittaminen, joka perustuu julkisen avaimen suojaukseen, suojaa sovellusta ja varmistaa, että kaikki päivitykset tulevat aina sovelluksen oikealta kehittäjältä (developer.android 2017b). Jotta sovellus pystyttiin allekirjoittamaan, oli luotava sertifikaatti, joka tehtiin menemällä komentosarjassa JDK:n (Java Development Kit) bin-kansioon ja ajamalla kuvan 21 komento. Komennon jälkeen oli syötettävä joitakin tietoja, mukaan lukien sertifikaatin ja yksityisen avaimen salasanat. Salasanat on hyvä pitää mielessä, koska niitä tarvitaan allekirjoittamisessa.

```
keytool -genkey -v -keystore {SERTIFIKAATTI}.jks
-keyalg RSA -keysize 2048 -validity 10000 -alias {SERTIFIKAATIN NIMI}
```

Kuva 21. Sertifikaatin luonti.

Kun sertifikaatti oli luotu, sovellus voitiin allekirjoittaa, mikä tässä tapauksessa tehtiin Cordovan avulla ajamalla kuvan 22 komento sovelluksen kansiossa. Komento rakentaa ja allekirjoittaa sovelluksen käyttäen edellä luotua sertifikaattia, ja luo siitä julkaisukelpoisen APK-tiedoston.

```
cordova build android --release --
--keystore={SERTIFIKAATTI}.jks --storePassword={SERTIFIKAATIN SALASANA}
--alias={SERTIFIKAATIN NIMI} --password={YKSITYISEN AVAIMEN SALASANA}
```

Kuva 22. Sovelluksen rakentaminen ja allekirjoitus.

Allekirjoitettu APK-tiedosto lisättiin Google Play konsolissa sijaitsevaan sovellukseen, jonka jälkeen sovellus oli julkaisuvalmis. Sovellus julkaistiin alfa-versiona, jossa sille suoritettaan viimeisiä hienosäätöjä ja varmistetaan sen toimivuus. Julkaisun jälkeen kului useita tunteja ennen kuin sovellus oli näkyvissä Google Playssa, minkä jälkeen se oli ladattavissa kaikille käyttäjille.

4 Pohdintaa ja jatkokehitysideat

Tässä luvussa käsitellään kehityksen aikana nousseet mietteet ja ongelmat teknologioiden sekä koko kehitysprosessiin suhteen. Luvussa pohditaan myös sovelluksen tulevaisuutta ja jatkokehitysmahdollisuuksia.

4.1 Pohdintaa kehityksestä

Sovelluksen ennalta määritetyt ominaisuudet saatiin valmiiksi juuri ennen opinnäytetyön päättämistä, joten sitä ei päästy testaamaan laajemmassa mittakaavassa. Myöhäinen julkaisu ja testaamattomuus synnyttivät useita kysymysmerkkejä sovelluksen toimivuuden ja luotettavuuden suhteen. Jotta käyttäjät ottaisivat tämän huomioon ja antaisivat palautetta käytettävyydestä, sovellus julkaistiin alfa-versiona.

Sovelluksen kehittämisen aikataulua olisi voinut hallita paremmin. Projektin alkupuolella kehitykseen käytettiin aivan liian vähän aikaa päivittäin, mikä kostautui projektin loppupuolella. Yhteistyö toisen kehittäjän kanssa sujui hyvin koko projektin ajan, ennalta sovitut kehitysajat ja tiivis kommunikointi auttoivat saaman projektin valmiiksi.

Käytetyt teknologiat soveltuivat kehitykseen hyvin OnsenUI:n pienestä yhteisöstä johtuvaa avun vähäisyyttä lukuun ottamatta, teknologioihin ongelmia ei liittynyt. Kehitys syvensi Cordovan taitoja huomattavasti ja perehdytti hyvin tarkasti Android-sovelluksien vaatimuksiin ja kehittämiseen. Jälkeenpäin pohtiessa Ionic olisi voinut olla varteenotettava vaihtoehto sovelluksen hybridikehykseksi, koska kehitys painottui vahvasti AngularJS:n puolelle. Mielenkiintoiseksi osoittautui erot sovelluksen käyttäytymisessä tietokoneen selaimen ja puhelimen välillä, tässä suurimmaksi haasteeksi osoittautui näppäimistön ja muiden puhelimen komponenttien käyttäytyminen.

Teknologiavalinnoista eniten jäi mietityttämään AngularJS, vaikkakin se toimi ongelmitta. Täysin uuden teknologian opiskelu ja käyttöönotto olisi voinut olla hyvinkin mielenkiintoista ja tarjota uuden näkökulman kehittämiseen.

Projektia voi pitää kokonaisuudessaan onnistuneena vaikkakin aika loppui hieman kesken. Kehityksen aikana tuli opittua valtavasti. Projektia on vaikea verrata koulussa osallistuttuihin ohjelmistokehityskursseihin, koska sovellus kehitettiin kahden kehittäjän voimin.

Työhön panostettu aika ja valtava vastuu ovat vertaansa vailla ja samalla arvokasta kokemusta tulevaisuuden varalle.

4.2 Sovelluksen jatkokehitys

Sovelluksen kehitystä tullaan jatkamaan opinnäytetyön ulkopuolellakin. Tarkoituksena on aluksi testata sen toimintaa laajemmin ja hioa toiminnallisuudet parhaiksi mahdollisiksi. Kehityksen aikana saatiin ideoita uusien toiminnallisuuksien suhteen mutta tiukan aikataulun takia ne jäivät toteuttamatta. Näiden toiminnallisuuksien toteuttaminen riippuu täysin omasta halusta jatkaa kehittämistä ja yleisestä sovellukseen kohdistuvasta mielenkiinnosta. Sovellus on ladattavissa ilmaiseksi eikä se sisällä mainoksia, joten sen ylläpitäminen ilman kasvavaa käyttäjämäärää ei ole kannattavaa. Mainoksien näyttämistä sovelluksessa voidaan harkita, mikäli sovellus kerää tarpeeksi mielenkiintoa, jotta ylläpitokulut pystytään kattamaan

Tarkoituksena on tehdä merkittävä osa tiedonkäsittelystä reaaliaikaisesti, koska ryhmät ja niiden toiminnot perustuvat interaktiivisuuteen. Tällä hetkellä muun muassa ravintoloiden lisääminen tai poistaminen listasta näkyvät ryhmän muilla käyttäjillä vasta heidän päivitettyään näkymän. Ideana on myös tarkemmin perehtyä ilmoituksiin ja niiden näyttämiseen joko käyttäjän ollessa sovelluksessa tai puskea ilmoitukset silloinkin, kun käyttäjä ei ole aktiivisesti käyttämässä sovellusta.

Ravintoloiden listaamista ja niiden näyttämistä olisi hyvä parantaa, käyttäjien pitäisi päästä tutustumaan ravintoloihin tarkemmin sovelluksen sisällä. Esimerkiksi arvosteluiden ja ruokalistan näyttäminen voisi parantaa käyttökokemusta ja auttaa syömäpaikan päätöksessä. Yleisestikin Googlen tarjoamien palveluiden käyttäminen niiden täydellä potentiaalilla olisi ihanteellista. Käyttäjien sijaintia voisi seurata jatkuvasti ja jakaa se kaikille muille ryhmän jäsenille. Tämä auttaisi valitsemaan ravintoloita, jotka ovat sopivan matkan päässä kaikista jäsenistä. Käyttäjän sijainnin tallentaminen tosin avaisi uusia tietoturvaan liittyviä ongelmia. Ongelmaksi Googlen palveluiden lisääntyvässä käytössä voivat nousta käyttörajoitteet, joiden poistaminen sovelluksen nykyisessä tilassa ei ole taloudellisesti kannattavaa.

Käyttökokemuksen parantaminen on tärkeää. Sovellusta ei ole päästy testaamaan eri tehoisissa ja eri käyttöjärjestelmän omaavissa puhelimissa, joten optimointia saattaa olla edessä. Sovelluksen käytössä tapahtuvien virheiden ilmoittamista käyttäjälle pitää parantaa, nykyisessä järjestelmässä on hieman puutteita.

Lähteet

AngularJs 2017a. AngularJS: Developer Guide: Data Binding. Luettavissa: https://docs.angularjs.org/guide/databinding Luettu: 2.5.2017.

AngularJs 2017b. AngularJS: Developer Guide: Directives. Luettavissa: https://docs.angularjs.org/guide/directive Luettu: 2.5.2017.

AngularJs 2017c. AngularJS: Developer Guide: Controllers. Luettavissa: https://docs.angularjs.org/guide/controller Luettu: 4.5.2017.

AngularJs 2017d. AngularJS: Developer Guide: Scopes. Luettavissa: https://docs.angularjs.org/guide/scope Luettu: 6.5.2017.

AngularJs 2017e. AngularJS: API: $http. Luettavissa: https://docs.angularjs.org/api/ng/service/$http Luettu: 7.5.2017.

AppBrain 2017. Android app frameworks – AppBrain. Luettavissa: https://www.appbrain.com/stats/libraries/tag/app-framework/android-app-frameworks?sort=apps Luettu: 22.5.2017.

Code.tutplus 2016. An Introduction to Cordova: Basics. Luettavissa: https://code.tutsplus.com/tutorials/an-introduction-to-cordova-basics--cms-25146 Luettu: 23.4.2017.

Cordova 2017a. Architectural overview of Cordova platform - Apache Cordova. Luettavissa: https://cordova.apache.org/docs/en/latest/guide/overview/index.html Luettu: 23.4.2017.

Cordova 2017b. Cordova support by platform – Apache Cordova. Luettavissa: https://cordova.apache.org/docs/en/latest/guide/support/index.html#core-plugin-apis Luettu: 25.4.2017.

Cordova 2017c. Creating your first Cordova app - Apache Cordova. Luettavissa: https://cordova.apache.org/docs/en/latest/guide/cli/index.html Luettu: 5.5.2017.

Developer.android 2017a. Download Android Studio and SDK Tools | Android Studio. Luettavissa: https://developer.android.com/studio/index.html Luettu: 22.5.2017.

Developer.android 2017b. Set the Application ID | Android Studio. Luettavissa: https://developer.android.com/studio/build/application-id.html Luettu: 22.5.2017.

Developer.android 2017c. Sign Your App | Android Studio. Luettavissa: https://developer.android.com/studio/publish/app-signing.html Luettu: 20.5.2017.

Developer.salesforce 2016. Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options. Luettavissa: https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options Luettu: 24.4.2017.

Developers.google 2017. Places API Policies. Luettavissa: https://developers.google.com/places/web-service/policies Luettu: 9.5.2017.

Developers.googleblog 2016. Google Developers Blog: Modernizing OAuth interactions in Native Apps for Better Usability and Security. Luettavissa: https://developers.googleblog.com/2016/08/modernizing-oauth-interactions-in-native-apps.html Luettu: 15.5.2017.

GitHub 2017. Cordova plugin to login with Google Sign-In on iOS and Android. Luettavissa: https://github.com/EddyVerbruggen/cordova-plugin-googleplus#6-usage Luettu: 8.5.2017.

Ionicframework 2017. Building Hybrid Apps with AngularJS and Ionic. Luettavissa: http://ionicframework.com/present-ionic/slides/#/ Luettu: 22.5.2017.

Onsen Ui 2017. Onsen UI 2: Beautiful HTML5 Hybrid Mobile App Framework and Tools. Luettavissa: https://onsen.io/ Luettu: 28.4.2017.

Stack overflow 2017. Frequent 'cordova' Questions - Stack Overflow. Luettavissa: http://stackoverflow.com/questions/tagged/cordova Luettu: 28.4.2017.

Tutorialspoint 2017. AngularJS Overview. Luettavissa: https://www.tutorialspoint.com/angularjs/angularjs_overview.htm Luettu: 2.5.2017.

Wikipedia 2017. Apache Cordova – Wikipedia. Luettavissa: https://en.wikipedia.org/wiki/Apache_Cordova Luettu: 22.5.2017.

Liitteet

Liite 1. Linkit sovelluksen sivuille

GitHub: https://github.com/TehOlli/Ruokasoftaoppari

Google Play: https://play.google.com/store/apps/details?id=com.app.dinnerchatter