jamk.fi

# Mobiilisovelluksen kehittäminen Pomolle.fi-palveluun

Juuso Hatakka

Opinnäytetyö
Marraskuu 2015
Tekniikan ja liikenteen ala
Ohjelmistotekniikan koulutusohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

**Kuvailulehti**

| Tekijä(t) | Julkaisun laji | Päivämäärä |
|---|---|---|
| Hatakka, Juuso | Opinnäytetyö, AMK | 23.11.2015 |
| | Sivumäärä<br>34 | Julkaisun kieli<br>Suomi |
| | | Verkkojulkaisulupa myönnetty: x |

Työn nimi
**Mobiilisovelluksen kehittäminen Pomolle.fi-palveluun**

Tutkinto-ohjelma
Ohjelmistotekniikan koulutusohjelma

Työn ohjaaja(t)
Jouko Kotkansalo

Toimeksiantaja(t)
Silmu Software Oy

Tiivistelmä

Opinnäytetyön tavoitteena oli mobiilisovelluksen suunnittelu ja toteutus Pomolle.fi-palveluun. Pomolle.fi on Silmu Softwaren kehittämä tunti-, kulu- ja kilometriraportointijärjestelmä. Mobiilisovellus tarjoaa palvelun jatkokehityksen kannalta mahdollisuuden hyödyntää mobiililaitteiden ominaisuuksia.

Työssä käydään läpi mobiilisovellusten eri kehittämistapoja ja niiden eroja. Tarkemmin tutustutaan hybridisovellusten kehittämiseen Ionic-sovelluskehyksellä. Lisäksi työssä käsitellään mobiilisovellusta varten toteutettuun palvelinrajapintaan käytettyjä teknologioita.

Työn tuloksena mobiilisovelluksesta saatiin kehitettyä toimiva versio ja suunnitellut toiminnot toteutettua. Jatkokehitettävää sovelluksesta löytyisi ainakin paremmasta offline-tuesta ja mahdollisuudesta hyödyntää mobiililaitteiden paikannusominaisuuksia raporttien luonnissa. Sovellus jää toimeksiantajan testattavaksi ja sen jälkeen palvelun asiakkaille julkaistavaksi.

Avainsanat (asiasanat)
Hybridisovellus, Ionic, AngularJS, REST

Muut tiedot

jamk.fi

**Description**

| Author(s) | Type of publication | Date |
|---|---|---|
| Hatakka, Juuso | Bachelor's thesis | 23.11.2015 |
| | | Language of publication: Finnish |
| | Number of pages: 34 | Permission for web publication: x |

Title of publication

**Developing mobile application for Pomolle.fi service**

Degree programme

Software Engineering

Supervisor(s)

Kotkansalo, Jouko

Assigned by

Silmu Software Oy


Abstract

The objective of this thesis was to design and develop a mobile application for Pomolle.fi service. Pomolle.fi is an hour, expense and kilometer reporting system developed by Silmu Software. The mobile application will allow the use of native features of mobile devices for the service's future development.

The thesis overviews different methods for developing mobile applications and the differences between them. Hybrid application development with Ionic framework is discussed in more detail. In addition, the thesis reviews technologies used for creating a server interface used by the mobile application.

The application developed during the thesis is functional, and the planned features were implemented. For further development the application's support for offline use could be improved, and also the use of a mobile device's positioning features when creating reports could be utilized. The application will be tested by the client and after that released to customers.



Keywords/tags (subjects)

Hybrid application, Ionic, AngularJS, REST


Miscellaneous

# Sisältö

**Kuviot**

**Taulukot**

# Sanasto

**API**

Ohjelmointirajapinta (Application Programming Interface)

**HTML5**

HTML5 on HTML-merkintäkielen uusin versio, jonka W3C julkaisi virallisesti 28.10.2014. HTML5-termiä käytetään myös yleisemmin puhuttaessa asiakaspuolen web-tekniikoista (JavaScript CSS, HTML).

**HTTP**

HTTP (Hypertext Transfer Protocol) on protokolla, jota käytetään tiedonsiirtoon web-palvelinten ja selainten välillä.

**JSON**

JSON (JavasScript Object Notation) on kevyt tekstipohjainen tiedostotyyppi tiedonvälitykseen.

**MVC**

MVC (Model View Controller eli Malli Näkymä Ohjain) on arkkitehtuuri, jossa sovellus jaetaan kolmeen alueeseen. Malli huolehtii tietokannan käsittelystä ja näkymä tietojen esittämisestä. Ohjain ottaa vastaan käyttäjän komennot ja muokkaa mallia sen mukaan.

**Ruby**

Ruby on dynaamisesti tyypitetty oliopohjainen ohjelmointikieli, ja se perustuu avoimeen lähdekoodiin. Rubya käytetään eniten web-sovellusten kehittämiseen Ruby on Rails-ohjelmistokehyksen avulla.

**SaaS**

Saas (Software as a Service) on ohjelmiston jakelutapa, jossa maksut peritään tilausperusteisesti. SaaS-ohjelmistot ovat yleensä web-pohjaisia.

**SDK**

SDK (Software Development Kit) sisältää joukon työkaluja, joiden avulla voidaan kehittää sovelluksia tietylle alustalle.

**TSL/SSL**

TSL (Transport Layer Security), aiemmin nimeltään SSL (Secure Sockets Layer), on salausprotokolla. TLS:ää käytetään yleisesti web-sivujen suojaamiseen HTTPS-protokollan yli.

**UI**

Käyttöliittymä (User Interface)

# 1 Johdanto

Työn toimeksiantajana toimi Silmu Software Oy. Silmu Software on vuonna 2011 perustettu jyväskyläläinen ohjelmistoyritys, joka on keskittynyt erityisesti mobiili- ja web-sovellusten kehittämiseen. Silmu tarjoaa asiakkaille palveluita ohjelmistojen suunnittelusta ja määrittelystä aina ylläpitoon asti.

Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa mobiilisovellus Pomolle.fi-palvelun asiakkaiden käyttöön. Nykyisin palvelua käytetään mobiililaitteille suunnatulta verkkosivulta, mutta palvelun kehityksen kannalta laitteissa ajettava itsenäinen sovellus tarjoaa enemmän mahdollisuuksia. Mobiilisovellusta varten toteutettiin myös palvelinrajapinta, jonka kautta sovelluksen ja palvelun välinen tiedonsiirto hoidetaan.

Työn teoriaosassa käydään läpi mobiilisovellusten eri kehittämistapoja ja niiden eroja. Sen jälkeen tutustutaan tarkemmin HTML5-pohjaisten hybridisovellusten kehittämiseen. Lisäksi käsitellään myös palvelinrajapinnan toteuttamiseen käytettyjä teknologioita.

Työn toteutusosassa käydään läpi sovelluksen kehitysprosessi. Työn toteutus alkoi suunnittelulla ja vaatimusmäärittelyllä. Rajapinnan ja mobiilisovelluksen toteutuksen osalta käydään läpi niiden rakennetta ja teknisiä yksityiskohtia.

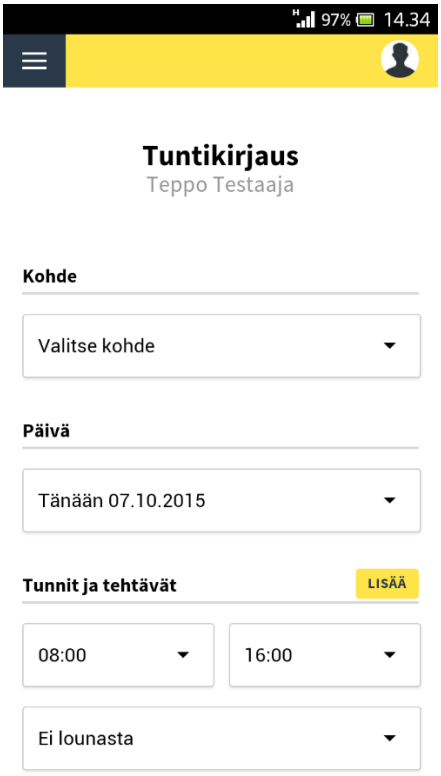
# 2 Pomolle.fi-palvelu

Pomolle.fi on alkuvuodesta 2014 julkaistu, Silmu Softwaren kehittämä, tunti-, kilometri- ja kulukirjausjärjestelmä. Palvelu toimii joustavasti eri toimialoille, mutta se on suunnattu erityisesti yrityksille, jotka toimivat useiden asiakkaiden kanssa eri aikoina, esimerkiksi rakennusliikkeet, huoltoyhtiöt ja siivousyritykset. (Pomolle 2015.)

Pomolle.fi toimii SaaS-ohjelmiston tyyppisesti. Palvelun käyttöönottoon tarvitaan vain laite nettiselaimella. Hinnoittelu perustuu palvelua käyttävien työntekijöiden määrään, ja veloitus tapahtuu kuukausittain. Palvelu jakautuu käyttöliittymän suhteen kahteen osaan: työntekijöiden raportointiosaan ja pomon hallintapaneeliin.

Raporttien syöttöpuolella työntekijät voivat luoda uusia tunti-, kilometri- ja kuluraportteja. Sivu on optimoitu mahdollisimman nopea- ja helppokäyttöiseksi mobiililait-

teille (ks. kuvio 1). Esimerkiksi tuntiraportin syötössä pakollisia tietoja on vain neljä kappaletta (kohde, päivä, tunnit ja tehtävä).




Kuvio 1. Pomolle.fi-palvelun mobiilikäyttöliittymä

Hallintapaneelissa yrityksen johto pystyy hallinnoimaan asiakastietoja, työkohteita ja tehtäviä. Työntekijöiden syöttämistä raporteista saa koostettua raportteja työntekijöittäin ja työkohteittain HTML-, PDF- ja Excel-muodossa. Asiakkaalle voidaan myös lähettää linkki, josta hän voi seurata työkohteelleen kertyneitä tunteja.

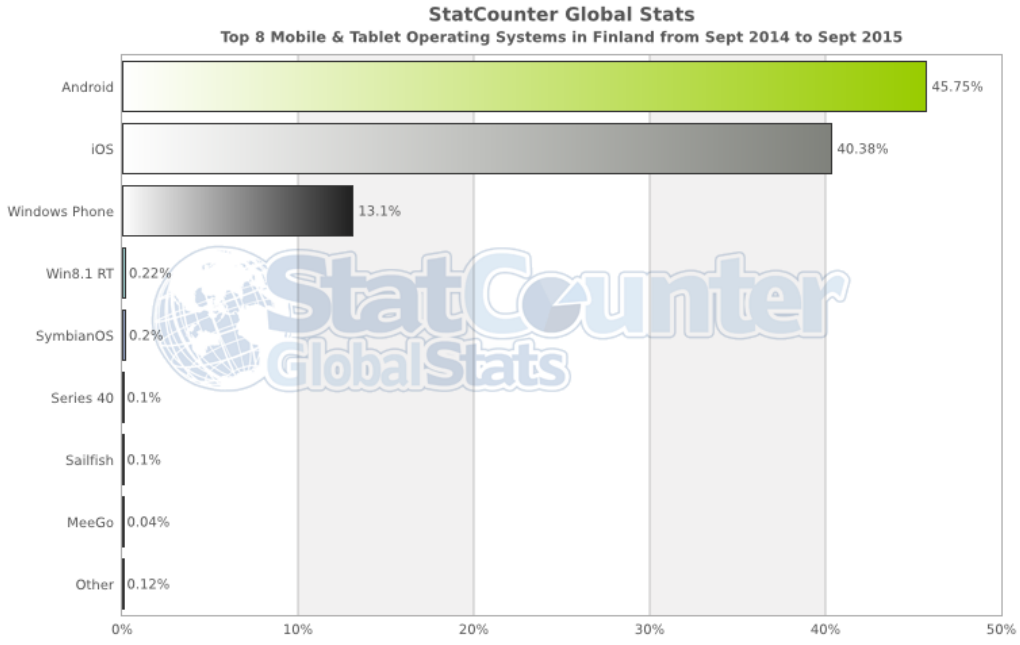
Opinnäytetyössä kehitetty mobiilisovellus keskittyy raporttien syöttöpuoleen ja se toimii nykyisen web-käyttöliittymän rinnalla. Web-sovellukseen verrattuna mobiilisovellus tarjoaa paremmat mahdollisuudet esimerkiksi virhetilanteiden käsittelyyn ja sovelluksen offline käyttöön.

# 3 Mobiilisovellukset

## 3.1 Yleistä

Nykyisen kaltaisten, kosketusnäyttöihin pohjautuvien mobiilisovellusten kehittäminen alkoi Applen julkaistua iPhone vuonna 2007. Mobiilisovelluksien jakelu tapahtuu käyttöjärjestelmäkohtaisista sovelluskaupoista. Myös sovellusten päivitykset jaetaan käyttäjille automaattisesti sovelluskaupoista. (Mobile app 2015). Ennusteiden mukaan mobiilisovellusten globaali liikevaihto tulee olemaan vajaa 40 miljardia euroa vuonna 2015. (Worldwide mobile app revenues 2015.)

Suomessa kolme suurinta mobiilikäyttöjärjestelmää ovat Googlen Android, Applen iOS ja Microsoftin Windows Phone (ks. kuvio 2). Windows Phonen osuus on globaalisti verrattuna Suomessa selkeästi suurempi. (StatCounter 2015.)




Kuvio 2. Mobiilikäyttöjärjestelmät Suomessa (StatCounter 2015)

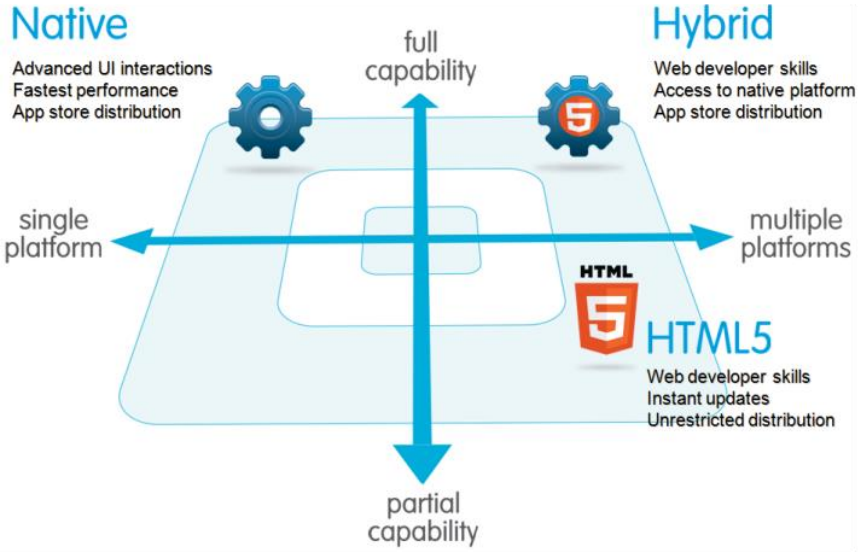
Mobiilisovellusten kehityksessä haasteita tuovat rajallinen akunkesto, prosessointiteho ja pienet näytöt. Käyttöliittymän suunnittelun pitää olla käyttäjälähtöistä ja tarvittavien toimintojen määrä tehtävien suorittamiseen mahdollisimman vähäinen. (Mobile application development 2015.)

## 3.2 Toteuttamistavat

### 3.2.1 Yleistä

Kuviossa 3 on esitetty mobiilisovellusten eri toteuttamistavat. Nykyisten mobiilikäyttöjärjestelmien alkuaikoina sovellusten toteuttamistavat olivat rajallisemmat, mutta nykyään sovellusten kehittämisessä pystytään käyttämään myös HTML5-tekniikoita.

Toteuttamistavan valinnassa tulee ottaa huomioon useita tekijöitä. Mitkä ovat sovelluksen kohdealustat? Mitä mobiililaitteiden ominaisuuksia halutaan hyödyntää? Tarvitseeko sovellus paljon suorituskykyä? Lisäksi käytössä olevat resurssit tekevät usein viimeiset rajaukset eli aika, raha ja kehittäjien osaamat teknologiat. (Native, HTML5 or Hybrid 2015.)




Kuvio 3. Natiivi-, HTML5- ja hybridisovellukset (Native, HTML5 or Hybrid 2015)

### 3.2.2 Natiivisovellukset

Natiivisovellukset ohjelmoidaan laitealustakohtaisilla teknologioilla ja menetelmillä. Androidilla sovellukset ohjelmoidaan yleensä Javalla, iOS:lla Swiftillä tai Objective-C:llä ja Windows Phonella C#:lla. Eri alustoille kehitetyt sovellukset eivät ole yhteensopivia keskenään, joten monialustaista sovellusta kehitettäessä joudutaan jokaiselle alustalle toteuttamaan oma versio.

Natiivisovellukset tarjoavat täyden tuen laitteen eri toiminnoille. Ne tarjoavat myös parhaan suorituskyvyn, josta paljon prosessointikapasiteettia käyttävät sovellukset hyötyvät, kuten esimerkiksi kuvan tai videon prosessointi. (Native, HTML5 or Hybrid 2015.)

### 3.2.3 HTML5-sovellukset

HTML5-sovellukset ovat käytännössä web-sivuja, jotka on suunniteltu toimimaan mobiililaitteiden pienille kosketusnäytöille. Käyttöliittymän osalta hyvin toteutettua HTML5-sovellusta on usein vaikea erottaa natiivisovelluksista. Sovelluksen ominaisuudet rajoittuvat laitteen selaimen mahdollistaviin toimintoihin. (Mobile: Native Apps, Web Apps, and Hybrid Apps 2015.)

### 3.2.4 Hybridisovellukset

Hybridisovellukset toteutetaan HTML5-tekniikoilla ja niitä ajetaan mobiililaitteen selainmoottorilla. Erona HTML5-sovellukseen on mahdollisuus käyttää mobiililaitteen natiiveja toimintoja (esim. kamera ja GPS-paikannus). Loppukäyttäjälle ero natiivi- ja hybridisovelluksen välillä ei usein näy merkittävästi. Natiivisovellusten tapaan hybridisovellukset voidaan jakaa käyttäjille sovelluskaupoista. (What is a Hybrid Mobile App? 2015.)

Koska hybridisovellusten pohjana on laitekohtainen selainmoottori, ei suorituskyvyssä päästä natiivisovellusten tasolle. Hybridisovellusten suurin etu on, että yhdellä koodipohjalla voidaan tukea useaa alustaa eli monialustaisen sovelluksen kehitykseen tarvittavat resurssit ovat huomattavasti pienemmät. (What is a Hybrid Mobile App? 2015.)

# 4 Hybridisovellusten kehittäminen

## 4.1 Cordova

Cordova on Apache Software Foundationin vuonna 2012 julkaisema, avoimeen lähdekoodin perustuva, mobiilisovelluskehys. Sen avulla web-tekniikoita voidaan käyttää monialustaisten mobiilisovellusten kehittämiseen. (Cordova 2015.)

Cordova tukee yhteensä kahdeksaa eri alustaa, mukaan luettuna suurimmat mobiilikäyttöjärjestelmät. Cordova tarjoaa yhteisen rajapinnan eri laitteiden natiivien ominaisuuksien käyttämiseen. Sovellukset ajetaan laitteen WebView-näkymässä. (Cordova 2015.)

Cordova ei sisällä UI-komponentteja tai MVC-kirjastoja. Cordovan päälle on kuitenkin rakennettu useita kehitystyökaluja, JavaScript-ohjelmistokehyksiä ja pilvipalveluita (Cordova 2015). Tarkempaan tarkasteluun valittiin Ionic, joka tarjosi kattavimman valikoiman työkaluja.

## 4.2 Ionic

### 4.2.1 Yleistä

Ionic on Driftyn vuonna 2013 julkaistu avoimeen lähdekoodiin perustuva SDK hybridisovellusten kehittämiseen. Se koostuu useista HTML5-, CSS- ja JavaScript-kirjastoista. Ionic tukee virallisesti Android- (minimiversio 4.1) ja iOS-käyttöjärjestelmiä (minimiversio 7). (Ionic 2015.)

Ionic käyttää animaatioissa CSS3:n siirtymäefektejä. Selainmoottorit käyttävät CSS:llä määritellyissä animaatioissa laitteen grafiikkasuoritinta, joka mahdollistaa animaatioiden sulavamman piirtonopeuden. (Ionic 2015.)

Ionicin asentamista varten kehityskoneella täytyy olla asennettuna Node.js-ympäristö. Node.js:n mukana tulevaa npm-paketinhallintaohjelmaa käytetään useasti JavaScript pohjaisten ohjelmistokirjastojen ja pakettien jakeluun. Ionic asennetaan komennolla:

```
npm install –g ionic
```

### 4.2.2 Komentorivityökalut

Ionicia käytetään pääasiallisesti sen mukana tulevan komentorivityökalun avulla. Muutamia esimerkkejä komennoista:

ionic start newProject [template]

Luo uuden Ionic projektin nimellä "newProject". Lisäparametrina voidaan antaa mallipohjan nimi, jolloin ionic generoi projektille valmiin pohjan käyttöliittymälle.

ionic serve

Käynnistää sovelluksen paikallisella palvelimella (localhost), jolloin sovellusta voidaan testata työpöytäselaimilla. Sovelluksen käyttöliittymää voidaan testata suhteellisen hyvin työpöytäselainten responsiivisuustilassa, mutta työpöytäympäristö ei tue Cordovan natiiveja liitännäisiä (esim. kamera), joten niiden testaamista varten täytyy sovellus ajaa mobiililaitteelle.

ionic run [platform]

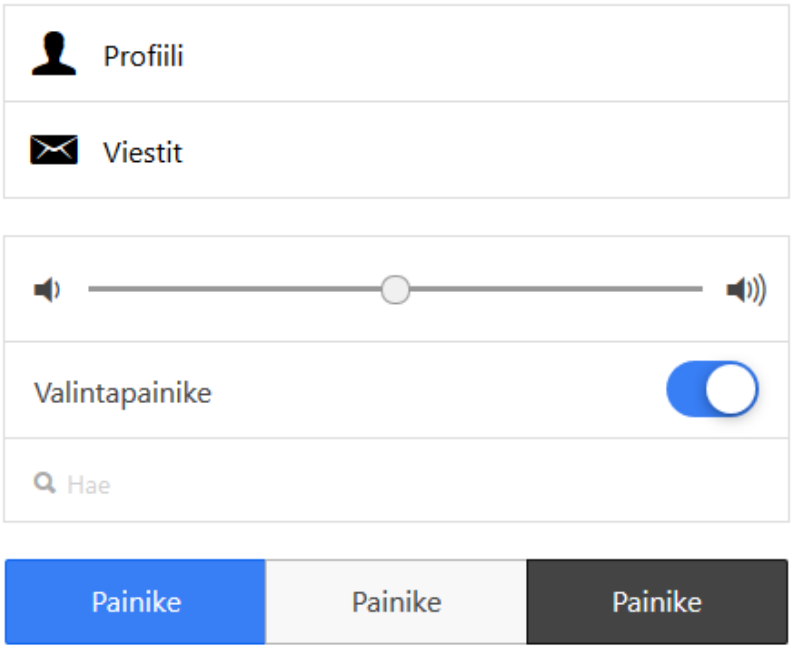
Kääntää sovelluksen määritetylle kohdealustalle. Alustan kehitystyökalut pitää asentaa erikseen ennen komennon suorittamista. Esimerkiksi Androidin tapauksessa kehityskoneella pitää olla asennettuna Android SDK.

ionic upload

Lähettää sovelluksen Ionic View-palveluun. Ionic View'n kautta sovellusta voidaan testata Android- ja iOS-laitteilla ilman, että sitä tarvitsee erikseen kääntää. Kehittäjä voi myös jakaa sovelluksessa muille palvelun käyttäjille, jolloin sovelluksen uusien versioiden jakelu testaajille helpottuu.

### 4.2.3 Käyttöliittymäkompontentit

Ionicissa on kattava määrä erilaisia mobiililaitteille suunniteltuja, CSS:llä määriteltyjä, käyttöliittymäkomponentteja. Komponentteihin sisältyy esimerkiksi painikkeita, responsiivia listoja ja lomakkeiden elementtejä (ks. kuvio 4). Niistä on myös koostettu valmiita pohjia, joiden päälle omaa sovellusta voi alkaa rakentamaan. Ionicissa on myös yli 700 vektoripohjaista ikonia.




Kuvio 4. Ionicin UI-elementtejä

## 4.3 AngularJS

### 4.3.1 Yleistä

Sovelluksen logiikan ja rakenteen luomiseen Ionicissa käytetään AngularJS:ää. AngularJS on Googlen ylläpitämä JavaScript-ohjelmistokehys, joka on suunniteltu yksisivuisten sovellusten (engl. single-page applications) kehittämiseen. (AngularJS 2015.)

AngularJS pohjautui alun perin MVC-arkkitehtuuriin, mutta nykyään se antaa kehittäjälle vapaammat kädet sovelluksen rakenteen määrittämiseen. Uuden arkkitehtuurin nimeksi onkin annettu MVW, jossa W tulee sanasta ”whatever” tarkoittaen ”mikä parhaiten toimii”. (Minar 2012.)

AngularJS:n sovelluksen rakenne jaetaan moduuleihin, joiden avulla ohjelmaa saadaan pilkottua pienempiin osiin. Yhteen moduuliin voidaan sijoittaa esimerkiksi ohjaimet ja toiseen palvelut. Sovellukselle määritetään yleensä ”päämoduuli”, jonka alle muut moduulit lisätään:

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

### 4.3.2 Ohjaimet

Angular sovellusten business-logiikka sijoitetaan ohjain-objekteihin. Kun uusi ohjain alustetaan, sille annetaan parametrina $scope-objekti. $scope-objektit ovat ohjain-kohtaisia, ja niihin määritetään funktiot ja muuttujat, joita voidaan käyttää näky-missä. Ohjaimessa määritelty funktio voidaan esimerkiksi sitoa painikkeen ng-click di-rektiiviin. (AngularJS 2015.)

Alla olevassa esimerkissä on liitetty sovelluksen moduuliin ”esimController”-niminen ohjain. $scope-objektiin on lisätty kolme muuttujaa (numero1, numero2 ja summa) ja funktio laske().

```
app.controller("esimController", function($scope) {
  $scope.numero1 = 0;
  $scope.numero2 = 0;
  $scope.summa = 0;

  $scope.laske = function() {
    $scope.summa = $scope.numero1 + $scope.numero2;
  }
});
```

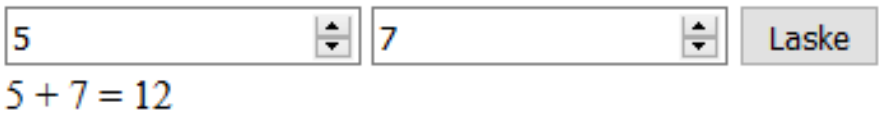
### 4.3.3 Näkymät

Angularin näkymät pohjautuvat normaaliin HTML-merkintäkieleen, johon voidaan liittää Angularin direktiivejä. Direktiivit ovat DOM-elementteihin liitettäviä määreitä, jotka Angular kääntää esim. tapahtumankuuntelijoiksi tai muokkaa sivun sisältöä tar-peen mukaan. (AngularJS 2015.)

Alla on esimerkki Angularin näkymästä, johon on liitetty ohjain ng-controller direktii-villä. Näkymään on määritetty kaksi numerokenttää, joihin on sidottu $scope muut-tujat numero1 ja numero2 ng-model direktiivillä.

```
<div ng-controller="esimController">
  <input ng-model="numero1" type="number">
  <input ng-model="numero2" type="number">
  <button ng-click="laske()">Laske</button>
  {{ numero1 }} + {{ numero2 }} = {{ summa }}
</div>
```

Kun esimerkeissä mainitut näkymä ja ohjain yhdistetään selaimessa näkyvät kuvion 5 mukaiset elementit.




Kuvio 5. Angular-näkymä

### 4.3.4 Palvelut

AngularJS:n palvelut ovat objekteja, joihin on sisällytetty funktioita. Palveluita voidaan hyödyntää eri puolilta sovellusta liittämällä ne riippuvuuksiin moduulin alustuksessa. Angulariin on sisäänrakennettu monia valmiita palveluja, esimerkiksi $http-palvelun kautta voidaan lähettää HTTP-pyyntöjä.

Palvelut voidaan alustaa kahdella tavalla. Alla olevassa esimerkissä on alustettu ”esimService”-palvelu, joka sisältää funktiot tiedon tallentamiseen ja hakemiseen selaimen localStorage-muistista.

```
app.factory("esimService", function() {
 return {
  get: function(key) {
   return JSON.parse(window.localStorage.getItem(key));
  },

  set: function(key, value) {
   window.localStorage.setItem(key, JSON.stringify(value));
  }
 }
});
```

# 5 Palvelinteknologiat

## 5.1 Yleistä

Web-palvelimet vastaanottavat HTTP-pyyntöjä toimien taustajärjestelminä asiakassovelluksille. Palvelinsovellusten kehittämiseen käytettyjä teknologioita ovat esimerkiksi PHP, Node.js ja Ruby on Rails. (Web application 2015.)

Palvelinsovellukset käyttävät tietokantoja tiedon pysyvään tallentamiseen. Käytetyimmät tietokantajärjestelmät pohjautuvat relaatiomalleihin, ja suurin osa niistä käyttää kyselykielenä SQL:ää. Tunnettuja SQL-pohjaisia tietokantaohjelmistoja ovat esimerkiksi MySQL, PostgreSQL ja Microsoft SQL Server. (Database 2015.)

Toteutettavaa mobiilisovellusta varten tarvittiin rajapinta, jonka kautta tietoa voidaan välittää nykyisen web-sovelluksen ja mobiilisovelluksen välillä. Pomolle.fi-palvelu on toteutettu Ruby on Rails-alustalla, joten se oli käytännössä järkevin vaihtoehto rajapinnan toteuttamiseksi.

## 5.2 REST

### 5.2.1 Arkkitehtuuri

REST (Representational State Transfer) on arkkitehtuurimalli dataliikenteen välittämiseen web-pohjaisissa ohjelmistoissa. REST-järjestelmät käyttävät yleensä HTTP-protokollan eri pyyntötyyppejä (esim. GET, POST, PUT, DELETE) datan siirtämiseen. (Learn REST n.d.)

REST-arkkitehtuurilla toteutetut ovat nopea ja kevyt tapa siirtää pieniä määriä dataa. Dataa käsitellään usein resursseittain, jolloin eri HTTP-pyyntötyypeillä voidaan suorittaa haluttuja toimintoja (ks. taulukko 1).

Taulukko 1. REST-rajapinnan HTTP-pyyntöjä

| **HTTP-pyyntötyyppi** | **osoite** | **tapahtuma** |
|---|---|---|
| GET | /cars | Hakee kaikki resurssit |
| POST | /cars | Luo uuden resurssin |
| PUT | /cars/:id | Päivittää resurssin annetulla id:llä |
| DELETE | /cars/:id | Poistaa resurssin annetulla id:llä |

Rajapinnat käyttävät formaattina yleensä XML- tai JSON-muotoista dataa. JSON on yleinen varsinkin JavaScript-pohjaisissa sovelluksissa, koska data voidaan muuntaa helposti objekteiksi.

### 5.2.2 Autentikointi

REST-rajapinnat täytyy suojata kuten muutkin web-sovellukset. Koska REST on arkkitehtuuriltaan tilaton, ei istuntoja voida tallentaa palvelimelle. Tämän takia asiakasohjelman täytyy lähettää jokaisella HTTP-pyynnöllä tarvittavat tiedot autentikointia varten. (The RESTful CookBook n.d.)

Yksinkertaisin tapa autentikointiin on lähettää käyttäjän tunnus ja salasana joka pyynnössä palvelimelle, jossa niiden vastaavuus tarkistetaan. Tietoturvan puolesta tämä ei ole paras tapa, koska jo yhden HTTP-pyynnön kaappaamisella saadaan haltuun käyttäjän kirjautumistiedot. (Secure Your REST API 2015.)

REST-rajapinnoissa autentikointiin yleisesti käytetty vaihtoehto on tokeneihin pohjautuva järjestelmä. Niissä käyttäjän tunnusta ja salasanaa vastaan annetaan token, jonka avulla myöhemmät pyynnöt suoritetaan. Tokenit ovat käytännössä pitkiä merkkijonoja joihin on sisällytetty käyttäjän tunnistamiseen tarvittavat tiedot ja ne on salattu vain palvelimen tiedossa olevalla avaimella. (Token Based Authentication 2004.)

SSL:n käyttö kaikissa autentikointia vaativissa toiminnoissa on tärkeää, koska muuten käyttäjän kirjautumistiedot kulkevat selkokielisenä. (Secure Your REST API 2015.)

## 5.3 Ruby on Rails

### 5.3.1 Esittely

Ruby on Rails (lyh. Rails) on Ruby-ohjelmointikielellä rakennettu ohjelmistokehys, jonka lähdekoodi on avointa. Rails on suunniteltu web-sovellusten kehittämiseen ja se perustuu MVC-arkkitehtuuriin. Suurimpia Ruby on Rails-alustan käyttäjiä ovat yhteisöpalvelu Twitter ja versionhallintapalvelu GitHub. (Ruby on Rails n.d.)

Tietokannan käsittelyyn Rails sovelluksissa käytetään Active Record rajapintaa. Active Record tukee useimpia SQL-pohjaisia tietokantoja. Tietokantaa käsitellään sovellukseen luotujen mallien kautta, jotka vastaavat tietokannan tauluja. (Active Record Basics n.d.)

### 5.3.2 Pakettienhallinta

Ruby-ohjelmointikielessä ohjelmistopaketeista ja kirjastoista käytetään nimitystä gem. Rails projekteissa gemien hallintaan käytetään Bundler-työkalua, jossa halutut paketit määritetään Gemfile-tiedostoon. Tiedostoon määritetyt gemit voidaan asentaa ajamalla komento bundle install. (Bundler n.d.)

REST API:n toteutusta varten valittiin Grape gemi. Grape on kevyt ohjelmistokehys REST API:en luomiseen, ja se on helppo liittää jo olemassa olevaan Rails-sovellukseen. Grape tukee teksti-, XML-, JSON-, binäärisisältötyyppejä. (Grape 2015.)

Grapessa rajapinnan koodi jaetaan resursseittain moduuleihin, joiden sisälle voidaan määrittää eri HTTP-pyyntötyyppejä vastaavat toiminnot. Grape tukee myös rajapinnan versiointia, joka on hyvä ominaisuus taaksepäin yhteensopivuuden kannalta, jos siihen joudutaan tekemään suurempia muutoksia. (Grape 2015.)

Alla olevassa esimerkissä on esitetty Grapella toteutettu resurssiluokka. Luokalle on toteutettu GET-, POST- ja DELETE-pyyntötyyppejä vastaavat metodit.

```
module API
 class Cars < Grape::API
  version 'v1'
  format :json

  resource :cars do
   desc 'Return all cars.'
   get do
    cars = Car.all
    present vehicles, with: API::Entities::Vehicle
   end

   desc 'Create a car.'
   params do
    requires :make, type: String
    requires :model, type: String
    requires :year, type: Integer
   end
   post do
    Car.create!({make: params[:make], model: params[:model], params[:year]})
   end

   desc 'Delete a car.'
   params do
    requires :id, type: Integer
   end
```

```
    route_param :id do
     delete do
       Vehicle.find_by_id(params[:id]).destroy
     end
    end

  end
 end
end
```

# 6 Työn toteututus

Työn toteutus aloitettiin vaatimusmäärittelyllä ja käyttöliittymän suunnittelulla. Kun sovelluksen ominaisuuksista ja käyttöliittymästä oli tarpeeksi hyvä kuva, voitiin varsinainen kehitystyö aloittaa. Kehitys aloitettiin rajapinnan toteuttamisella, joka mahdollisti heti oikean datan käytön, kun mobiilisovellusta alettiin toteuttaa.

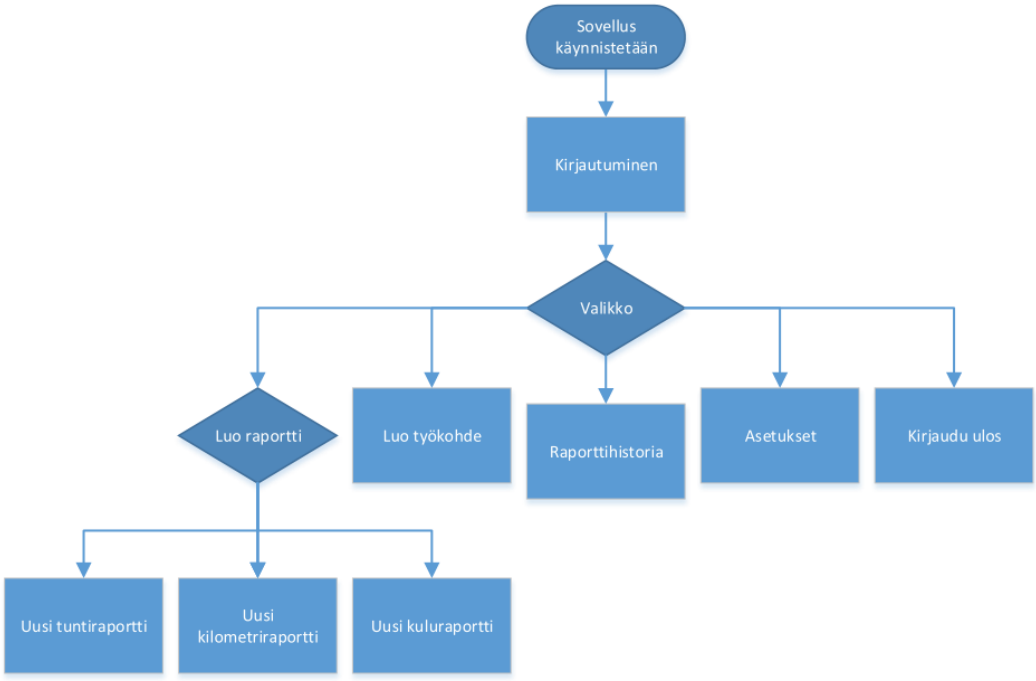
## 6.1 Vaatimusmäärittely

Sovellukseen kehitettävien ominaisuuksien määrittelyssä lähdettiin liikkeelle nykyisestä web-sovelluksesta. Suurimmat uudistukset liittyivät työntekijäkohtaisen kirjautumisen toteuttamiseen, joka mahdollisti esimerkiksi työntekijätasojen lisäämisen.

- Tunti-, kulu- ja kilometriraporttien luonti
- Työntekijöille mahdollisuus selata omia raportteja
- Työntekijöille mahdollisuus raporttien muokkaamiseen
- Työntekijöille mahdollisuus lisätä työkohteita
- Työntekijätasot

## 6.2 Suunnittelu

Kuviossa 6 on esitetty sovelluksen suunniteltua rakennetta. Kun sovellus käynnistetään ensimmäisen kerran, käyttäjältä kysytään kirjautumistietoja. Sovelluksen rakenne haluttiin pitää mahdollisimman yksinkertaisena, joten toteutettavia näkymiä ei kertynyt kovinkaan paljon.




Kuvio 6. Sovelluksen suunniteltu rakenne

Mobiilisovelluksen käyttöliittymästä tehtiin havainnekuvia FluidUI-työkalulla. FluidUI on tarkoitettu erityisesti mobiilisovellusten käyttöliittymien suunnitteluun. Työkalulla pystyy myös simuloimaan näkymissä liikkumista lisäämällä elementeillä tapahtuman-kuuntelijoita.

Kuviossa 7 on esitetty kaksi FluidUI:lla piirrettyä käyttöliittymäkuvaa. Vaikka kuvien tyylistä tuleekin aika ”karkeita”, oli niistä kuitenkin näkymiä ohjelmoitaessa hyötyä, kun mallia ei tarvinnut vain pyöritellä päässä. Kuvia voidaan verrata kuvioissa 9 ja 14 esitettyihin valmiisiin näkymiin.


Kuvio 7. FluidUI:lla piirrettyjä UI-kuvia

## 6.3 Rajapinnan toteutus

REST rajapinta toteutettiin nykyisen Ruby on Rails -sovelluksen osaksi. Rajapinnan toteutus oli melko suoraviivaista, koska se voitiin toteuttaa valmiiden tietokantamallien päälle. Työksi jäi siis lähinnä tiedon muuntaminen oikeaan formaattiin ja rajapinnalle sopivan autentikaatiojärjestelmän rakennus.

Autentikaatiota varten toteutettiin koko API:lle yhteinen Helper-metodi, jota voidaan kutsua eri resurssien sisältä:

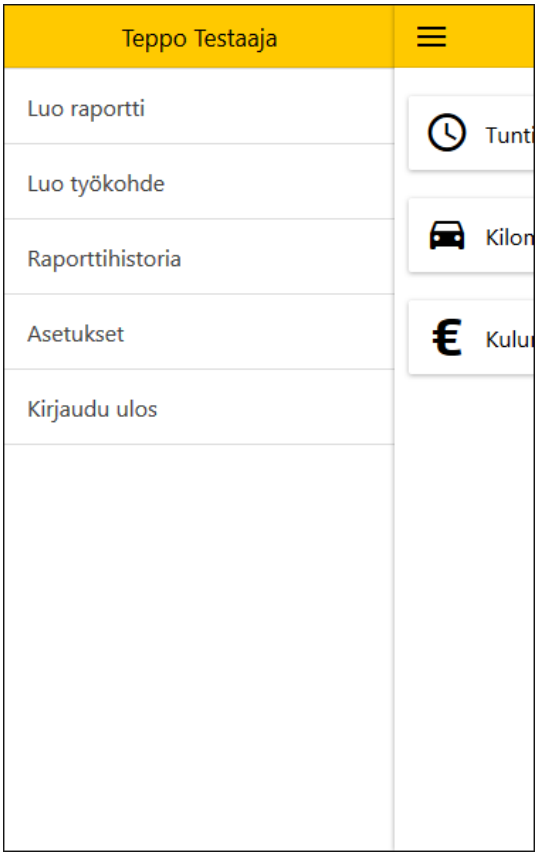
```
def authenticate
  token = headers['Authorization']
  begin
    decoded = jwt_decode(token)
    @current_user = User.find_by_id(decoded["user_id"])
  rescue
    # Decoding token failed
    error!('Unauthorized. Invalid or expired token.', 401)
  end
end
```

Yhteensä toteutettavia resurssiluokkia kertyi 11 sisältäen eri raporttityypit, niihin liitettävät kuvat ja liitteet, yrityksen työkohteet ja tehtävät.

## 6.4 Mobiilisovelluksen toteutus

### 6.4.1 Valikkorakenne

Sovelluksessa navigointi tapahtuu, mobiilisovelluksissa melko yleisesti käytetyn, sivuvalikkorakenteen kautta (ks. kuvio 8). Valikko voidaan avata joko vasemman yläreunan painikkeella tai pyyhkäisyeleellä vasemmalta oikealle. Valikon yläreunassa lukee sisäänkirjautuneen käyttäjän nimi. Luo työkohde –painike on näkyvissä vain käyttäjille, joille ominaisuus on aktivoitu pomon hallintapaneelista.




Kuvio 8. Sivuvalikko

Ionicissa sivuvalikon määrittämiseen on valmiit HTML-elementit. Valikko voidaan määrittää aukeamaan joko vasemmalta tai oikealta.

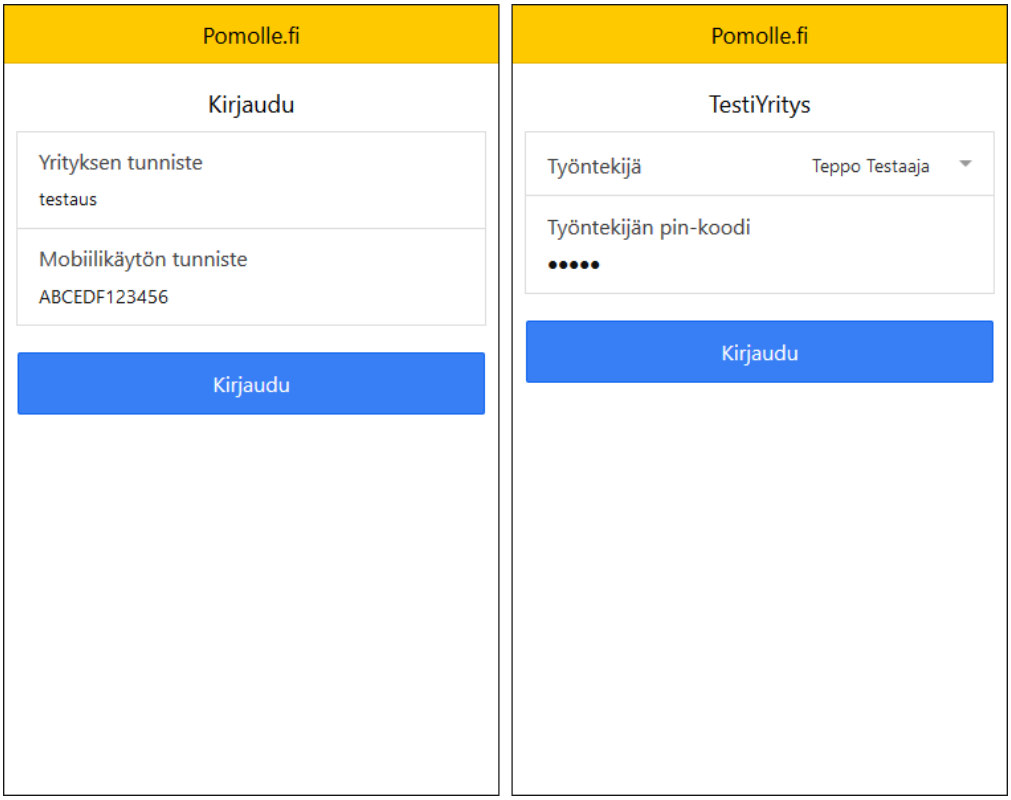
```
<ion-side-menus>
  <ion-side-menu side="left">
    <!-- Vasemman sivuvalikon sisältö -->
  </ion-side-menu>

  <ion-side-menu-content>
   <!-- Näkymän sisältö -->
  </ion-side-menu-content>
</ion-side-menus>
```

### 6.4.2 Kirjautuminen

Kun sovellus käynnistetään ensimmäistä kertaa, käyttäjä ohjataan kirjautumisnäkymään (ks. kuvio 9). Sovellukseen kirjautuminen on toteutettu kaksivaiheisesti: ensin tunnistetaan yritys ja sen jälkeen työntekijä.

Yrityskirjautumisessa käyttäjä syöttää yrityksen tunnistetiedot. Sen jälkeen käyttäjälle näytetään yrityksen työntekijälista, josta hän valitsee itsensä ja syöttää henkilökohtaisen pin-koodinsa.

Onnistuneen kirjautumisen jälkeen käyttäjälle palautetaan palvelimella generoitu token, jolla käyttäjä autentikoidaan REST-rajapinnan välisessä liikenteessä. Sen jälkeen käyttäjä ohjataan sovelluksen etusivulle.

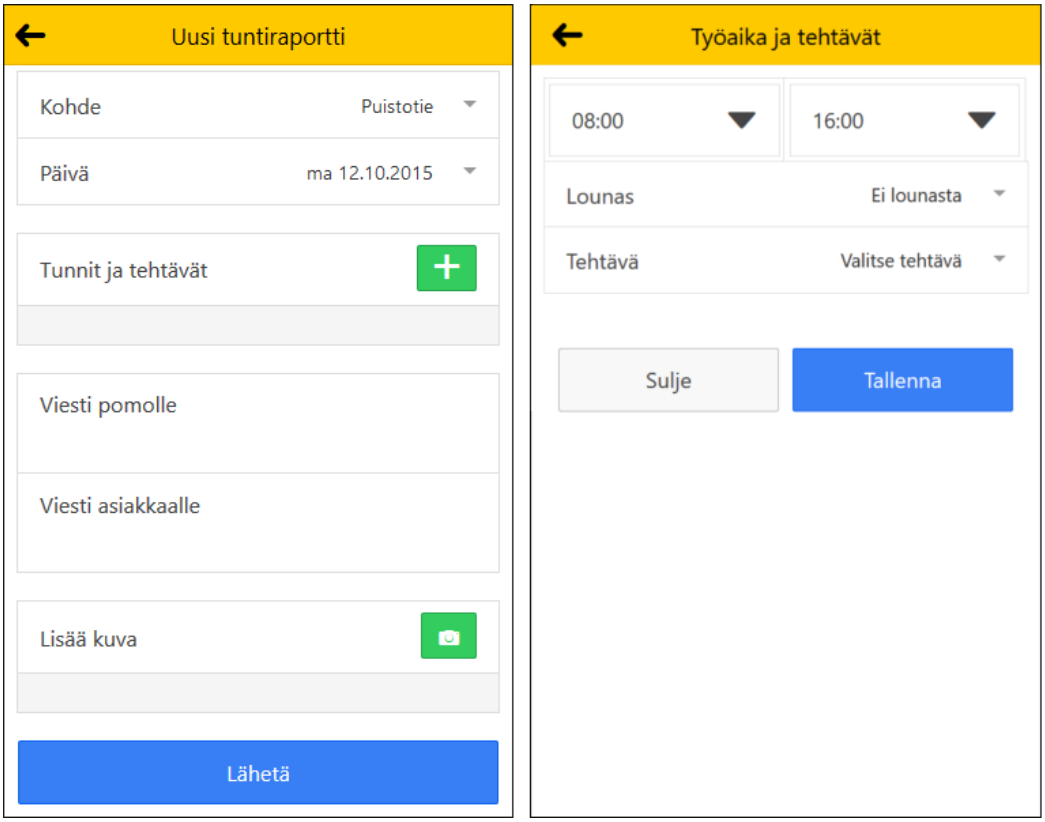



Kuvio 9. Yritys- ja työntekijäkirjautumisen näkymät

### 6.4.3 Raporttien luonti

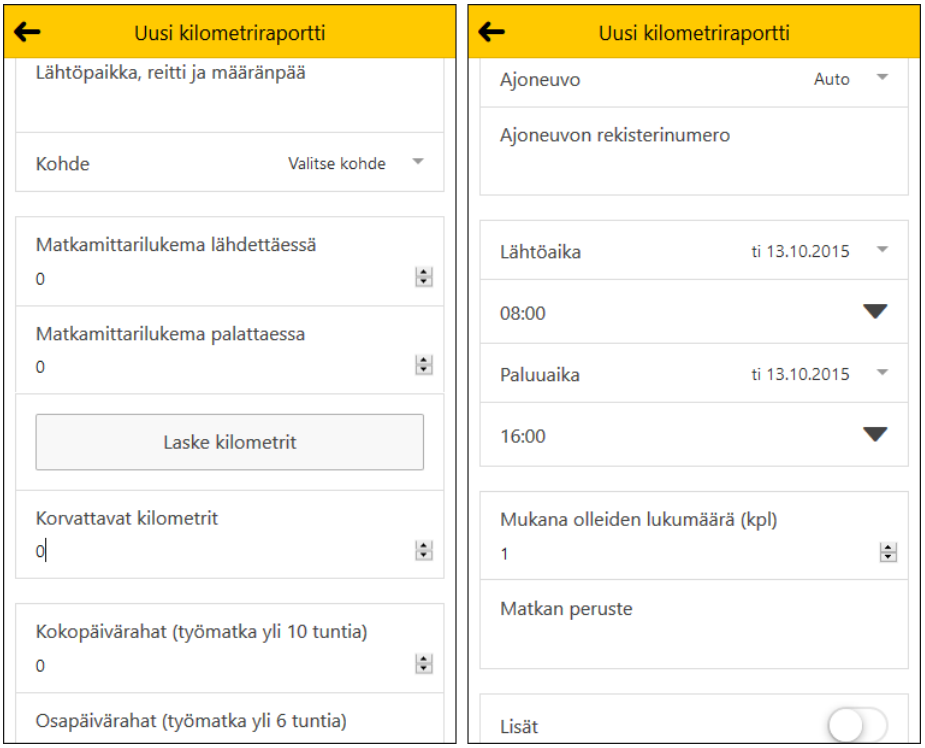
Raporttien luonti sivujen rakenteessa lähdettiin liikkeelle nykyisen web-sovelluksen lomakkeiden pohjalta. Raporteissa lähetettävä data pysyi samanlaisena, joten rakenteellisesti suuria muutoksia ei tarvittu.

Tuntiraportin luonti aloitetaan työkohteen ja päivämäärän valinnalla (ks. kuvio 10 vasemmalla). Sen jälkeen lisätään tunnit ja tehtävät erikseen avautuvan ikkunan kautta (ks. kuvio 10 oikealla). Vapaavalintaisina kenttinä ovat viesti pomolle ja asiakkaalle kentät. Viesti asiakkaalle-kenttään voi merkitä tarkemmat tiedot tehdyistä työtehtävistä.



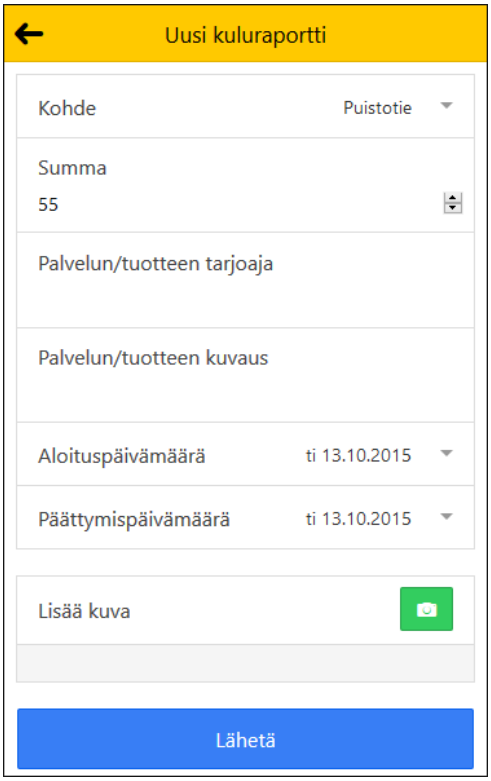

Kuvio 10. Uuden tuntiraportin näkymä

Kilometriraportteihin syötetään eniten tietoa, joten sen lomake oli työläin tehdä. Raportteihin syötetään reitti, työkohde, ajoneuvo, lähtö- ja paluuajat ja matkamittarilukemat (ks. kuvio 11). Korvattavat kilometrit voidaan joko laskea matkamittarilukemista tai syöttää erikseen. Lisäksi raportissa voidaan ottaa huomioon erilaisia lisiä tai päivärahat.




Kuvio 11. Uuden kilometriraportin näkymä

Kuluraportteihin syötetään työkohde, summa, palvelun tarjoaja ja kuvaus sekä päivämäärät (ks. kuvio 12).



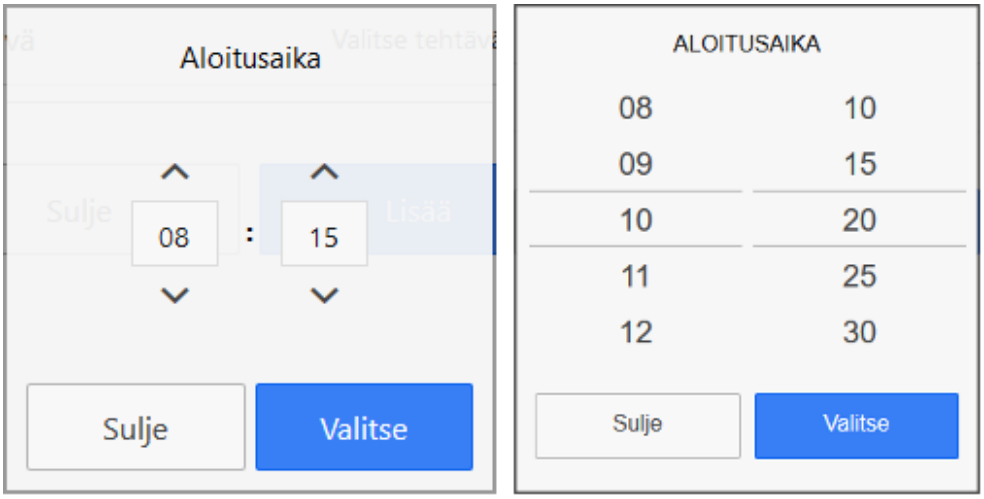

Kuvio 12. Uuden kuluraportin näkymä

Raportteihin pystyy liittämään kuvia esimerkiksi työkohteelta tai kulukuiteista. Koska kuvien käsittely poikkeaa muun datan käsittelystä niitä varten tehtiin oma palvelu Angulariin. Tiedostojen lähettämiseen käytetään Cordovan file-transfer liitännäistä.

```
angular.module('pomolle.services')

// Service for sending image attachments to reports.
.factory('AttachmentsService', function($http, $q, API_URL, UserService) {
 return {
  create: function(url, image, fileName, fileKey, params) {
   var defer = $q.defer();

   var options = new FileUploadOptions();
   options.fileKey = fileKey;
   options.fileName = fileName;
   options.headers = { 'Authorization': UserService.getToken() }
   options.params = params;

   var ft = new FileTransfer();
   ft.upload(image, encodeURI(API_URL + url), uploadSuccess, uploadError, options);

   function uploadSuccess(response) {
    defer.resolve(response);
   }

   function uploadError(response) {
    defer.reject(response.data);
   }

   return defer.promise;
  }
 }
});
```

Raporttien ajan syötöstä haluttiin mahdollisimman nopea ja helppokäyttöinen kosketusnäytöillä. Ensimmäisenä kokeiltavana vaihtoehtona oli puhtaasti Ionicille suunniteltu ionic-timepicker komponentti (ks. kuvio 13 vasemmalla). Se todettiin kuitenkin melko hitaaksi ja kömpelöksi käyttää kun arvojen muuttaminen piti tehdä yksitellen nuolia klikkaamalla.

Lopullisessa valinnassa päädyttiin MobiScroll-kirjastoon. MobiScroll tarjoaa useita kosketusnäytöille suunniteltuja komponentteja. Toteutettu ajanvalitsin toimii rullamaisesti kosketuseleiden avulla (ks. kuvio 13 oikealla).



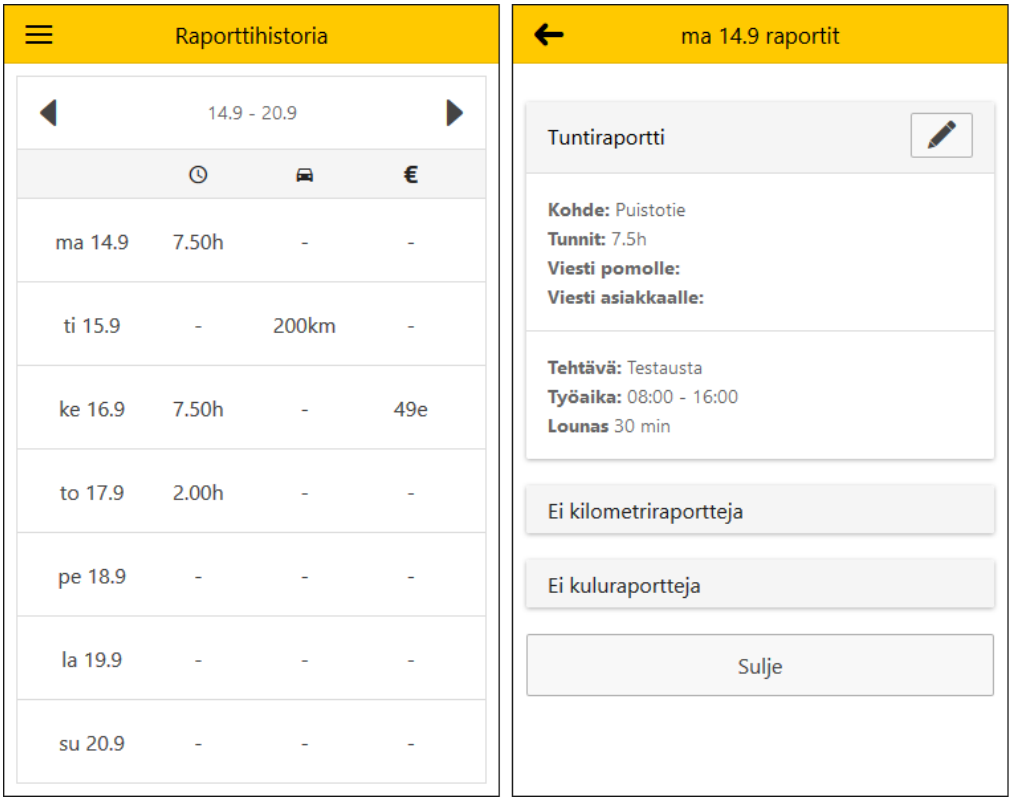

Kuvio 13. Vanha ja uusi ajanvalitsin

### 6.4.4 Raporttihistoria

Raporttihistoriasta työntekijä pystyy selaamaan lähettämiään raportteja kymmenen viikon ajalta. Raportit näytetään viikoittain ja listattuna on jokaisen päivän kokonaistunnit, -kilometrit ja -kulut, jotka on erotettu kello, auto ja euro-symboleilla (ks. kuvio 14 vasemmalla).

Yksittäistä päivää klikkaamalla aukeaa näkymä josta näkee raporttien tarkemmat tiedot (ks. kuvio 14 oikealla). Työntekijällä on myös mahdollisuus muokata raportteja 24 tunnin ajan. Muokkaus mahdollisuus on osoitettu kyseisen raportin kohdalla kynä ikonilla.

Raporttien muokkaussivut perustuvat samoihin pohjiin kuin niiden luontikin; erona vain raportin tiedot haetaan palvelimelta.

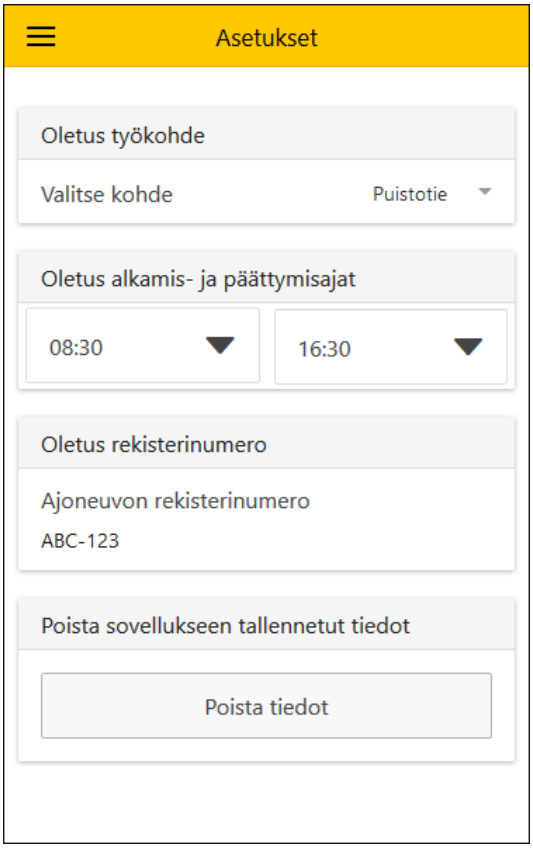



Kuvio 14. Raporttihistorian näkymät

### 6.4.5 Asetukset

Asetukset-sivulle on kerätty lähinnä sovelluksen personointiin liittyviä ominaisuuksia (ks. kuvio 15). Käyttäjä voi valita itselleen oletus työkohteen ja työajat jotka valitaan automaattisesti uutta raporttia luodessa. Tämä nopeuttaa raporttien syöttöä varsinkin jos työntekijä työskentelee samassa työkohteessa useamman päivän ajan.

Asetusten kautta voidaan myös poistaa sovellukseen tallennetut tiedot. Jos käyttäjän pitää esimerkiksi vaihtaa yritystä jolle hän on kirjautunut, täytyy vanhat tiedot poistaa tätä kautta. Tiedot saatetaan myös joutua poistamaan jonkin harvinaisemman vikatilanteen sattuessa.




Kuvio 15. Asetukset-näkymä

# 7 Testaus

Ohjelmistotestauksella pyritään suunnitelmallisesti havaitsemaan sovelluksessa mahdollisesti ilmenevät virheet. Ohjelman täydellistä virheettömyyttä niilläkään ei kuitenkaan pystytä osoittamaan (Haikala & Mikkonen 2011, 205). Ajan tasalla pidetyt automaattiset testit helpottavat sovelluksen ylläpitoa varsinkin uuden kehittäjän tullessa mukaan projektiin.

## 7.1 Rajapinnan testaus

Ruby on Rails sovellusten testaaminen on pyritty tekemään mahdollisimman vaivattomaksi. Yleensä testausta aloitettaessa eniten aikaa vievä osa on testiympäristön pystyttäminen. Rails projekteissa tämä on otettu huomioon määrittämällä sovellukselle oletuksena kolme eri ympäristöä: kehitys, testaus ja tuotanto. Jokaiselle ympäristölle on määritetty oma tietokanta. (A Guide to Testing Rails Applications n.d.)

Kehitetylle REST-rajapinnalle tehtiin yksikkötestit, jotka testaavat rajapinnan funktioita eri syötteillä. Testit kirjoitettiin Minitest-kirjastolla, jota käytetään oletus testauskirjastona Rails projekteissa.

Minitestillä tehdään HTTP-pyyntö ja sen palauttamaa vastausta verrataan haluttuun tulokseen. Alla olevassa testissä haetaan tuntiraportteja virheellisellä tokenilla, jolloin palvelimen tulisi vasta HTTP statuskoodilla 401 Unauthorized.

```
test 'GET work reports with invalid token' do
  get '/api/v1/work_reports', nil, {'HTTP_Authorization' => 'ABCDEF'}
  assert_equal last_response.status, 401
end
```

### 7.2 Mobiilisovelluksen testaus

Mobiilisovellusta varten tehtiin e2e (end-to-end) testit Protractor-kirjastolla. Protractorilla kirjoitetuissa testeissä määritetään komennot, jotka ajetaan suoraan selaimessa.

Alla olevassa testissä yrityskirjautumista testataan väärillä syötteillä. Epäonnistuneen kirjautumisen jälkeen käyttäjälle tulisi näkyä virheilmoitus, jonka näkyvyys testissä tarkastetaan.

```
it('should not login company with invalid credentials', function() {
   // Invalid request should cause error popup
   element(by.model('company.identification')).sendKeys('foo');
   element(by.model('company.hash_identifier')).sendKeys('bar');
   element(by.css('.button')).click();
   browser.wait(EC.presenceOf(element(by.css('.popup-title'))), 3000);
   expect(element(by.css('.popup-title')).getText()).toContain('Virhe!')
});
```

## 8 Tulokset ja johtopäätökset

Projekti eteni ilman suurempia ongelmia. Sovelluksen vaatimusmäärittely saatiin tehtyä heti alkuvaiheessa riittävän tarkasti, joten toteutusvaiheessa ei ilmennyt enää isoja yllätyksiä. Työtä oli mielenkiintoista tehdä ja siinä pääsi käyttämään monipuolisesti sekä palvelin-, että asiakaspuolen teknologioita.

Hybridisovellusten kehittämiseen tutustuminen oli mielenkiintoista. Olen aiemmin ohjelmoinut natiivisovellusta Androidin kehitystyökaluilla. Siihen verrattuna hybridisovelluksen toteuttaminen kehittäjän näkökulmasta oli monilla osa-alueilla paljon mieluisampaa. Samalla sain tutustua myös AngularJS:ään johon pääsin nopeasti kiinni.

Työn tuloksena mobiilisovelluksesta saatiin toimiva versio ja suunnitellut toiminnot toteutettua. Sovelluksen avulla pystytään lähettämään kaikkia raporttityyppejä, luomaan työkohteita ja selaamaan ja muokkaamaan lähetettyjä raportteja.

Sovelluksessa parannettavaa löytyisi ainakin paremmasta offline-tuesta. Nyt käyttäjälle vain ilmoitetaan virheestä kun raportin lähetys epäonnistuu internet-yhteyden takia ja hän voi yrittää lähetystä uudelleen. Raportit voitaisiin tallentaa ensin laitteen muistiin ja lähettää kun internet-yhteys on taas saatavilla. Lisäksi sovelluksessa voitaisiin hyödyntää enemmän muitakin mobiililaitteiden ominaisuuksia esim. GPS-paikannusta työkohteen haussa.

Toimeksiantaja oli tyytyväinen saavutettuun tulokseen. Sovellus jää toimeksiantajan testattavaksi ja sen jälkeen asiakkaille julkaistavaksi.

# Lähteet


Active Record Basics. N.d. RailsGuides-opas. Viitattu 7.10.2015. http://guides.rubyonrails.org/active_record_basics.html

A Guide to Testing Rails Applications. N.d. RailsGuides-opas. Viitattu 30.10.2015. http://guides.rubyonrails.org/testing.html

AngularJS. 2015. AngularJS kotisivut. Viitattu 15.10.2015. https://angularjs.org/

Bundler. N.d. Bundler kotisivut. Viitattu 8.10.2015. http://bundler.io/

Cordova. 2015. Apache Cordovan kotisivut. Viitattu 15.10.2015. https://cordova.apache.org/

Database. 2015. Wikipedia artikkeli. Viitattu 30.10.2015. https://en.wikipedia.org/wiki/Database

Grape. 2015. GitHub projekti. Viitattu 8.10.2015. https://github.com/ruby-grape/grape

Haikala, I. & Mikkonen, M. 2011. Ohjelmistotuotannon käytännöt. Helsinki: Talentum Media Oy.

Ionic. 2015. Wikipedia artikkeli. Viitattu 6.10.2015. https://en.wikipedia.org/wiki/Ionic_%28mobile_app_framework%29

Learn REST. N.d. Blogger artikkeli. Viitattu 14.10.2015. http://rest.elkstein.org/

Minar. I, 2012. Google+ viesti. Viitattu 16.10.2015. https://plus.google.com/+AngularJS/posts/aZNVhj355G2

Mobile app. 2015. Wikipedia artikkeli. Viitattu 8.10.2015. https://en.wikipedia.org/wiki/Mobile_app

Mobile application development. 2015. Wikipedia artikkeli. Viitattu 21.10.2015. https://en.wikipedia.org/wiki/Mobile_application_development

Mobile: Native Apps, Web Apps, and Hybrid Apps. 2013. Nielsen Norman Groupin artikkeli. Viitattu 10.10.2015. http://www.nngroup.com/articles/mobile-native-apps/

Native, HTML5 or Hybrid. 2015. Salesforce developers. Viitattu 11.10.2015 https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options

Pomolle. 2015. Pomolle.fi-palvelun kotisivut. Viitattu 6.10.2015. https://pomolle.fi/

Ruby on Rails. N.d. Ruby on Rails kotisivut. Viitattu 7.10.2015. http://rubyonrails.org/

Secure Your REST API. 2015. Stormpath blogikirjoitus. Viitattu 17.10.2015. https://stormpath.com/blog/secure-your-rest-api-right-way/

StatCounter. 2015. StatCounter-tilastopalvelu. Viitattu 6.10.2015. http://gs.statcounter.com/

The RESTful CookBook. N.d. The RESTful CookBook-opas. Viitattu 16.10.2015. http://restcookbook.com/Basics/loggingin/

Token Based Authentication. 2004. W3C-yhteisön artikkeli. Viitattu 17.10.2015. http://www.w3.org/2001/sw/Europe/events/foaf-galway/papers/fp/token_based_authentication/

Web application. 2015. Wikipedia artikkeli. Viitattu 14.10.2015. https://en.wikipedia.org/wiki/Web_application

What is a Hybrid Mobile App?. 2015. Telerik Developer Network. Viitattu 6.10.2015. http://developer.telerik.com/featured/what-is-a-hybrid-mobile-app/

Worldwide mobile app revenues. 2015. Statista-tilastopalvelu. Viitattu 21.10.2015. http://www.statista.com/statistics/269025/worldwide-mobile-app-revenue-forecast/