jamk.fi

WEB-TEKNIIKAT JA MOBIILIKEHITYS

Taneli Hartikainen

Opinnäytetyö
12 / 2013

Ohjelmistotekniikan koulutusohjelma
Tekniikan ja liikenteen ala

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

KUVAILULEHTI

Tekijä(t) HARTIKAINEN, Taneli	Julkaisun laji Opinnäytetyö	Päivämäärä 6.12.2013
	Sivumäärä 45	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty (X)
Työn nimi WEB-TEKNIIKAT JA MOBIILIKEHITYS		
Koulutusohjelma Ohjelmistotekniikan koulutusohjelma		
Työn ohjaaja(t) PIETIKÄINEN, Kalevi		
Toimeksiantaja(t) JAMK, ”Metsävaramittaus nuorissa kunnostamattomissa metsissä”-projekti		

Tiivistelmä

Opinnäytetyössä toteutettiin Jyväskylän Ammattikorkeakoulun Metsävaramittaus-projektille mittaustyössä käytettävä mobiilisovellus. Sovelluksella tulisi lukea dataa Masser BT Caliper mittalaitteelta ja esittää data mittaajalle järkevässä muodossa. Tavoitteena oli käydä läpi web-kehityksessä yleisiä tekniikoita ja hyödyntää niitä mobiilialustoille kehitettävässä sovelluksessa.

Mobiilisovellus toteutettiin HTML5-tekniikoilla alustavasti Android-käyttöjärjestelmän mobiililaitteille PhoneGap-ohjelmistokehystä hyödyntäen. Sovelluksen käyttöliittymä määriteltiin HTML-merkintäkielellä käyttäen ”jQuery Mobile”-kirjaston valmiita CSS-komponentteja. Sovellus keskustelee opinnäytetyön ulkopuolelle jääneen palvelimen kanssa REST-rajapinnan välityksellä. Kehityksen aikana toteutettiin PhoneGap-ohjelmistokehykselle Bluetooth-rajapinta, joka julkaistiin myöhemmin myös GitHub-palvelussa julkiseen levitykseen.

Sovellusta on esitelty eri metsäalan toimijoille ja sitä jatkokehitetään edelleen. Sovellus on testattu täysin toimivaksi Androidin lisäksi myös Windows Phone 8 alustalla.

Avainsanat (asiasanat)
Android, Bluetooth, PhoneGap, Cross-platform, HTML5, JavaScript, CoffeeScript, jQuery Mobile, Backbone

Muut tiedot

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Author(s)	Type of publication	Date
HARTIKAINEN, Taneli	Bachelor´s Thesis	6.12.2013
	Pages 45	Language Finnish
		Permission for web publication (X)

Title
WEB-TECHNIQUES AND MOBILE DEVELOPMENT

Degree Programme
Software Engineering

Tutor(s)
PIETIKÄINEN, Kalevi

Assigned by
JAMK, ”Forest resource measurement in untreated forests”-project

Abstract

The thesis focused on creating a mobile application for the Forest Resource Measurement project in Jyväskylä University of Applied Sciences. The mobile application would be used to read and display data from Masser BT Caliper device. The data transfer would occur via Bluetooth.

The application would first be developed for Android using the PhoneGap cross-platform framework. This would enable future development for additional platforms with relative ease. A Bluetooth-plugin was developed for the PhoneGap-framework, which was later released to the public via the GitHub-service.

The software was written largely in CoffeeScript, with HTML defining the user interface layout. JQuery Mobile was used as a CSS-framework to simplify development regarding the user interface. The application communicates with a server through a REST-interface, however the implementation of the server-side application remained outside the scope of this thesis.

The application is constantly being developed further and it has been presented to various people working in the forest-industry. Currently the application works in both Android and Windows Phone 8 sharing all the web-application code between the platforms.

Keywords
Android, Bluetooth, PhoneGap, Cross-platform, HTML5, JavaScript, CoffeeScript, jQuery Mobile, Backbone

Miscellaneous

Sisältö

Kuviot

SANASTO

HTML5

HTML5 tarkoittaa virallisesti (kirjoituksen hetkellä) uusinta versiota HTML-merkintäkielestä (HyperText Markup Language), jota käytetään verkkosivujen rakentamiseen. Nykyään termiä käytetään yleisesti kuvaamaan moderneja web-tekniikoita ja niin tehdään myös tässä opinnäytetyössä.

JavaScript

JavaScript on web-kehityksessä yleisimmin käytetty ohjelmointikieli. JavaScript on syntaksiltaan C-kielen kaltainen, mutta se sisältää kuitenkin monia nykyaikaisen dynaamisesti tulkittavan ohjelmointikielen ominaisuuksia.

CoffeeScript

CoffeeScript on JavaScriptiksi kääntyvä ohjelmointikieli, joka kätkee monet JavaScriptin puutteet vaihtoehtoisen syntaksin taakse. CoffeeScript on pohjimmiltaan JavaScriptiä, mutta lievän erilaisuuden vuoksi sitä tulisi kirjoittaa kuin CoffeeScriptiä.

REST

REST eli Representational State Transfer käsitettä käytetään usein kuvaamaan web-kehityksessä käytettävää tiedonsiirto-arkkitehtuuria. REST-arkkitehtuurin ominaisuuksiin kuuluvat esimerkiksi asiakasohjelman ja palvelimen erotus, palvelimelle lähetettyjen käskyjen käsittely riippumattomina (Stateless protocol) ja vastausten helppo tallennus asiakasohjelman puolella offline-käyttöä varten.

Responsiivisuus

Käyttöliittymäkehityksen puolella responsiivisuudella tarkoitetaan ulkoasun kykyä muuntautua eri laitteille sopivaksi. Isolla ruudulla käyttöliittymä pyrkii käyttämään saatavilla olevan tilan mahdollisimman tehokkaasti, kun taas pienellä ruudulla käyttöliittymästä voidaan karsia joitain ominaisuuksia ja optimoida sitä esimerkiksi mobiililaitteelle.

SDK

SDK eli Software Development Kit on yleensä kokoelma työkaluja, jotka mahdollistavat sovellusten kehittämisen tietylle alustalle. SDK sisältää usein kehittäjää helpottavia työkaluja kuten debug-monitoreita, esimerkkikoodia ja joskus jopa kehitysympäristön.

Android

Android on Linuxiin pohjautuva Googlen omistama mobiilikäyttöjärjestelmä, jolle sovelluksia kehitetään pitkälti Java-ohjelmointikielellä.

Java

Java on virtuaaliympäristössä ajettava olio-ohjelmointikieli, joka on useista turvallisuusuhistaan huolimatta käytössä hyvin laajasti ympäri maailmaa. Javan suosio pe-

rustuu sen helppoon kielioppiin sekä oliopohjaisuuteen. Java sisältää myös automaattisen muistinhallinnan, joka helpottaa sen käyttöä verrattuna esimerkiksi C-kieleen.

GPS

GPS eli Global Positioning System on alun perin sotilaskäyttöön tarkoitettu Yhdysvaltain puolustusministeriön kehittämä satelliittipaikannusjärjestelmä.

Glonass

Glonass on Venäjän vastine GPS-satelliittipaikannusjärjestelmälle.

NMEA

NMEA tai oikeastaan NMEA 0183 on National Marine Electronics Associationin määrittelemä viestiprotokolla, jota esimerkiksi Masser BT Caliper käyttää.

1 OPINNÄYTETYÖN LÄHTÖKOHDAT

Metsän mittauksen tarkoituksena on luoda inventaariota ja saada konkreettisia perusteita metsän arvon arvioimiselle. Yleisesti mittauksia suorittavat pienet tiimit, jotka kantavat mukanaan kallista laitteistoa. Opinnäytetyön yhtenä tarkoituksena oli tutkia mahdollisia keinoja vähentää mittaukseen vaadittavia resursseja, kuitenkin samalla tuottaen vertailukelpoista dataa. Työn toisena tavoitteena oli tutkia, kuinka HTML5 teknologioita voitaisiin hyödyntää kehitettäessä mobiilisovelluksia.

Työn teoriaosuudessa käydään läpi lyhyesti metsän mittauksen teoriaa, mobiilikehitystä yleisesti, sekä erilaisia HTML5-nimikkeen alle niputettavia web-tekniikoita ja ohjelmointikehyksiä. Teoriaosassa käsitellään myös lyhyesti sovelluksessa käytettyjä karttateknologioita ja GPS-satelliittipaikannusjärjestelmää. Toteutusosassa käsitellään mittausta helpottamaan luodun mobiilisovelluksen kehitysprosessia ja toteutusta. Opinnäytetyössä edellä mainittuihin asioihin pureudutaan sovelluskehittäjän näkökulmasta.

Opinnäytetyön työnantajana toimi Jyväskylän Ammattikorkeakoulun projektialue. Työ tehtiin osana Metsävaramittaus kunnostamattomissa metsissä – projektia, jossa tavoitteena on muun muassa edistää metsän inventointia varsinkin nuorien kunnostamattomien metsien osalta.

2 METSÄN INVENTOINTI

Suomessa metsä jaetaan perinteisesti kuvioihin ja edelleen mikrokuvioihin. Nämä kuviot edustavat yleensä homogeenistä metsä-aluetta, eli kuvion sisällä metsän tyyppi ei hirveästi muutu. Kuvioilla tapahtuva maastomittaus on perinteisesti ollut koealapohjaista. Tämä tarkoittaa sitä, että kuviolle suoritetaan muutama ympyrän muotoinen ala, jolta kaikki puut mitataan. Näitä koealoja edelleen tarkastelemalla voidaan muodostaa yleinen käsitys kuvion senhetkisestä tilasta.

Nykyisin käytössä olevat metsän mittausmenetelmät perustuvat laserkeilatun aineiston käsittelyyn ja analysointiin. Tämän lisäksi maastossa suoritetaan täydentävää mittausta, jossa kuviolta mitataan koealoja. Koealat ovat yleisesti ympyräpohjaisia

alueita, joilla on kiinteä säde. Koealojen mittaaminen maastossa on kuitenkin työlästä, aikaa vievää sekä potentiaalisesti hankalaa riippuen maaston vaikeakulkuisuudesta. (Vastaranta, Holopainen, Kaartinen, Hyyppä & Hyyppä 2009.)

”Koealan mittaustehtäviin on jo olemassa automatisoituja tai puoliautomaattisia tekniikoita, mutta niistä ei ole vielä kehitetty maasto-olosuhteissa täysin toimivia metsäsovelluksia” (Mt.).

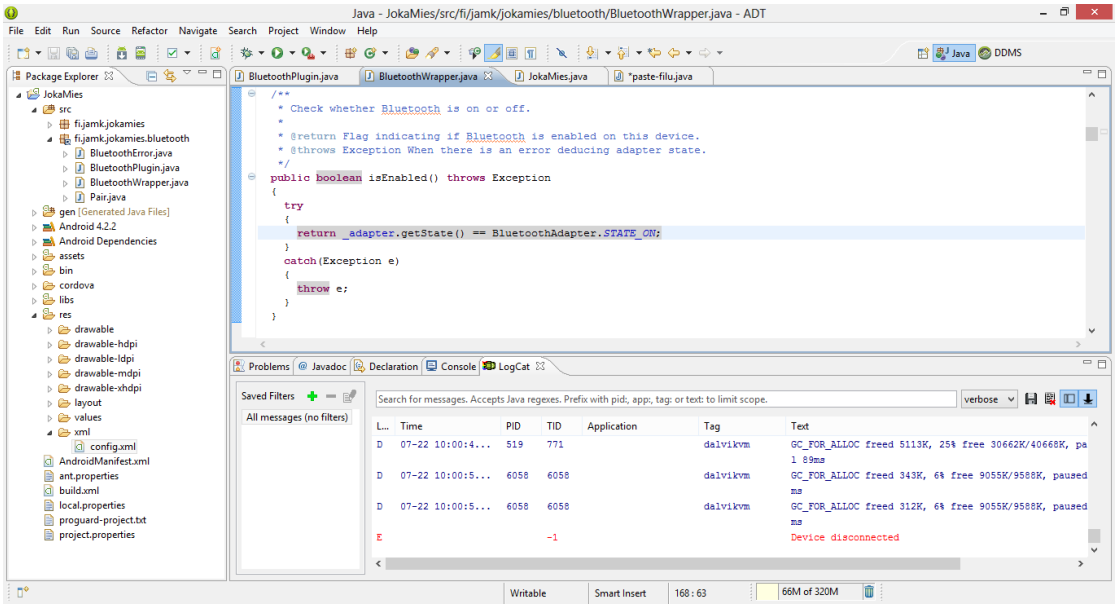
Tamperelainen Trestima Oy on kehittänyt mobiilisovellusta, jossa metsä kuvataan matkapuhelimella ja tämän jälkeen kuvat analysoidaan pilvipalvelussa konenäön avulla. Sovellus tuottaa automaattisesti puustotunnusraportteja, jotka näkyvät suoraan mittaajan laitteessa. Trestima Oy:n kehittämä menetelmä on myös huomattavasti nopeampi verrattuna perinteisiin koealapohjaisiin menetelmiin, joita käytetään vielä tänä päivänä. (Trestima Oy 2013.)

3 MOBIILIKEHITYS

3.1 Alustana Android

Android on alun perin Android Inc:n kehittämä, myöhemmin Googlen ostama ja edelleen kehittämä mobiilikäyttöjärjestelmä. Kirjoitushetkellä Androidin osuus älypuhelinmarkkinoista oli 64 %, josta suurin osa Samsungin tuotteita. Toukokuussa 2013 Googlen Play kaupasta oli ladattu ja asennettu noin 48 miljardia sovellusta. (Android (operating system) 2013.)

Perinteisesti Android sovellukset kehitetään Java-ohjelmointikielellä, jota ajetaan Androidista löytyvällä Dalvik-virtuaalikoneella. Tämä eroaa perinteisestä Javan virtuaalikoneesta, joka perustuu pinopohjaiseen arkkitehtuuriin siinä, että Dalvik perustuu rekisteripohjaiseen arkkitehtuuriin (Dalvik (software) 2013). Kehitys onnistuu myös natiivikoodiksi käännettävillä kielillä kuten C ja C++, joita varten Androidiin on olemassa NDK (Native Development Kit) (Android software development 2013). Ohjelmointiympäristönä käytetään useasti Eclipse-kehitysympäristöä, joka nähdään kuviossa 1.



KUVIO 1. Eclipse-kehitysympäristö

Oletusarvoisesti jokainen Android sovellus pyörii omalla virtuaalikoneellaan, jolloin sovelluksen koodi on eristettyä. Kahden sovelluksen on kuitenkin mahdollista pyöriä samalla virtuaalikoneella edellyttäen, että niiden sertifikaatit ovat samat. Android sovelluksiin sisällytetään manifesti, jossa järjestelmälle kerrotaan esimerkiksi mitä oikeuksia sovellus vaatii ja mitkä sen järjestelmävaatimukset ovat. Sovellukset koostuvat pääasiassa neljästä eri komponentista: aktiviteeteista (activities), palveluista (services), sisällöntarjoajista (content providers) ja vastaanottimista (broadcast receivers). (Application Fundamentals n.d.)

Aktiviteetit

Aktiviteetti on käytännössä yksi sovelluksen ruutu, jossa on käyttöliittymä. Tavallisesti sovelluksissa on useampia aktiviteetteja. (Mt.)

Palvelut

Palvelut ovat pitkäaikaisia prosesseja, joilla ei ole välttämättä määrättyä lopetusta. Palveluilla ei ole käyttöliittymää, vaan ne suorittavat operaationsa taustalla. Palveluita voi jakaa sovellusten välillä. (Mt.)

Sisällöntarjoajat

Sisällöntarjoajat hallinnoivat sovelluksen dataa, joka voi olla tallennettu esimerkiksi tietokantaan tai paikalliseen tiedostojärjestelmään. Sisällöntarjoajia voi jakaa sovellusten välillä. (Mt.)

Vastaanottimet

Vastaanottimet vastaanottavat koko järjestelmän kattavia lähetyksiä, käytännössä viestejä esimerkiksi siitä, että Bluetoothin on laitettu pois päältä tai että ruutu on sammunut. Vastaanottimet rekisteröidään kuuntelemaan niitä lähetyksiä, mitä halutaan. (Mt.)

3.2 PhoneGap

PhoneGap on mobiilikehitys-ohjelmistokehys, joka mahdollistaa mobiililaitteille kehityksen käyttäen web-tekniikoita perinteisien alustakohtaisien kielien sijaan. PhoneGap tarjoaa rajapinnat moniin laitteen ominaisuuksiin kuten kompassiin ja paikannukseen. PhoneGapista puhutaan usein samassa yhteydessä kuin Apache Cordovasta, joka on PhoneGapin alla piilevä avoimen lähdekoodin ohjelmisto. (PhoneGap 2013.)

PhoneGap toimii tarjoamalla natiivin web-näkymän, jolla on rajapinta laitteen ominaisuuksiin. Tässä web-näkymässä voidaan pyörittää HTML5 koodia, joka ei ole mitenkään riippuvaista kehitysalustasta. PhoneGapin vahvuus verrattuna perinteiseen mobiilikehitykseen syntyykin kehitettävän ohjelman laiteriippumattomuudesta: ohjelma voidaan siirtää usealle eri laitteelle hyvin vähällä vaivalla. PhoneGapin heikkoutena voi mainita vaihtelevan suorituskyvyn eri alustojen välillä. Optimisaatio on yksi avaintekijöistä kehitettäessä PhoneGap sovellusta, joka tuntuu samalta kuin perinteinen sovellus. (Mt.)

PhoneGap kehityksessä varsinkin debuggaus ja testaus on hankalampaa kuin perinteisessä HTML5 sovelluksessa. Työpöytäselaimella kuten Chromella tai Firefoxilla HTML5 koodin testaus on usein vaivatonta monien tehokkaiden työkalujen kuten Firebugin tai Chrome Developer Toolsin kautta. Mobiililaitteelle kehitettäessä näitä työkaluja on usein paljon hankalampi hyödyntää, varsinkin jos sovellus nojaa joihinkin PhoneGapin rajapintoihin joita ei perinteisellä selaimella voi käyttää. (Avola 2012.)

Pluginit eli liitännäiset ovat tärkeä osa PhoneGap sovellusta. Niiden avulla web-näkymässä pyörivät sovellukset pääsevät kiinni natiivista alustasta löytyvään toiminnallisuuteen. Käytännössä myös valmiit rajapinnat kuten kamera ja geolokaatio ovat liitännäisiä, mutta ne on paketoitu valmiiksi alustakohtaiseen PhoneGap jakeluun. Liitännäiset koostuvat JavaScript rajapinnasta ja natiivista toteutuksesta, joka täytyy toistaa jokaiselle halutulle alustalle. (Plugin Development Guide n.d.)

3.3 Bluetooth

Bluetooth on alun perin Ericssonin kehittämä avoin standardi langattomaan lähietäisyydellä tapahtuvaan kommunikointiin. Sopivan matkan päässä olevat Bluetooth-laitteet voidaan parittaa toisiinsa, jolloin syntyy niin sanottu ad hoc -verkko. Tässä verkossa eli ”piconetissä” laitteet ovat yhteydessä toisiinsa perustuen master-slave protokollaan. (Bluetooth 2013.)

Android sovelluksessa Bluetooth ominaisuudet saadaan käyttöön merkitsemällä sovelluksen manifestiin oikeat oikeudet. BLUETOOTH-oikeus antaa sovellukselle oikeuden pyytää yhteyttä, ottaa vastaan yhteyksiä ja siirtää dataa. BLUETOOTH_ADMIN-oikeus puolestaan antaa sovellukselle luvan käyttää laitteen muita Bluetooth-ominaisuuksia kuten laitteiden etsiminen tai Bluetooth-tilan muuttaminen. Näitä ominaisuuksia käytettäessä tulisi muistaa tarvittaessa käyttää Android alustan Intent-ominaisuutta. (Bluetooth – Android Developers n.d.)

Android-alustalla toiseen laitteeseen yhdistäminen tapahtuu yleensä luomalla sille oma säikeensä, jossa yhteydelle avataan *BluetoothSocket*. Yhdistäminen suoritetaan yleensä omassa säikeessään siksi, että se on verrattain raskas ja hidas prosessi eikä sen aikana tulisi suorittaa muita Bluetooth-toimintoja kuten laitteiden etsintää. Seuraavassa on esitetty hyvin riisuttu tapa muodostaa yhteys toiseen laitteeseen ohjelmallisesti:

```
private class ConnectionAttempt
  extends AsyncTask<Void, Void, BluetoothSocket>
{
  BluetoothSocket _socket;

  public ConnectionAttempt(BluetoothDevice device, UUID uuid)
    throws Exception
  {
    _socket = device.createRfcommSocketToServiceRecord(uuid);
  }
```

```
  @Override
  protected BluetoothSocket doInBackground(Void... params)
  {
    _socket.connect();
    return _socket;
  }

  @Override
  protected void onPostExecute(BluetoothSocket connectedSocket)
  {
    // ...
  }
}
```

Bluetooth-yhteyttä hallinnoidaan omassa säikeessään. Yhteyden hallinnoimisessa on syytä muistaa olla tarkkana: BluetoothSocket-instanssit tulisi sulkea, jos yhteydessä on jotain vikaa. Java-kielellä ohjelmoitaessa on syytä myös sulkea yhteyden hallinnoimiseen liittyvä koodi try-catch lohkon sisään unohtamatta kuitenkaan käsitellä virheitä asiaankuuluvalla tavalla. Seuraavassa yksinkertaistettu esimerkki, jossa yhteydestä luettu data kirjoitetaan takaisin lähettäjälle ohjelmallisesti:

```
private class ConnectionManager
  extends Thread
{
  private final BluetoothSocket _socket;
  private final InputStream     _input;
  private final OutputStream    _output;

  public ConnectionManager(BluetoothSocket socket)
  {
    _socket = socket;
    _input  = socket.getInputStream();
    _output = socket.getOutputStream();
  }

  @Override
  public void run()
  {
    byte[] buffer = new byte[1024];

    while(true)
    {
      int bytes   = _input.read(buffer);
      byte[] data = new byte[bytes];

      for(int i = 0; i < bytes; i++)
      {
        data[i] = buffer[i];
      }
      _output.write(data);
    }
  }
}
```

4 WEB-TEKNIIKAT

4.1 Responsiivisuus

4.1.1 Mitä on responsiivisuus?

Web-sovelluksen responsiivisuudella tarkoitettiin alun perin sivuston kykyä mukautua päätelaitteelle käyttäen erilaisia CSS3 ratkaisuja. Käsite on kuitenkin laajentunut ulkoasun responsiivisuudesta tarkoittamaan myös itse sisällön responsiivisuutta. Tällä voidaan tarkoittaa esimerkiksi pienempien kuvien lataamista mobiilipäätteille, mikä säästää huomattavan määrän kaistaa ja saa sivuston tuntumaan nopeammalta. (Rand-Henriksen 2013.)

Responsiivisuutta saavutetaan usein käyttämällä *media queryja* eli vapaasti suomennettuna mediakyselyitä niin, että käyttäjälle esitetään käyttäjän päätelaitteelle optimoitu sisältö. Seuraavassa esimerkki, jossa elementin korkeus muuttuu riippuen laitteen näennäisestä asennosta:

```
@media all and (orientation:landscape) {
    .example {
        height: 180px;
    }
}

@media all and (orientation:portrait) {
    .example {
        height: 360px;
    }
}
```

Responsiivisuuteen liittyvät läheisesti myös käsitteet *mobile-first*, *unobtrusive JavaScript* ja *progressive-enhancement* (Responsive web design 2013). Mobile-first, eli mobiili-ensin tarkoittaa pitkälti web-sivujen suunnittelemista ensin mobiilipäätelaitteille ja vasta sitten muihin ympäristöihin. Tähän tapaan toimia liittyy vahvasti termi progressive-enhancement, jolla tarkoitetaan sivun skaalaamista ylöspäin tehokkaimpiin mobiili tai työpöytäympäristöihin, eli ominaisuuksien lisäämistä riippuen sivustoa käyttävästä asiakasohjelmasta. (Johnson 2013.)

Termi unobtrusive JavaScript eli vapaasti suomennettuna ”huomaamaton” JavaScriptin käyttö tarkoittaa käytännössä modernia JavaScriptin käyttöä. Ensimmäinen tavoite on erottaa JavaScript koodi HTML-merkintäkielestä täysin, mikä säästää web-

kehittäjien aikaa sekä hermoja. Vaikeampana tavoitteena voidaan pitää sivuston ”arvokasta hajoamista”, jolla tarkoitetaan sivun ominaisuuksien hiljaista karsimista sitä mukaa, kun ominaisuuksiin liittyvien skriptien suorittaminen epäonnistuu. Kolmas, ja ehkä vaikein tavoite on sivuston pääominaisuuksien tarjoaminen riippumatta siitä, onko sivuston JavaScriptia saatu ajettua riviäkään onnistuneesti. (Unobtrusive JavaScript 2013.)

Seuraavassa esimerkki, jossa JavaScript on sotkettu merkintäkielen sekaan:

```
<!-- JavaScript sotkettu merkintäkieleen -->
<script type="text/javascript">
    function doSomething()
    {
        console.log("Hei maailma!");
    }
</script>
<button id="btnExample" onclick="doSomething()">Paina minua!</button>
```

Seuraavassa puolestaan esimerkki, jossa JavaScript on erotettu merkintäkielestä:

```
<!-- JavaScript erotettu merkintäkielestä -->
<button id="btnExample">Paina minua!</button>

// Käytetty jQuery kirjastoa
$("#btnExample").on("click", function () {
    console.log("Hei maailma!");
});
```

4.1.2 JQuery Mobile

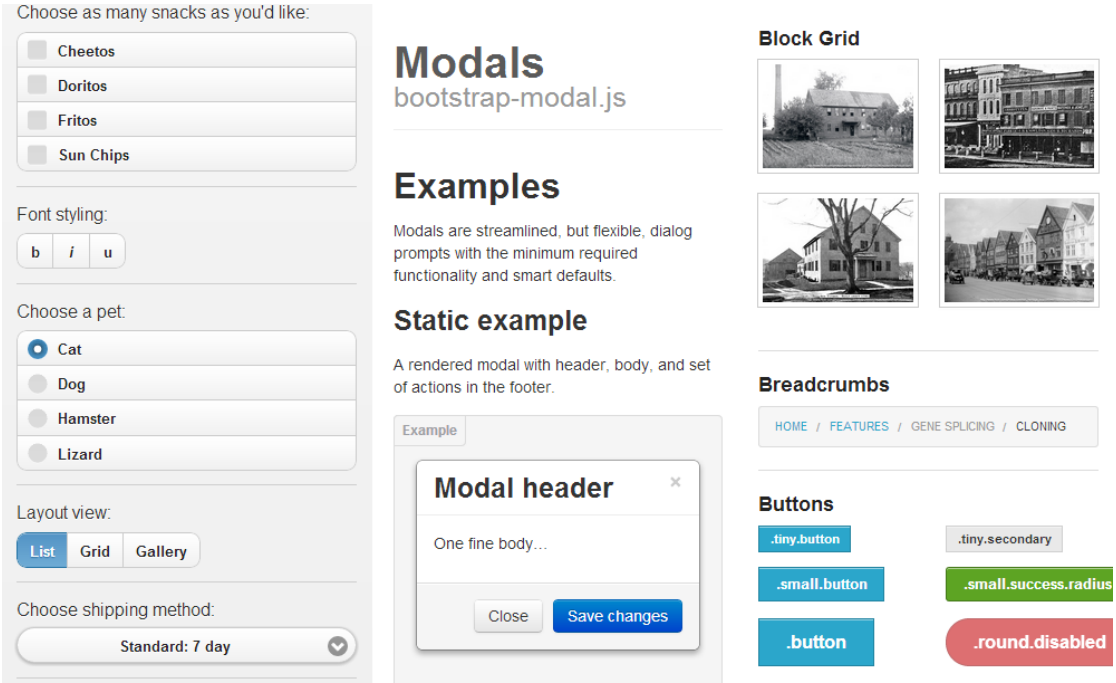
JQuery on alun perin John Resigin kehittämä JavaScript kirjasto, joka mm. helpottaa DOM-rakenteen manipuloimista koodin puolella. Se sisältää myös muita elämää helpottavia ominaisuuksia, joiden vuoksi jQuery on yleisin käytössä oleva JavaScript kirjasto. jQuery on myös helposti laajennettavissa erilaisilla liitännäisillä, joita on jo valmiina monia. (jQuery 2013.)

JQuery Mobile on puolestaan web-ohjelmointikehys, joka on optimoitu kosketuslaitteille, kuten puhelimille ja tableteille. Kuten on yleistä web-ohjelmointikehyksille, tarjoaa jQuery Mobile kattavan valikoiman erilaisia *”widgettejä”*, kuten painikkeita ja valikkoja. Responsiiviset ja mobiilit ohjelmointikehykset pitävät sisällään yleensä jonkinlaisen ruudukko eli *”grid”*-systeemin. Tämä tarkoittaa sitä, että sivun sisältö saadaan mukautumaan ruudun koon mukaan helposti ilman, että kehittäjä saa tästä harmaita hiuksia. Muista tässä esitellyistä web-ohjelmointikehyksistä poiketen jQuery

Mobile ei keskity pelkästään ulkonäöllisiin seikkoihin, vaan se sisältää myös ominaisuuksia, jotka mahdollistavat esimerkiksi sovelluksen sisäisen navigoinnin helpon implementoinnin (jQuery Mobile 2013). Seuraavassa esimerkki sivusta, jolla on yläpalkki ja sisältönä listanäkymä.

```
<!-- Itse sivun määritys -->
<div id="testi" data-role="page" data-title="Ankat" data-url="ankat">
    <!-- Yläpalkki -->
    <div data-role="header">
        <h1>Ankkasivu</h1>
    </div>
    <!-- Sivun sisältö -->
    <div data-role="content" data-theme="c">
        <h1>Ankat</h1>
        <ul data-role="listview" data-inset="true">
            <li data-icon="arrow-r">Aku</li>
            <li data-icon="arrow-r">Iines</li>
        </ul>
    </div>
</div>
```

Muita yleisesti käytettyjä jQuery Mobilen kaltaisia työkalupakkeja ovat esimerkiksi Bootstrap ja Foundation. Bootstrap on alun perin Twitterin sisäisien työkalujen käyttöön kehitetty käyttöliittymäkehys, joka takasi yhdenmukaisen ulkoasun työkalujen välillä (Twitter Bootstrap 2013). Foundation on ZURB-yhtiön kehittämä Bootstrapin kaltainen käyttöliittymäkehys, joka pyrkii helppoon muokattavuuteen ja keveyteen. Foundation käyttää nykyään jQueryn sijasta Zeptoa yleiskirjastonaan, joka on käytännössä kevyempi versio jQurysta (Foundation (framework) 2013). Edellä mainitut työkalupakit kuitenkin eroavat jQuery Mobilesta siinä, etteivät ne yritä millään tavalla sotkeutua itse sovelluksen arkkitehtuuriin tai mahdollisiin ohjelmoinnillisiin ratkaisuihin, vaan ne keskittyvät pääasiassa visuaalisuuteen. Kuviossa 2 on esitelty näiden kehyksien ulkoasua perinteisillä DOM-elementeillä.



KUVIO 2. JQuery Mobile, Boostrap ja Foundation 4 vierekkäin.

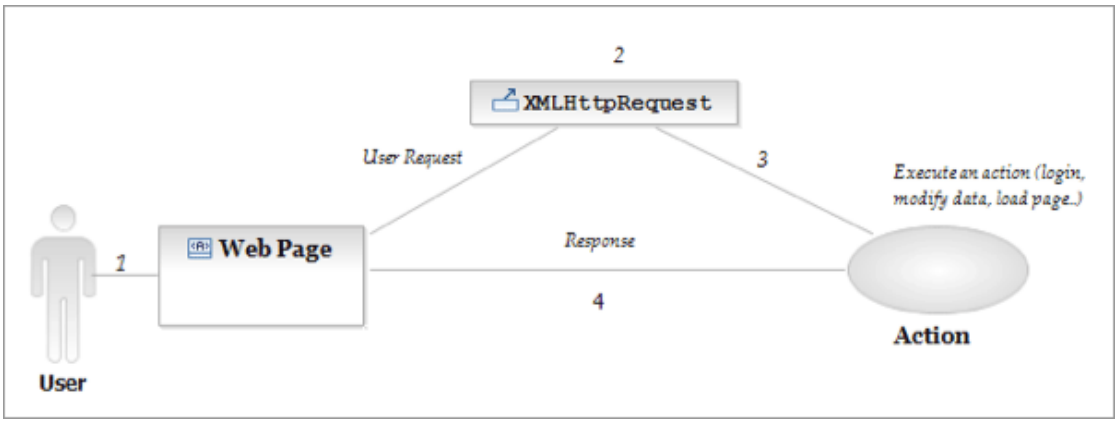
4.2 Asynkronisuus

Web-tekniikoista puhuttaessa asynkronisuudella tarkoitetaan yleensä Ajax-tekniikoita. Ajax-tekniikat ovat asiakasohjelman puolella käytettäviä tekniikoita, joiden avulla ohjelma voi lähettää http pyyntöjä asynkronisesti, eli ohjelman suoritus ei keskeydy pyynnön suorittamisen ajaksi. Käytännössä kaikki modernit web-sovellukset perustuvat asynkronisuuteen jollain tasolla: sivuja harvoin ladataan uudelleen käyttäjän syötteiden pohjalta, vaan ne käsitellään asynkronisesti. (Ajax (programming) 2013.)

Ajax toimii luomalla ensin selainkohtaisen pyyntö-objektin: tämä objekti on Internet Explorerilla *ActiveXObject* ja muilla selaimilla JavaScriptiin sisäänrakennettu *XMLHttpRequest*. Tähän objektiin sisällytetään tieto pyynnön käyttämästä metodista, pyynnön osoitteesta ja siitä tulisiko pyyntö käsitellä asynkronisesti. Pyynnön metodi tarkoittaa vastaanottajan päässä haluttua toimintoa: esimerkiksi tietoa haettaessa käytetään usein GET-metodia ja tietoa lähetettäessä POST-metodia. Pyyntö voidaan myös käsitellä synkronisesti asynkronisuuden sijaan. Tämä tarkoittaa sitä, että ohjelman suoritus keskeytyy pyynnön ajaksi. Pyyntö-objektin muodostamisen jälkeen se lähetetään vastaanottajalle, jolta saatu vastaus voidaan tulkita esimerkiksi objektista löytyvän *status*-kentän pohjalta. Pyynnön status-kenttä ilmaisee HTTP-statuskoodin,

joka tarkoittaa käytännössä pyynnön suoriutumista, esimerkiksi statuskoodi 200 tarkoittaa pyynnön onnistumista. Jos pyyntö on suoritettu onnistuneesti, voidaan vastaus lukea merkkijonona *responseText* kentästä. (AJAX – Asynchronous JavaScript and XML 2008.)

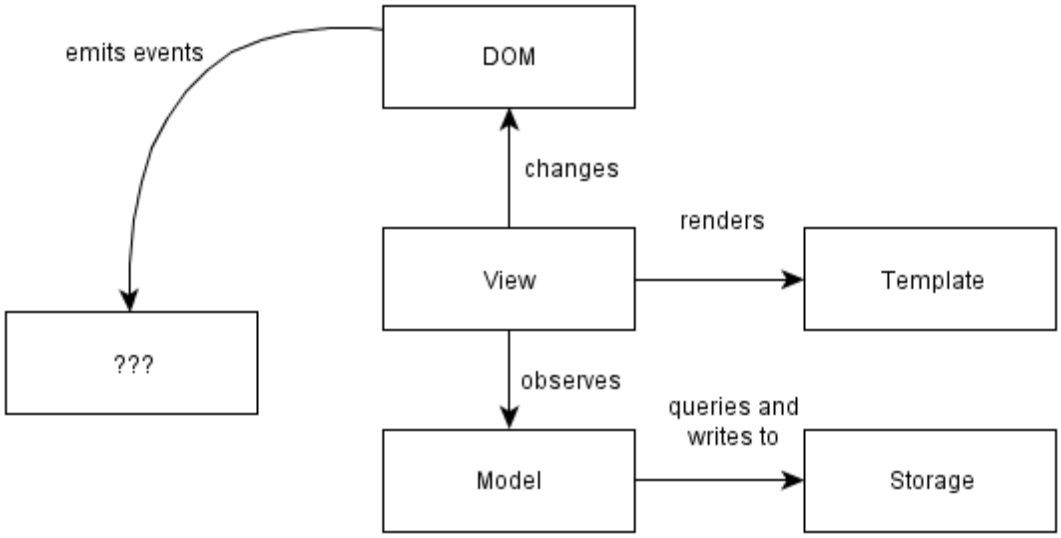
Kuviossa 3 on esitetty yksinkertainen kuvaus Ajaxin toiminnasta.



KUVIO 3. Ajax pähkinänkuoressa (How Ajax Works 2008)

Nykyajan web-sovellukset ovat usein SPA (Single-page application) mallin mukaisia. Tämä tarkoittaa sitä, että koko sivusto on käytännössä yksi sivu, jonka sisältö ladataan asynkronisesti käyttäjän toimien perusteella (Single-page application 2013). Esimerkiksi navigoidessa sovelluksen toiseen osioon ladataan sen vaatima sisältö ja esitetään se käyttäjälle ilman, että sivua varsinaisesti ladataan uudelleen. Modernit yksisivuiset sovellukset on yleensä rakennettu niin, että sovelluksen tieto mallinnetaan eräänlaisilla malli-olioilla (model), joita sovelluksen eri kontrollerit (controller) tarkkailevat, ja piirtävät tarvittavan HTML rakenteen sovelluksen DOM-puuhun (ks. kuvio 2). Yksisivuisista sovelluksista puhuttaessa käytetään kontrolleri-termin tilalla usein myös termiä näkymä (view), sillä kontrolleri vastaa käytännössä näkymän päivittämisestä ja muusta koodista, joka liimaa varsinaisen käyttäjälle näkyvän osuuden sovelluksen sisäiseen tilaan ja malleihin. (Takada. N. d.)

Kuvio 4 pyrkii selventämään yksisivuisen sovelluksen rakennetta ja toimintaa.



KUVIO 4. Yksisivuisen sovelluksen rakenne (Takada N. d.)

4.3 Paikallinen tallennus

Ennen paikallista varastointia selainpohjaisien sovelluksien haittana on ollut kykenemättömyys tallentaa (merkittävää määrää) tietoa paikallisesti. Web-sovelluksien kyky tallentaa tietoa paikallisesti sovelluksen sen hetkisestä tilasta mahdollistaa pienempien tietomäärien siirron palvelimen ja asiakasohjelman välillä, mikä puolestaan nopeuttaa sovelluksen toimintaa. Keksit eli *cookiet* olivat aikaisin web-tekniikoiden historiassa keksitty teknologia, jonka kautta sovellukset pystyivät tallentamaan ”pysyvästi” pieniä määriä dataa. Keksien tapauksessa niiden kapasiteetti oli kuitenkin verrattain pieni (4 KB), ja ne kulkivat mukana jokaisessa pyynnössä asiakasohjelman ja palvelimen välillä. (Pilgrim N. d.)

Paikallisen varaston historia alkoi Microsoftin Internet Explorer -selaimen kyvystä tallentaa tietoa paikallisesti. Tämä kuitenkin rajoittui Internet Explorer -selaimeen, eikä tallennuskapasiteetti ollut mikään massiivinen (64 KB). Adoben yritys poiki pitemmän historian, kun Flash 6 mahdollisti noin 100 KB:n tallennuskapasiteetin verkkotunnusta kohti. Kysymys paikallisesta varastosta poiki myös muita viritelmiä, ja Dojo Toolkit ohjelmistokehys pyrki integroimaan nämä kaikki yhtenäisen rajapinnan taakse. Varsinaisen HTML5-spesifikaation myötä saatiin myös määritelmä *Web Stora-*

gelle, josta käytetään usein nimitystä *HTML5 Storage* tai *Local Storage*. HTML5 Storage mahdollistaa tiedon tallentamisen avain – arvo-pareihin yhtenäisen rajapinnan taakse, joka on sisäänrakennettu selainohjelmiin (Mt).

Seuraavassa esimerkki HTML5 Storagen käytöstä.

```
// tallennetaan tieto paikallisesti
window.localStorage.setItem("Koira", "Hessu");

// haetaan tieto paikallisesti
window.localStorage.getItem("Koira");
```

HTML5 Storagen heikkouksina voidaan mainita sen rajoitettu kapasiteetti (5 MB), joka on kuitenkin huomattava parannus edeltäjiinsä verrattuna. Avain – arvo-pareihin perustavan HTML5 Storagen rinnalla elää kuitenkin muita ratkaisuja paikalliseen tallennukseen: *WebDatabase* perustuu *SQLite* tietokantaan, johon voi suorittaa perinteisiä SQL komentoja suoraan JavaScript koodista (Mt). WebDatabasen SQL ”murretta” ei ole kuitenkaan standardoitu mitenkään, vaan se perustuu täysin SQLiten murteeseen. (Dive Into …) Koska SQLite on kuitenkin web-tekniikoista erillään oleva ohjelmisto, on yleisen rajapinnan perustaminen siihen hieman riskialtista: muutokset SQLite ohjelmistoon heijastuvat suoraan tarpeena tehdä muutoksia niihin web-sovelluksiin, jotka käyttävät WebDatabase rajapintaa. (Ranganathan 2010.)

Ratkaisuna WebDatabase rajapinnan ongelmiin Mozilla on kehittänyt *IndexedDB* rajapinnan. IndexedDB rajapinta eroaa WebDatabasesta sillä, ettei se tarjoa suoraa SQL rajapintaa, vaan eräänlaisen ”objektivaraston”. Tätä varastoa voidaan iteroida läpi transaktion sisällä valmiiden JavaScript metodien avulla, eikä kehittäjän tarvitse kirjoittaa varsinaista SQL koodia (Ranganathan, Wilsher 2010). Seuraavassa yksinkertaiset esimerkit sekä WebDatabasen että IndexedDB:n käytöstä.

```
// (WebDatabase) tietokannan avaus, määritelty nimi, versio, kuvaus ja koko
db = openDatabase("Koirat", "1.0", "Koirien tietokanta", 1024 * 1024);
// (WebDatabase) valitaan kaikki taulun Koira solut
db.transaction(function (tx) {
    tx.executeSql("SELECT * FROM Koira", [], function () { }, function () { });
});

// (IndexedDB) avataan tietokanta, määritelty nimi ja versio
request = indexedDB.open("Koirat", 1);
request.onsuccess = function (ev) {
    var db = ev.target.result;
    var transaction = db.transaction("Koira")
    var objects = transaction.objectStore("Koira");
    // Haetaan objekteista avaimella 1
    var req = objects.get(1);
    req.onsuccess = function () {
        var name = req.result.nimi;
    }
}
```

4.4 CoffeeScript

CoffeeScript on Jeremy Ashkenasin kehittämä ohjelmointikieli, joka kääntyy puhtaaksi JavaScript ohjelmointikieleksi. Sen tarkoituksena on ollut parantaa JavaScriptin usein kyseenalaista luettavuutta lisäämällä erilaisia luettavuutta parantavia ilmaisutapoja. CoffeeScript siis lisää JavaScriptiin *"syntactic sugaria"*. Ehkä näkyvin ero JavaScriptiin on siinä, ettei CoffeeScript käytä aaltosulkeita eri koodisulkeiden kuten funktioiden määrittämiseen, vaan eri sulkeet ilmaistaan sisennyksen avulla (CoffeeScript 2013).

Seuraavana esimerkki RequireJS ja PhoneGap sovelluksesta, joka demonstroi näkyvimmät erot CoffeeScriptin ja sen tuottaman JavaScriptin välillä.

```
# CoffeeScript implementaatio
require.config
    baseUrl: "js/"
    paths:
        templates:       "../templates"
        text:            "lib/require-text-1.0.8"

($ document).on "deviceready", () ->
    window.plugins =
        bluetooth:       cordova.require "cordova/plugin/bluetooth"
        wakelock:        cordova.require "cordova/plugin/wakelock"

    require ["app", "utils/handlebars"], (App) -> new App
```

```
// JavaScript käännös CoffeeScript kääntäjällä
require.config({
    baseUrl: "js/",
    paths: {
        templates: "../templates",
        text: "lib/require-text-1.0.8"
    }
});

($(document)).on("deviceready", function() {
    window.plugins = {
        bluetooth: cordova.require("cordova/plugin/bluetooth"),
        wakelock: cordova.require("cordova/plugin/wakelock")
    };
    return require(["app", "utils/handlebars"], function(App) {
        return new App;
    });
});
```

CoffeeScript implementoi lisänä luokat ja perinnän. Tämän voi toki saavuttaa myös perinteisellä JavaScriptillä, mutta varsinkin perintä on hieman likaista työtä. Seuraavassa esimerkki kuinka tämä on toteutettu sekä CoffeeScriptillä, että sen tuottamalla JavaScriptillä.

```
# CoffeeScript
define (require) ->
    Events = require "utils/events"

    class Network extends Events
        constructor: () -> ...

// JavaScript
var __hasProp = {}.hasOwnProperty,
    __extends = function(child, parent) { for (var key in parent) { if
(__hasProp.call(parent, key)) child[key] = parent[key]; } function ctor() {
this.constructor = child; } ctor.prototype = parent.prototype; child.prototype
= new ctor(); child.__super__ = parent.prototype; return child; };

define(function(require) {
    var Events, Network;
    Events = require("utils/events");
    return Network = (function(_super) {
        __extends(Network, _super);
        function Network() { ... }
```

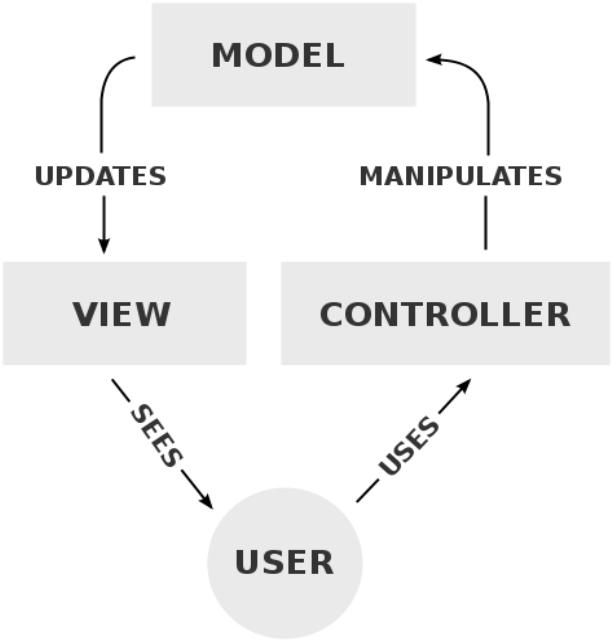
Koska CoffeeScript voidaan käsittää eräänlaiseksi vaihtoehtoiseksi syntaksiksi JavaScriptille, on se saanut osakseen myös kritiikkiä. Koska CoffeeScript käännetään JavaScriptiksi, on sen testaaminen hankalampaa johtuen selainten ilmoittamien virheiden viittaamisesta JavaScriptin puolelle. CoffeeScriptin kääntämä JavaScript ei ole aina täydellisen luettavaa johtuen sen tiiviistä syntaksista vaikkakin CoffeeScript kääntyy hyvin ennustettavalla tavalla JavaScriptiksi (Ryan Florence 2011). On kuitenkin huomattava, että CoffeeScript on jatkuvasti kehittyvä kieli ja se on toiminut vaikuttimena tuleviin JavaScriptin päivityksiin. (CoffeeScript 2013.)

5 SELAINPÄÄN MVC-OHJELMISTOKEHYKSET

5.1 Mikä MVC

MVC, eli *Model-View-Controller* on hyvin suosittu ohjelmistotekninen arkkitehtuuriratkaisu. Siinä erotetaan sovelluksen tieto malleiksi (Model), visuaalinen representaatio näkymiksi (View) ja näkymien ohjaaminen kontrollereiksi (Controller). Mallit sisältävät sovelluksen tiedon: esimerkiksi henkilörekisterissä henkilö olisi malli. Nämä mallit visualisoidaan käyttäjälle näkymien avulla: rekisterisovelluksessa jokainen näkyvä rivi olisi näkymä, joka puolestaan liittyisi johonkin malliin. Näitä näkymiä ohjaamaan sovelluksessa on kontrollereita: käyttäjän valitessa sovelluksessa rivin ja klikatessa kuvitteellista ”poista”-nappulaa, kontrolleri sisältäisi logiikan, joka poistamiseen tarvittaisiin. Nykyään web-sovelluksissa puhutaan myös MV*-arkkitehtuurista. MV*-arkkitehtuurissa sinulla on yleisesti malli ja näkymä, mutta muuten kortit ovat ilmassa: esimerkiksi näkymä saattaa periä kontrollerin tehtävät tai pakkaan on heitetty jotain muutakin. Varsinkin yhden sivun sovellukset ovat usein rakennettu jonkin MV*-arkkitehtuurin omaavan ohjelmointikehyksen, kuten *Backbonen* tai *Angularin*, päälle. (Osmani 2012.)

Kuviossa 5 on esitetty perinteisen MVC-arkkitehtuurin toiminta.



KUVIO 5. MVC-arkkitehtuurin toiminta (Model-view-controller 2013)

5.2 Backbone

Backbone on sekä Underscoren, että CoffeeScriptin kehittäjän Jeremy Ashkenasin kehittämä JavaScript ohjelmointikehys (Backbone.js 2013). Backbone on hyvin kevyt eikä tarjoa suuria ratkaisuja valmiiksi: tämän takia Backbonea haukutaan joskus myös kirjastoksi ohjelmointikehyksen sijaan.

Backbone ei ole varsinaisesti MVC-ohjelmointikehys, vaan jotain sinne päin. Se sisältää mallit (Model) ja näkymät (View), mutta kirjaimellisia kontrollereita siinä ei ole. Käytännössä Backbonen näkymä (View) ottaa vastuun kontrollerin tehtävistä ja vaikka se yleensä sisältää sovelluksen piirtologiikan, ei se ota kantaa varsinaiseen esitystapaan. Voidaankin sanoa, että Backbonen näkymä-komponentti on varsinaisesti kontrolleri jolloin näkymä on käyttäjälle esitettävä HTML. Backbone löytää yleensä paikkansa MV*-ohjelmointikehyksien perheestä, jonne myös Angular usein sijoitetaan. (Derick Bailey 2011.)

Alla esimerkki yksinkertaisen Backbone näkymän määrittelystä.

```
LoginController = Backbone.View.extend
    initialize: () ->
        template = require "text!templates/page-login.html"
        @template = Handlebars.compile template

    render: () ->
        ((@$ "[data-role='content']").html @template)
            .trigger "create"
```

Mallin lisäksi Backbonessa on myös kokoelmia (Collection), jotka koostuvat joistain määritellyistä malleista. Näiden valmiiden komponenttien avulla saadaan esimerkiksi tieto siitä, jos mallissa tai kokoelmassa jokin muuttuu jolloin näkymä voidaan päivittää. Seuraavana on esitetty yksinkertaistettu määrittely Backbonen mallille sekä kokoelmalle.

```
device = Backbone.Model.extend
    defaults:
        state: "disconnected"

devices = Backbone.Collection.extend
    model: device
```

Backbonen kokoelmiin on rakennettu sisään monet Underscore kirjaston utiliteetit, esimerkiksi suodattaminen tai tietyn mallin etsiminen onnistuu todella helposti. Underscore onkin ainoa varsinainen riippuvuus, joka Backbonella on ja sekin on hyvin kevyt kirjasto.

Viimeisenä osana Backbonessa on reititin (Router), jonka avulla yksisivuisten sovellusten implementointi on huomattavasti helpompaa. Backbonen reititin kuuntelee muutoksia sivun osoitteessa, ja kutsuu kehittäjän määrittämää funktiota sen muuttuessa. Yhden sivun sovelluksissa käytetään usein risuaitaa osoitteessa merkitsemään jotain aluetta sovelluksen sisällä, osoitteen muuttuessa kutsutaan esimerkiksi sovelluksen yhden näkymän piirtofunktiota, jolloin käyttäjälle näyttää siltä kuin hän olisi vaihtanut sivua sovelluksen sisällä ilman sivun uudelleenlatausta.

Alla on esitetty hyvin yksinkertainen esimerkki Backbonen reitittimestä.

```
App = Backbone.Router.extend
    routes:
        "#person/:id": "person"
        "#dogs": "dogs"

    person: (id) ->
        personController = new PersonController id
        do personController.render

    dogs: () ->
        dogsController = new dogsController
        do dogsController.render
```

Backbone on selvästi suunniteltu pitkälti REST-rajapintaa ajatellen. Siinä tieto liikkuu JSON-muodossa perustuen tapauskohtaisiin http-pyyntöjen metodeihin. Backbone osaakin muodostaa mallit ja kokoelmat suoraan JSON-datasta ilman kehittäjän peliin puuttumista. Backbone mahdollistaa kuitenkin toiminnan myös muidenkin kuin REST-rajapintojen kanssa johtuen sen antamasta mahdollisuudesta ylitse kirjoittaa joitain sen perusmetodeja.

5.3 Angular

Angular on alun perin 2009 julkaistu, nykyisin Googlen ylläpitämä JavaScript ohjelmistokehys, jossa on painotettu vahvasti sovelluksen eri osien, kuten logiikan ja käyttöliittymäkoodin erottamista toisistaan (AngularJS 2013.) Angular (tai AngularJS) on selvästi Backbonea pidemmälle viety ohjelmistokehys: se määrittää kaikki tarvittavat komponentit, mutta pitkälti myös sen miten niitä tulisi käyttää. Angular kuuluu samaan MV*-ohjelmistokehyksien sarjaan kuin Backbone: siinäkin on perinteiset komponentit kuten näkymä (View) ja kontrolleri (Controller), mutta ei valmiiksi määritel-

tyä mallia (Model) kuten Backbonessa. Angularissa on myös lisänä palveluita (Service) ja direktiivejä (Directive), jotka yhdessä edellä mainittujen peruskomponenttien kanssa muodostavat Angularin kokonaisuuden.

Angular-sovelluksen rakentaminen saa alkunsa moduulin (Module) määrittelystä. Moduuli tarkoittaa Angularissa pitkälti sovelluksen ylintä tasoa: moduulilla voi olla määrittely ja perinteistä main-metodia muistuttava ajo-blokki. Angular on muutenkin huomattavasti tarkempi verrattuna Backboneen siitä, kuinka sitä tulisi käyttää. Sovelluksilla on selkeä rakenne ja komponenteilla omat vastuunsa. Backbone puolestaan tarjoaa lähinnä nämä komponentit, mutta ei määrittele erityisen tarkasti kuinka niitä tulisi käyttää.

Alla on esitetty yksinkertainen moduulin määrittely Angular-sovelluksessa.

```
var app = angular.module("ExampleApp");
```

Scope on tärkeä käsite Angular-sovelluksessa. Se tarkoittaa käytännössä sovelluksen osille asetettuja rajoja. Esimerkiksi näkymään kiinnitetty kontrolleri pitää sisällään sen näkymän *scopen*, jota muokkaamalla näkymä saadaan päivittymään automaattisesti, ilman että se vaikuttaa muihin näkymiin. Näkymän automaattinen päivitys on yksi Angular-ohjelmistokehyksen myyntivalteista. Käytännössä *scope* on siis yksi Angularin sisäinen komponentti. Tämä komponentti on kehittäjän näkökulmasta tavallinen Angularilta ”pyydettävä” objekti, jota muokkaamalla näkymä saadaan päivittymään. Yleensä kontrolleri liittää *scopeen* perinteisen MVC-mallin mukaisesti jonkin mallin (Model), jonka arvojen muuttuessa näkymä (View) päivittyy. *Scope* siis toimii eräänlaisena liimana kontrollerin, ja näkymän välissä.

Seuraavassa on esitetty Angular-ohjelmistokehyksen mukainen kontrolleri, joka ”pyytää” *scope*n ja liittää siihen *hello*-muuttujan.

```
app.controller("HelloCtrl", function($scope, $routeParams) {
    $scope.hello = "Hello " + $routeParams.name;
});
```

Yksisivuisessa web-sovelluksessa sisältö usein ladataan sivulle asynkronisesti. Käytännössä Angular piilottaa tämän taakseen, ja automatisoi prosessin kehittäjän näkökulmasta. Angular-sovelluksessa reititys onnistuu valmiin *routeProvider* komponentin avulla, jolle annetaan halutun sivun templaatti, sekä siihen kuuluva kontrolleri. Tämä eroaa Backbonen reitittimestä määrittämällä suoraan tavan kuinka sitä tulisi käyttää.

Alla olevassa esimerkissä, on sovelluksen konfiguraatio-lohkossa määritelty *routeProvider* komponentille, kuinka sovellus käyttäytyy osoitteen muuttuessa. Huomaa, että *hello* HTML-tiedostoon viitataan relatiivisella osoitteella, eli tässä tapauksessa se olisi samassa kansiossa sovelluksen juuren kanssa. Aikaisemmin mainittu kontrolleri on nyt liitetty mukaan reitin määritykseen, jolloin siinä pyydetty *scope*-muuttuja viittaa nyt *hello*-tiedoston sisältöön.

```
app.config(function($routeProvider) {
    $routeProvider.when("hello/:name", {
        templateUrl:    "hello.html",
        controller:     "HelloCtrl"
    });
});
```

Angularissa näkymät ovat deklaratiivista HTML-koodia, joka muuttuu dynaamiseksi siihen liitettyjen direktiivien avulla. Direktiivit ovat käytännössä eräänlaisia HTML-tageja, jotka Angularin ”koostamisvaiheen” aikana liitetään niiden JavaScript toteutukseen. Direktiivitkin ovat täten käytännössä vain JavaScript koodia, jotka mahdollistavat dynaamisen toiminnan HTML-rakenteessa. Kehittäjä voi luoda täysin omia direktiivejään, jotka mahdollistavat hyvinkin

Angular pitää sisällään monia valmiita, useasti tarvittavia direktiivejä. Esimerkiksi *repeat* direktiivi toimii iteroimalla sen sisältävän näkymän *scopeen* liitetyn JavaScript taulukon läpi, toistaen direktiivin sisältävän elementin sisältöineen jokaisen iteraation kohdalla.

Seuraavassa pieni esimerkki *repeat* direktiivin käytöstä, ja sen tuottamasta HTML-rakenteesta.

```
$scope.items = ["Koira", "Kissa", "Musti", "Mirri"];

<ul>
    <li ng-repeat="item in items">
        <p>{{ item }}</p>
    </li>
</ul>

<ul>
    <li><p>Koira</p></li>
    <li><p>Kissa</p></li>
    <li><p>Musti</p></li>
    <li><p>Mirri</p></li>
</ul>
```

6 KARTTATEKNOLOGIAT

6.1 Koordinaattijärjestelmät ja GPS

Koordinaattijärjestelmä määrittää käytännössä pisteen sijainnin maapallolla. Yleisesti ottaen järjestelmissä on jonkinlaiset leveys- ja pituusasteet, joilla ilmaistaan maantieteelliset koordinaatit. Koska mannerlaatat liikkuvat hiljalleen, on suurta tarkkuutta vaativassa paikannuksessa nojattu alueellisiin koordinaattijärjestelmiin. Alueellisissa järjestelmissä on käytetty jotain kiinteää pistettä, jonka paikka ei muutu ajan myötä: esimerkiksi ETRS89 järjestelmä on kiinnitetty Euraasian mannerlaattaan. Suomessa on käytössä ETRS89 järjestelmään pohjautuva EUREF-FIN järjestelmä, jota varten on mitattu noin 450 kiinnepistettä Suomen alueen läheisyydestä. (Koordinaattijärjestelmä 2013.)

Maailmanlaajuisesti käytössä on Yhdysvaltain puolustusministeriön määrittämä WGS84-koordinaattijärjestelmä, jonka kiinnepisteen (tai origon) uskotaan sijaitsevan maapallon massan keskipisteessä. Maailmanlaajuisen WGS84-järjestelmän ero alueellisiin järjestelmiin vaihtelee mannerlaattojen liikkeistä johtuen. Kuitenkin Suomessa käytössä oleva EUREF-FIN-järjestelmä poikkeaa WGS84-järjestelmästä vain alle metrin. (WGS84 2013.)

GPS on Yhdysvaltain puolustusministeriön kehittämä satelliitteihin perustuva globaali paikannusjärjestelmä, joka perustuu WGS84-koordinaattijärjestelmään (GPS 2013.) Täten esimerkiksi mobiililaitteista saatava GPS-paikkatieto on WGS84-koordinaattijärjestelmän mukaista. Vaikka GPS on hyvin tarkka varsinkin pilvettömän taivaan alla, on suurta tarkkuutta vaativissa mittauksissa hyvä ottaa mukaan myös Venäjän puolustusministeriön kehittämä GLONASS-satelliittipaikannusjärjestelmä (Glonass 2013.) Nykyisin suurin osa uusista älylaitteista tukee myös GLONASS-paikannusta GPS-paikannuksen lisäksi, jolloin kehittäjän ei tarvitse huolehtia paikannuksen tarkkuudesta niin paljoa.

6.2 Leaflet

Leaflet on JavaScript-kirjasto mobiiliystävällisiä karttoja varten. Leaflet tukee myös hyvin *GeoJSON*-formaattia, jota projektissa käytettiin välittämään geolokaatio-dataa

palvelimen ja asiakasohjelman välillä. Leaflet toimii hakemalla karttadatan TMS (Tile Map Service) palvelusta, jossa varsinainen karttadata on pilkottu pieniin *tiileihin*, jotka ladataan näkymän siirtyessä oikeaan kohtaan. Leaflet karttoihin on myös mahdollista lisätä interaktiivisia ominaisuuksia, kuten erilaisia merkkejä. Leaflet käyttää WGS84-koordinaattijärjestelmää.

Alla olevassa esimerkissä kartta kiinnitetään ennalta määritettyyn DOM-elementtiin, jonka jälkeen näkymä asetetaan Jyväskylän päälle ja varsinainen data ladataan TMS palvelusta.

```
jkl = L.map("elementti")
jkl.setView([62.242639, 25.747358], 13)

L.tileLayer("http://tiles.kartat.kapsi.fi/peruskartta/{z}/{x}/{y}.jpg")
    .addTo(jkl)
```

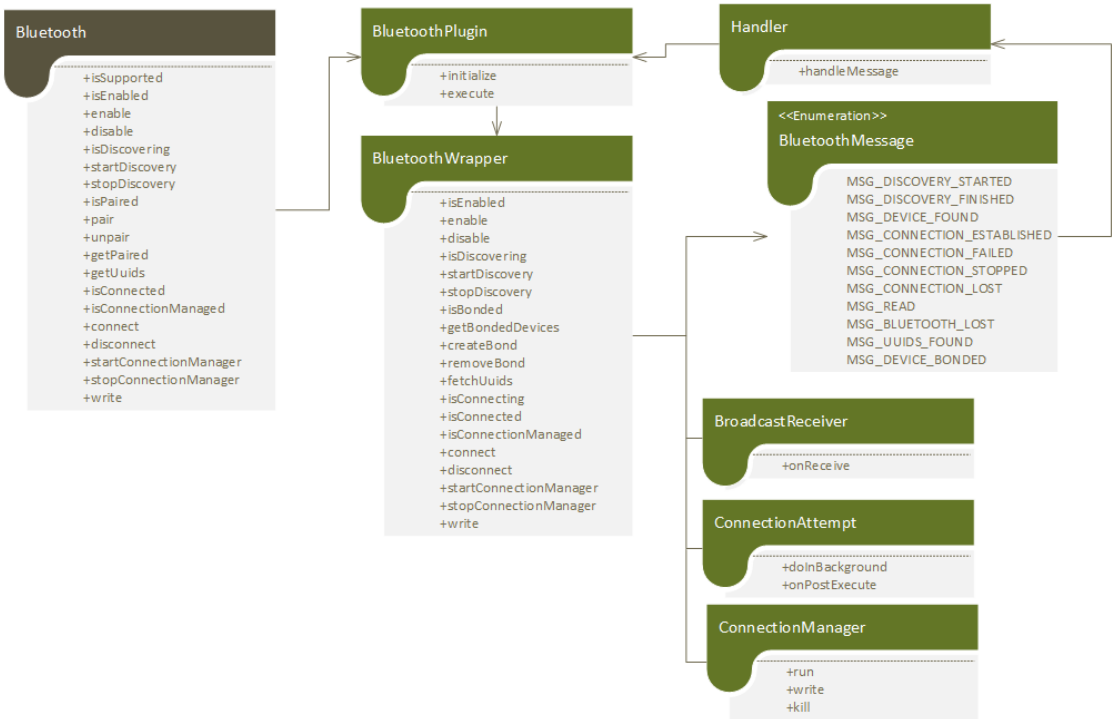
7 TYÖN TOTEUTUS

7.1 BluetoothPlugin

Projektin alussa oli selvää, että sovellus tulisi kommunikoimaan Bluetoothin välityksellä Masser BT Caliper mittalaitteen kanssa. Koska PhoneGap-alustalle ei löytynyt tällöin varteenotettavaa liitännäistä Bluetooth-yhteyksien hallintaan päätettiin sellainen kehittää itse.

Tavoitteena oli saada aikaan Java-ohjelmointikielellä kirjoitettu PhoneGap-liitännäinen Android alustalle, joka käyttäisi Androidin omaa Bluetooth-rajapintaa. Tämä liitännäinen puolestaan tarjoaisi JavaScript-rajapinnan HTML5-sovellukselle. Liitännäisen toiminta olisi tarkoitus saada mahdollisimman yksinkertaiseksi kuitenkin samalla tarjoten riittävästi kontrollia JavaScript-rajapinnan kautta.

Kuviossa 6 on esitetty Bluetooth-liitännäisen rakenne luokkakaaviona. Kuviota varten luokkakaaviosta on riisuttu privaatit jäsenet ja jäljelle jätetty ymmärtämisen kannalta olennaiset osat.



KUVIO 6. BluetoothPlugin rakenne

Kuviossa vasemmalla *Bluetooth* on JavaScript-rajapinnan määritys, joka kommunikoi Javalla toteutetun *BluetoothPlugin*-luokan kanssa. *BluetoothPlugin* on PhoneGap alustan määritysten mukaisesti *CordovaPlugin* luokasta peritty luokka, joka mahdollistaa sen kutsumisen JavaScript-rajapinnan kautta käyttäen PhoneGap (tai Cordova) kirjastosta löytyvää *exec*-funktiota.

```
// exec-funktion käyttö
var exec = require('cordova/exec');

Bluetooth.prototype.enable = function (onSuccess, onError) {
    exec(onSuccess, onError, "Bluetooth", "enable", []);
}
```

BluetoothPlugin käyttää Android-alustan Bluetooth-rajapintaa erillisen *BluetoothWrapper* luokan kautta. Näin PhoneGap-alustaan liittyvä ohjelmakoodi on erotettu Androidin Bluetooth-rajapintaan liittyvästä koodista. Johtuen usean Androidin Bluetooth-rajapinnan toiminnon asynkronisuudesta kertoo *BluetoothWrapper BluetoothPluginille* suoritetuista toiminnoista erillisen *Handler*-luokan kautta. *Handler* on Android-alustan oma viestinvälitykseen tarkoitettu luokka, joka tässä tapauksessa käsittelee *BluetoothMessage*-enumeraatioita. Seuraavassa on esitetty pieni osa *Handler*-luokan toiminnasta.

```
@Override
public boolean handleMessage(Message msg) {
  switch(msg.what) {
    case BluetoothWrapper.MSG_DISCOVERY_FINISHED:
      if(!_wasDiscoveryCanceled) {
        if(_discoveryCallback != null) {
          PluginResult result = new PluginResult(
            PluginResult.Status.OK, false
          );
          _discoveryCallback.sendPluginResult(result);
          _discoveryCallback = null;
        }
      }
      break;
```

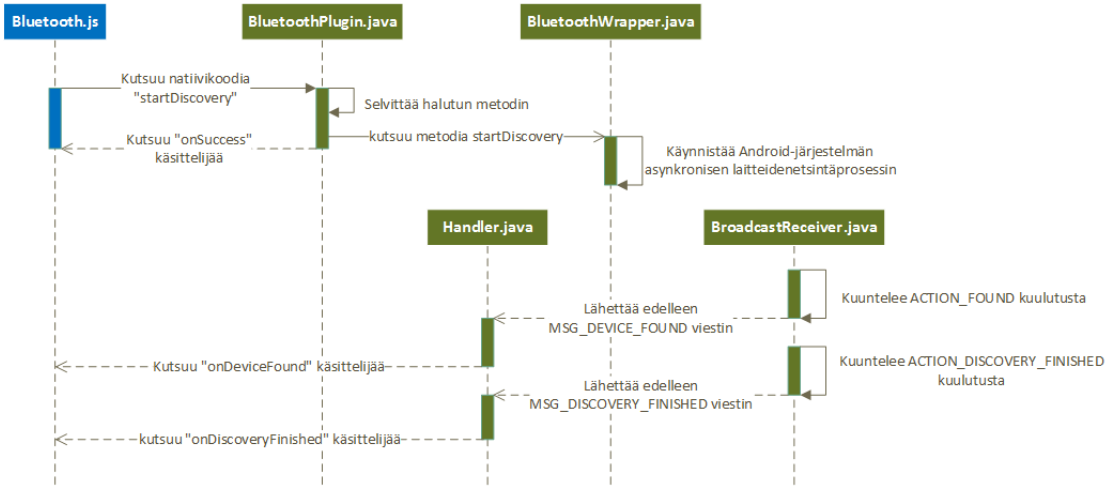
Koska Android käyttää koko järjestelmän kattavia kuulutuksia (broadcast), on Bluetooth-rajapintaa käyttävällä *BluetoothWrapper*-luokalla oma *BroadcastReceiver*-luokan instanssi. Tämän avulla *BluetoothWrapper* saa tartuttua erilaisiin Android-järjestelmän viesteihin kuten Bluetooth-laitteiden etsinnän päättymiseen. Viestin saavuttua se muunnetaan *BluetoothMessage*-enumeraation mukaiseksi ja lähetetään jälleen *BluetoothPlugin*-luokalle, joka lähettää sen jälleen JavaScript-rajapinnalle.

Alla on esitetty pieni pätkä *BluetoothWrapper*-luokan *BroadcastReceiver* implementaatiota, jossa lähetetään viesti *Handler*-luokan kautta *BluetoothPlugin*-luokalle.

```
@Override
public void onReceive(Context ctx, Intent intent) {
  String action = intent.getAction();
  if(BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action)) {
      _handler.obtainMessage(MSG_DISCOVERY_FINISHED).sendToTarget();
  }
```

BluetoothWrapper sisältää omat säieluokkansa Bluetooth-yhteyden muodostamiseen ja hallintaan, koska molemmat operaatiot ovat kyseisen säikeen estäviä. *ConnectionAttempt* pyrkii muodostamaan yhteyden laitteen kanssa ja *ConnectionManager* lukee yhteydestä tulevaa dataa sekä mahdollistaa siihen kirjoittamisen.

Kuviossa 7 on esitelty laitteiden etsintä sekvenssikaaviona. On huomioitava ettei kaaviossa oteta huomioon mahdollisia poikkeustilanteita laitteiden etsinnän kanssa.



KUVIO 7. Bluetooth-laitteiden etsintä

Sekä laitteen löytyminen että etsinnän päättyminen ovat molemmat asynkronisia tapahtumia. Tätä varten liitännäinen kuuntelee Android-järjestelmän kuulutuksia joiden kautta se saa tiedon esimerkiksi löytyneestä laitteesta ja sen tunnistetiedoista. Löydetty laite palautetaan JavaScript-rajapinnalle JSON-objektina, joka sisältää laitteen nimen ja sen Bluetooth-osoitteen.

7.2 PhoneGap sovellus

7.2.1 Yleisarkkitehtuuri

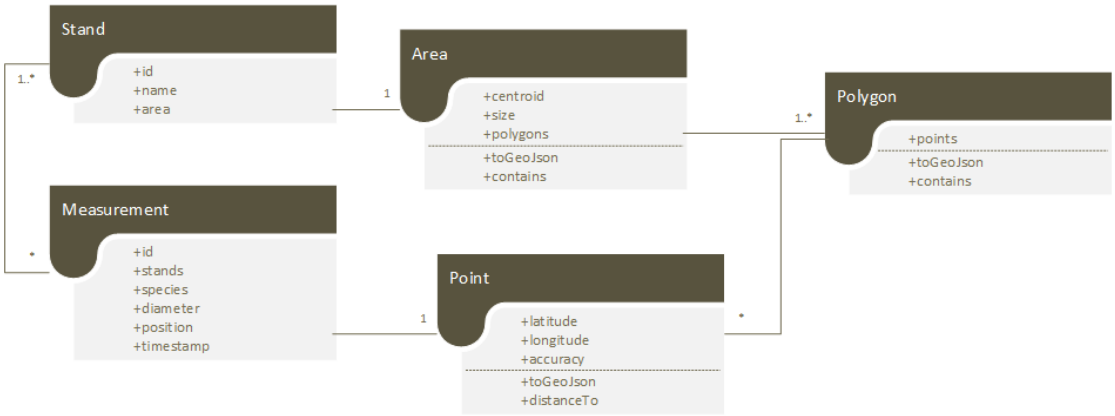
Projektin pitkän keston vuoksi oli tärkeää lähteä kehittämään sovellusta ylläpidettävyys mielessä: kehittäjiä saattaisi olla useampia projektin edetessä. Sovelluksen rakenteen tulisi olla selkeä ja yksinkertainen, joka kuitenkin mahdollistaisi laajentamisen uusien ominaisuuksien kannalta.

Kehittäjän näkökulmasta ylläpidettävyyttä haettiin valitsemalla ohjelmointikieleksi CoffeeScript, joka eliminoisi osan JavaScript-ohjelmointikielen heikkouksista. CoffeeScript-ohjelmointikieli kääntyy JavaScriptiksi, jolloin kääntäjä hoitaisi tyylipuhtaat null-varmistukset (null-check) ja estäisi globaalien muuttujien käytön vahingossa.

Toinen ylläpidettävyyden kannalta oleellinen valinta oli RequireJS-kirjaston käyttö. Perinteisesti web-sovelluksissa kaikki eri JavaScript-tiedostot listataan sovelluksen indeksi-sivulle, jolloin ne ajetaan sen sivun latautuessa. RequireJS-kirjaston avulla Ja-

vaScript-tiedostot määritetään erillisen define-funktion sisään moduuleiksi, jotka voidaan ladata asynkronisesti sovelluksen niitä tarvitessa erityisellä require-funktiolla. Tällöin sovelluksessa käytettäviä tiedostoja ei tarvitse listata ”oikeassa” järjestyksessä HTML-sivulle, vaan ne voidaan ladata yksi kerrallaan tarvittaessa.

Kuviossa 8 on esitelty sovelluksen yksi pääkomponenteista, mallit (models). Sovelluksessa käsitellään pääasiassa metsäkuvioita (*Stand*) ja mittauksia (*Measurement*). Kuviossa antaa myös jotain osviittaa mallien välisistä suhteista.



KUVIO 8. Sovelluksen mallit

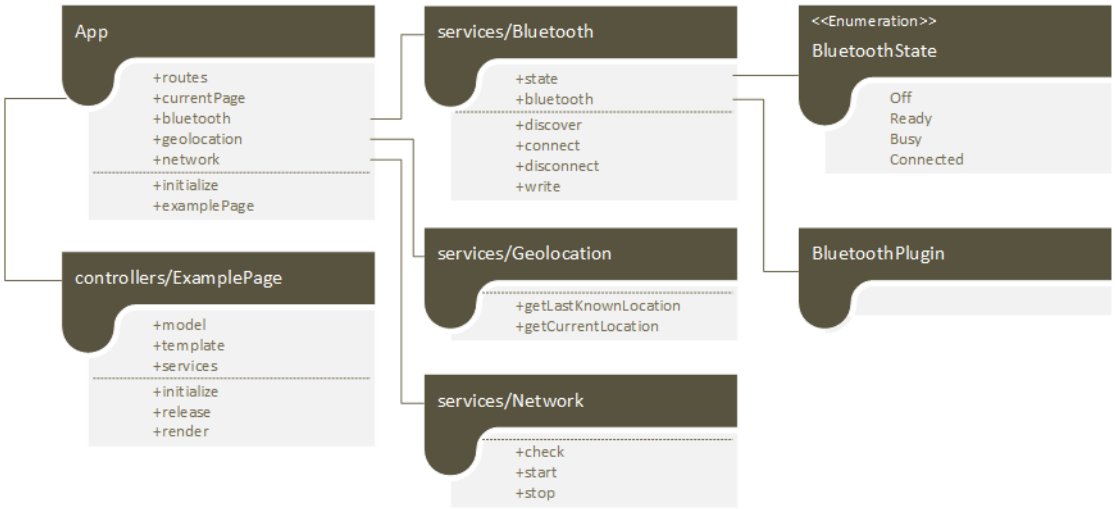
Area, *Polygon* ja *Point* ovat sovelluksen tapa kuvata erilaista maantieteellistä dataa. Ne sisältävät *toGeoJson*-metodin, joka palauttaa olion attribuutein varustellun GeoJSON-formaatin.

Sovellusta kehitettäessä tuli pian selväksi, että se tulisi sisältämään käyttäjän kannalta useita eri ”sivuja”. Sovelluksessa eri ”sivut” toteutettiin kuitenkin yksisivuisten web-sovellusten periaatteella, jossa varsinaisia HTML-sivuja ei ole kuin yksi, jonka sisältöä vaihdettaisiin riippuen sovelluksen senhetkisestä tilasta. Tämä loisi käyttäjän kannalta kokemuksen eriytetyistä sivuista. Sovelluksessa sivujen ulkoasu ja siirtymät ovat toteutettu ”jQuery Mobile”-kirjaston avulla.

Seuraavassa on kuvattu erään sivun näkymän sisältävän HTML-tiedoston lataaminen RequireJS-kirjaston avulla. Tämän jälkeen HTML-sisältö templatoidaan Handlebars-kirjaston avulla, jonka seurauksena siihen voidaan liittää relevantti data ja lisätä lopputulos DOM-rakenteeseen.

```
<!-- templates/page-device.html -->
<h2>{{device.name}}</h2>
<h3>{{device.address}}</h3>

<fieldset class="ui-grid-a">
    <div class="ui-block-a">
        <button id="connect" data-theme="c">Yhdistä</button>
    </div>
    <div class="ui-block-b">
        <button id="disconnect" data-theme="c">Katkaise Yhteys</button>
    </div>
</fieldset>

# controllers/device.coffee
template    = require "text!templates/page-device.html"
@template   = Handlebars.compile(template)

@$("[data-role='content']")
    .html(@template({ device: @model.toJSON() }))
    .trigger("create")
```

Kuviossa 9 on esitetty sovelluksen rakenne luokkakaaviona, josta on riisuttu rakenteen ymmärtämisen kannalta irrelevantit osat. Luokkakaaviossa ei ole esitelty erikseen jokaista sivua, johtuen niiden rakenteen samankaltaisuudesta.



KUVIO 9. HTML5-sovelluksen rakenne

Varsinaisen arkkitehtuurin ylimmällä tasolla on *App*-luokka, joka hoitaa myös sovelluksen sisäisen reitityksen. *App* on käytännössä Backbone-kirjaston Router-luokan instanssi, jolle on annettu lisävastuuna toimia koko sovelluksen yhteen sitovana kokonaisuutena.

Sovelluksessa usein tarvittavat ominaisuudet on erotettu itsenäisiksi moduuleiksi, jotka eivät ole riippuvaisia muista sovelluksen osista. Nämä ”palvelut” (services) suorittavat toimintoja, jotka ovat usein asynkronisia ja arvaamattomia: kuten Bluetooth-

laitteiden etsintä. Palvelut välitetään hierarkiassa alaspäin niitä tarvitseville sivuille ja komponenteille. Palvelut ovat koko sovellukselle samoja, sillä uusia instansseja niistä ei sovelluksessa luoda muualla kuin *App*-luokan ainoassa instanssissa.

Johtuen sovelluksen usein asynkronisesta ja ennustamattomasta luonteesta on tärkeää, että sovelluksen eri osat saavat kiinni eri tapahtumista kätevästi. Siksi palvelut peritään *Events*-luokasta, joka mahdollistaa Backbone-kirjaston Model- ja Collection-luokista tutut *on*, *off* ja *trigger* –metodit. Alla on esitelty *Events*-luokan toteutus kokonaisuudessaan.

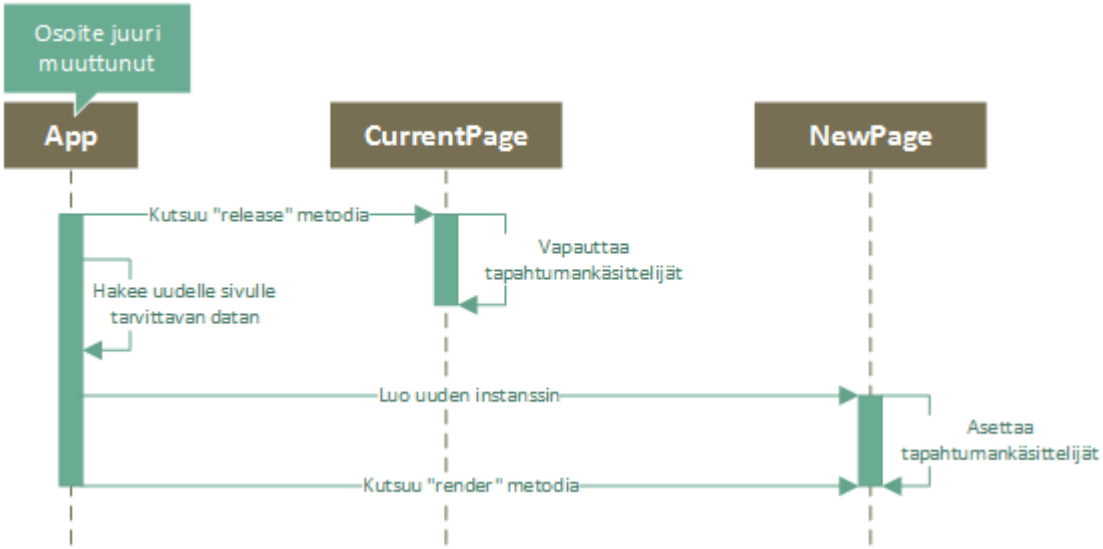
```
class Events
    listeners: {}

    on: (name, cbk, ctx) ->
        if not _.isArray(@listeners[name])
            @listeners[name] = []
            @listeners[name].push({ callback: cbk, context: ctx })

    off: (name, cbk) ->
        if name? and cbk?
            if _.isArray(@listeners[name])
                @listeners[name] = (listener for listener in @listeners[name]
                    when listener.callback is not cbk)
        else if name?
            @listeners[name] = []
        else
            @listeners = {}

    trigger: (name, args = {}) ->
        if _.isArray(@listeners[name])
            _.each @listeners[name], (listener) ->
                if _.isFunction(listener.callback)
                    if listener.context?
                        cbk = _.bind(listener.callback, listener.context)
                    else
                        cbk = listener.callback
                    cbk(args)
```

Sovelluksessa sivut koostuvat MV*-arkkitehtuurin mukaisesti niihin kiinnitetyistä malleista (Model), HTML-merkintäkielellä määritetystä näkymästä (View) ja sivun kontrollerista (Controller). Sivun näkymä ladataan RequireJS-kirjaston text-liitännäisellä, joka mahdollistaa esimerkiksi juurikin HTML-tiedostojen asynkronisen latauksen. Tällöin sivulle relevantti HTML-koodi voidaan eriyttää sovelluksessa omaksi tiedostokseen, pitäen näin sovelluksen rakenteen siistinä.

Seuraavana ovat listattu kirjoitushetkellä sovelluksessa olleet sivut ja niiden vastuualueet:

- **HomePage:** sovelluksen kotisivu, joka sisältää tarvittavat linkit sovelluksessa navigoimiseen.
- **MeasurementPage:** näyttää kartalla käyttäjän senhetkisen sijainnin ja lähellä olevien metsäkuvioiden rajat. Sivu näyttää myös mittaukset sille metsäkuviolle, jolle sovellus paikantaa käyttäjän.
- **StandListPage:** listaa kaikki tiedetyt metsäkuviot ja listaa ne linkeiksi *StandPage*:lle.
- **StandPage:** näyttää kartalla valitun metsäkuvion rajat ja sille kuuluvat mittaukset. Sisältää myös tarkempaa tietoa metsäkuviosta kuin *MeasurementPage.*
- **DeviceListPage:** listaa löydetyt Bluetooth-laitteet ja linkittää *DevicePage*:lle. Mahdollistaa Bluetooth-laitteiden etsimisen.
- **DevicePage:** näyttää valitun Bluetooth-laitteen tilan. Antaa käyttäjän yhdistää ja katkaista yhteyden laitteeseen.
- **OptionsPage:** listaa sovelluksessa mahdolliset asetukset, kuten TMS:n (Tile Map Service) osoitteen.

Kuviossa 10 on kuvattu sivujen välillä siirtymisen prosessi sekvenssikaaviona. Kaaviossa näytetyt sivut ovat esimerkkejä, mutta prosessi on sama myös yllä listatuilla sivuilla.



KUVIO 10. Sivujen välillä siirtyminen

7.2.2 Mittaaminen

Sovelluksen avulla mittaaminen toimii käytännössä niin, että mittaaja syöttää sovellukseen läpimitan ja lajikkeen joko käsin tai Bluetooth yhteyden kautta (esim. Masser BT Caliper). Sovellus tarkastaa käyttäjän syötteet oikeellisiksi ja liittää mobiililaitteen kautta saatavan GPS-paikkatiedon niihin, jonka jälkeen tästä saatava *Measurement*-mallin mukainen tieto lähetetään palvelimelle.

BluetoothPlugin-liitännäisen rajapintaa varten luotiin abstraktiokerros Bluetooth-palvelun muodossa. Bluetooth-palvelu pitää sisällään vain mittaus-sovellusta varten tarvittavan rajapinnan Bluetooth-toiminnoille: esimerkiksi laitteiden paritus on jätetty palvelusta pois kokonaan. Palvelu peritään aiemmin esitellystä *Events*-luokasta, jonka avulla on toteutettu palvelulle kuviossa 9 esitellyt eri tilat (*BluetoothState*).

Bluetooth-palvelu sisältää kokoelman *BluetoothDevice*-malleja, joilla on jokaisella nimi, osoite ja yhteyden tila. Nämä mallit kuvastavat jokainen yhtä Bluetooth-palvelun avulla löydettyä Bluetooth-laitetta ja sitä, onko sovellus niiden kanssa yhteydessä. BluetoothDevice-mallit ovat Backbone-kirjaston Model-luokan implementaatioita, joten on mahdollista kiinnittää kuuntelijoita esimerkiksi laitteen yhteyden tilan vaihtumiseen.

Seuraavassa on esitelty lyhyesti ja yksinkertaistetusti hieman Bluetooth-palvelun luokkamäärittelyä.

```
class Bluetooth extends Events

    _bluetooth = window.plugins.bluetooth

    state: null

    states:
        Off: 0
        Ready: 1
        Busy: 2
        Connected: 3

    constructor: () ->
        @devices = new BluetoothDevice.Collection
        _bluetooth.isEnabled (isEnabled) =>
            if isEnabled
                if @state is @states.Off then @set(@states.Ready)
            else if not isEnabled then @set(@states.Off)

    set: (state) ->
        if @state != state
            @state = state
            @trigger("change")
```

Bluetoothin välityksellä saapuneista viesteistä ilmoitetaan *messagereceived*-tapahtumalla, johon tarttumalla käsittelijä saa tietää sekä viestin sisällön että laitteen mikä sen lähetti. Viestit ovat NMEA-formaattia, jossa viestin perässä tulee viestin valmiiksi laskettu tarkastusarvo. Tarkastusarvon avulla viestin sisältö voidaan tarkastaa oikeaksi, joka on tärkeää varsinkin, kun Bluetooth-viesteillä on tapana tulla useampana osana yhden merkkijonon sijaan. Johtuen viestien tulemisesta useammassa osassa, on sovelluksessa erillinen *parser*-luokka. Parser voi palauttaa sitä hallinnoivalle luokalle esimerkiksi *incomplete*-tilan kun viestistä arvioidaan puuttuvan osia.

Seuraavana on esitelty kuinka sovelluksen vastaanottama viesti validoidaan ja sitten muunnetaan merkkijonosta JavaScript-objektiksi.

```
_validate = (msg) ->
    startIndicators = msg.split("$").length - 1
    checksumSeparators = msg.split("*").length - 1

    if startIndicators > checksumSeparators
        return MessageState.Incomplete
    else if checksumSeparators > startIndicators
        return MessageState.Invalid

    data = msg.split(/[\$\*]/)
    data.shift()

    actualMessage = data[0]
    impliedChecksum = parseInt(data[1], 16)

    checksum = 0
    for i in [0...actualMessage.length]
        checksum = checksum ^ actualMessage.charCodeAt(i)

    if checksum isnt impliedChecksum
        return MessageState.Invalid
    else return MessageState.Complete

_parse = (msg) ->
    data = msg.split(/[\$\*]/)
    data.shift()

    actualMessage = data[0]
    messageParts = actualMessage.split(",")

    results = ({ field: actualMessage[i], value: actualMessage[i + 1] }
        for i in [0...messageParts.length] when i % 2 is 0)

    species = _.find(results, (result) -> result.field == "SPECIES")
    diameter = _.find(results, (result) -> result.field == "DIAMETER")
    quality  = _.find(results, (result) -> result.field == "QUALITY")

    return { species: species, diameter: diameter, quality: quality }
```

Sovelluksessa Parser tallentaa sille annetut viestit, ja yhdistää ne aina edelliseen kunnes viestin validointi onnistuu ja se voidaan parsia JavaScript-objekteiksi. Nämä objektit palautetaan yhdessä tilan kanssa Parseria hallinnoivalle luokalle.

7.2.3 Geolokaatio

PhoneGap sovelluksessa HTML5 geolokaatio-rajapinta on ylikirjoitettu käyttämään laitteen omaa GPS-sensoria. Rajapinta voi kuitenkin mahdollisesti palauttaa sijainnin käyttämällä verkkojen avulla paikannusta, joka on huomattavasti epätarkempaa verrattuna satelliittipaikannukseen. Android-alustoilla täytyy rajapinnalle kertoa erikseen haluavansa korkean tarkkuuden paikkatietoa.

Sovelluksessa paikkatieto vaadittiin käyttäjän suorittamille mittauksille. Paikkatietoon kuuluu WGS84-koordinaattijärjestelmän mukaiset x- ja y-koordinaatit sekä arvioitu virhe metreissä. Paikkatiedon virheen ylärajaksi sovelluksessa määritettiin 30 metriä. Sovelluksen päällä ollessa paikkatietoa kerätään jatkuvasti, jotta viimeisimmän paikkatiedon hakeminen olisi mahdollisimman nopeaa, eikä erillistä kutsua paikannuksen rajapinnalle tarvittaisi. Sovelluksessa HTML5 paikannuksen rajapinta enkapsuloitiin erilliseksi palveluksi, joka seuraa ja tallentaa n-määrän viimeisimpiä tietoja laitteen sijainnista. Palvelu hylkää liian suuren virheen omaavat paikkatiedot tallennetuista tiedoista.

Kuvio- ja mittaus-tiedon paikkatietojen käsittelemiseksi luotiin kuviossa 8 nähdyt *Area*-, *Point*- ja *Polygon*-mallit. Näiden mallien lisäksi luotiin myös erillinen vektori-luokka, joka sisälsi vektoreille ominaisia laskutoimituksia. Vektori-luokkaa käytettiin hyväksi laskiessa esimerkiksi mille kuviolle (jos millekään) mitattu piste kuuluu.

Seuraavassa on esitetty pieni pätkä vektori-luokan toteutuksesta.

```
class Vector2D

    add: (that) ->
        return new Vector2D({ x: @x + that.x, y: @y + that.y })

    multiply: (scalar) ->
        return new Vector2D({ x: @x * scalar, y: @y * scalar })

    magnitude: () ->
        return Math.sqrt((@x * @x) + (@y * @y))

    normalize: () ->
        mag = @magnitude()
        return new Vector2D({ x: @x / mag, y: @y / mag })
```

```
    dot: (that) ->
        return (@x * that.x) + (@y * that.y)
```

Mittaustulosten sijoittaminen oikeille kuvioille tapahtuu ajattelemalla mittauksen sijaintia ympyränä, jonka säteen paikkatiedon virhe määrittää. Sovelluksessa tarkastellaan meneekö tämä ympyrä päällekkäin minkään tiedetyn kuvion kanssa; tästä seuraa se, että mittaustulos voi olla useammalla kuviolla. Lähdekoodissa tämä on toteutettu ensin katsomalla, onko mittauksen paikkatiedon keskipiste kuvion sisällä ja tämän jälkeen katsomalla, leikkaako mittauksen paikkatiedon muodostama ympyrä kuvion yhtäkään sivua.

Seuraavassa on esitelty kuinka sovelluksessa on toteutettu viivan ja ympyrän välisen leikkauksen etsintä. Koodissa C on ympyrän keskipiste, jolla on myös *acc*-attribuutissa ympyrän säde. A- ja B-pisteet ovat kuvion sivun alku- ja loppu-pisteet.

```
C = point
for i in [0...@points.length]
    A = @points[i]
    B = if (i + 1) < @points.length then @points[i + 1] else @points[0]
    AB = new Vector2D(A, B)
    AC = new Vector2D(A, C)

    scalar = AC.dot(AB.normalize())

    if scalar < 0
        if C.acc >= A.distanceTo(C)
            return true
    else if scalar > AB.magnitude()
        if C.acc >= B.distanceTo(C)
            return true
    else
        AD   = AB.normalize().multiply(scalar)
        AAD = new Vector2D(A).add(AD)
        D    = new Point({ lat: AAD.x, lng: AAD.y })

        distanceAC = A.distanceTo(C)
        distanceAD = A.distanceTo(D)
        distanceCD = Math.sqrt(
            (distanceAC * distanceAC) - (distanceAD * distanceAD)
        )

        if C.acc >= distanceCD
            return true
```

Käyttäjän sijaintia sovelluksessa voi seurata kartalta, joka näyttää sovelluksen sivusta riippuen esimerkiksi valitun kuvion rajat. Kartta on toteutettu sovelluksessa *Leaflet*-kirjastolla, joka mahdollistaa TMS:n (Tile Map Service) kytkemisen helposti ja näyttävästi HTML5-sovellukseen. Sovelluksessa kartalle luotiin oma luokkansa, joka sisältää tarvittavat metodit Leaflet-pohjaisen kartan luomiseen. Kartta kuuntelee muutoksia

laitteen sijainnissa ja näyttää sijainnin jatkuvasti sinisenä ympyränä, jossa säde määräytyy senhetkisen paikkatiedon virheen mukaan.

Seuraavaksi on esitelty pätkä kartan lähdekoodia, jossa nähdään, kuinka kartta piirretään DOM-rakenteeseen sille varattuun paikkaan.

```
class Map

    render: () ->
        console.log("Map.render")

        if @map is not null then @map.remove()
        @map = L.map($(@el)[0], { attributionControl: false })

        loc = @geolocation.getLastKnownLocation()

        @map.setView(loc.toLatLngArray(), 13)

        @tilemap.addTo(@map)
        d.addTo(@map) for d in @data
        @map.addLayer(@location)

        setTimeout(_.bind(@map.invalidateSize, @map), 2000)
```

7.2.4 Tiedonsiirto

Opinnäytetyössä luotu sovellus keskustelee opinnäytetyön ulkopuolelle jääneen palvelimen kanssa REST-rajapinnan välityksellä. Palvelin tarjoaa rajapinnat kuvioille (*Stand*) ja mittauksille (*Measurement*), joiden kautta voidaan hakea joko kaikki tai yksittäinen malli. Palvelimelta tuleva data on JSON muotoista, joka muunnetaan asiakasohjelman puolella kuvion 8 mukaisiksi malleiksi.

Backbone-kirjaston malleille määritetään sille varatun rajapinnan osoite. Alla on esitetty kuinka tällaiset REST-arkkitehtuurin mukaiset osoitteet toimivat käytännössä.

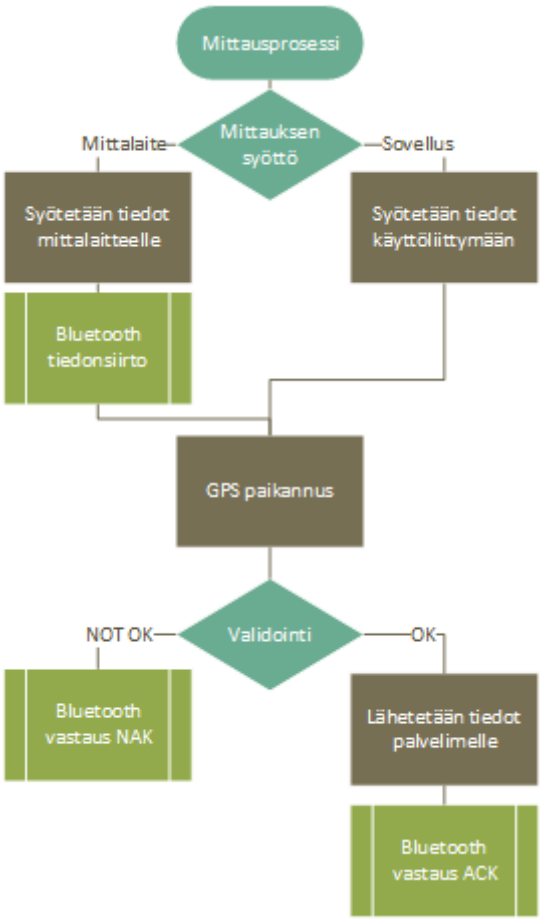
```
Measurement = Backbone.Model.extend
    urlRoot: "#{globals.api}measurements/"

Measurements = Backbone.Collection.extend
    model: Measurement

# Pyyntö http://esimerkki.labranet.jamk.fi/api/measurements/5
new Measurement({ id: 5 }).fetch()

# Pyyntö http://esimerkki.labranet.jamk.fi/api/measurements/
new Measurements().fetch()
```

Edellä olevasta esimerkistä nähdään, kuinka Backbonessa mallin rajapinnan osoite määräytyy sille asetetun *id*-attribuutin mukaan. Kokoelmaa haettaessa osoitteeseen

ei määritetä *id*-attribuuttia, jolloin tuloksena on JSON-muotoinen taulukko, joka koostuu yksittäisistä malleista.

Sovelluksen lähdekoodista löytyy palvelu myös laitteen verkon tilan tarkkailulle. Verkon tilaa tarkkaillaan sovelluksessa lähinnä kokeellisen välimuistiominaisuuden vuoksi. Tämän ominaisuuden olisi tarkoitus mahdollistaa HTTP-pyyntöjen tulosten tallentaminen lokaalisti, jotta sovellus voi vaikuttaa toimivan ilman verkkoyhteyttä. Tätä varten Backbone-kirjaston kylkeen liimattiin *DualStorage*-liitännäinen, joka ”*cachettaa”* Backbone-kirjaston kautta kulkevan liikenteen. Kaikki Backbone-kirjaston avulla tapahtuva REST-rajapinnan kanssa keskustelu tapahtuu niin sanotun *Sync*-metodin kautta, jonka DualStorage-liitännäinen muuntaa käyttämään HTML5 *Local Storagea*. Valitettavasti sovellusta kehitettäessä tämä ominaisuus jäi taka-alalle, eikä sitä testattu erityisen tarkasti. Tästä johtuen ominaisuuden toiminta jäi hieman mysteeriksi: välillä DualStorage-liitännäinen vaikutti toimivan moitteettomasti, mutta osan ajasta pyynnöt eivät tallentuneet laitteen välimuistiin tai ne unohdettiin kokonaan.

Sovelluksessa keskeisin tiedonsiirto tapahtuu Bluetooth-laitteelta palvelimelle. Tätä varten sovellus validoi laitteelta tulevan datan ja sijoittaa sen valmiiksi kuviolle, mutta myös palvelin suorittaa tuloksille validoinnin ennen kuin ne tallennetaan tietokantaan. Kuviossa 11 on pyritty visualisoimaan sovelluksessa tapahtuva tiedonsiirtoprosessi mittaustuloksien suhteen hyvin yksinkertaisella tavalla.



KUVIO 11. Sovelluksen mittausprosessi

8 TYÖN TULOKSET

Lopputuloksena saatiin PhoneGap kehyksen päälle toteutettu mobiilisovellus, joka testattiin toimivaksi myös Windows Phone 8 alustalla. Sovellus toteutti projektin aluksi listatut tavoitteet kuten tiedonsiirron Bluetooth-laitteen ja HTML5-mobiilisovelluksen välillä. Sovelluksella on mahdollista suorittaa maastomittauksia, joista kerätyt tiedot lähetetään palvelinsovellukselle. Sovellus hakee palvelimelta kerätyn datan ja esittää sen käyttäjälle HTML5-tekniikoin toteutetulla kartalla. Kerätystä datasta lasketaan myös muita arvoja, joka esitetään käyttäjälle esimerkiksi kaavioin ja taulukoin.

HTML5-sovelluksen yksi tavoite oli olla mahdollisimman helppo jatkokehitettävä. Selkeä arkkitehtuuri ja suhteellisen helposti ymmärrettävien kirjastojen käyttö sallii projektin osakkaiden suorittaa sovellukselle haluttua jatkokehitystä ilman kummempia

vaikeuksia. Sovelluksen ohjelmakoodi on kirjoitettu ja kommentoitu selkeästi ja hyvien tapojen mukaisesti niin, että myös hieman kokemattomampi ohjelmoija pääsee sisälle ohjelman rakenteeseen.

Projektin aluksi luotu Bluetooth-liitännäinen PhoneGap-ohjelmointikehykselle osoittautui varsin onnistuneeksi. Se julkaistiin GitHub-palvelussa projektin lähestyessä loppuaan ja se on kerännyt jopa hieman huomiota projektin ulkopuolisilta toimijoilta. Projektin aikana liitännäistä esiteltiin eri tahoille, jotka osoittivat myös kiinnostusta sitä kohtaan. Liitännäisen käytöstä kirjoitettiin myös blogimerkintä JAMK:n blogipalveluun, joka myöhemmin linkitettiin PhoneGap-yhteisön viralliseen artikkelilistaukseen.

9 POHDINTA

Projektin alussa kävi ilmi, että varsinaista vaatimusmäärittelyä sovellukselle ei ollut, eikä projektissa noudatettu mitään perinteisiä ketteriä menetelmiä. Sovellukseen halutut ominaisuudet tulivat ilmi enemmän tai vähemmän vapaamuotoisen keskustelun lomassa. Tästä johtuen sovellusta kehitettäessä täytyi usein tehdä pientä refaktorointia eri komponenttien osalta, kun vaatimukset saattoivat muuttua iltapäiväkahvin aikana.

Onneksi sovellusta kehitettäessä oli kokoajan olemassa jonkinlainen visio, mitä osia sovellus sisältäisi. Tämä auttoi suunnittelemaan sovellukselle helposti laajennettavan arkkitehtuurin, joka takaisi sen, ettei itse pääasiallista arkkitehtuuria tarvitsisi refaktoroida vaikka jotkin vaatimukset muuttuisivat. Myös valmiiden käyttöliittymäkehysten käyttö helpotti kehitystyötä huomattavasti johtuen niiden vaivattomuudesta. Projektia auttoi myös siirtyminen työnhallinnan saralla kanban-pohjaisen taulun käyttöön, joskaan sitä ei mielestäni käytetty aina tarpeeksi ja oikealla tavoin.

Sovelluksen kehitysprosessi oli hyvin opettava kokemus. Vaikka HTML5-tekniikat olivat itselleni entuudestaan tuttuja, oli niiden soveltaminen mobiiliympäristössä varsin tervetullut taito. Vaikka sovellus ei ole mitenkään erityisen laaja tällä hetkellä, on sen tekemiseen mennyt kuitenkin aikaa huomattava määrä johtuen oikeiden kirjastojen ja ohjelmointikehyksien etsimisestä ja testaamisesta. Esimerkiksi Windows Phone 8

alustalla sovellusta ei saatu toimimaan ennen kuin AngularJS-ohjelmointikehyksen käytöstä luovuttiin ja siirryttiin perinteisempään Backbone-ohjelmointikehykseen.

Teknisistä ratkaisuista on CoffeeScript-ohjelmointikielen käyttö maininnan arvoinen. Mielestäni se on looginen jatke JavaScript-ohjelmointikielelle ja se tekee ohjelmakoodista huomattavasti luettavampaa. Sen avulla voidaan välttää myös monia JavaScript-ohjelmoinnin sudenkuoppia ja tätä kautta auttaa kehittäjää olevaan huomattavasti enemmän tuottavampi. Toinen mielestäni hyvä tekninen ratkaisu oli RequireJS-kirjaston käyttöönotto. JavaScript-tiedostojen lataus suoraan koodista siistii ohjelmaa huomattavasti. Se myös pakottaa kehittäjän jakamaan koodiaan useammaksi tiedostoksi tai niin sanotuiksi moduuleiksi.

Sovellus kehittyy jatkuvasti. Jatkon kannalta tuki useammalle laitteelle on mielestäni tärkeä. Windows Phone 8 alustalla testaamisesta ja kehityksestä onkin jo jonkin aikaa vastannut Samppa Tiihonen, joka mm. kirjoitti Bluetooth-liitännäiselle Windows Phone 8 toteutuksen. Myös käännöksestä Applen iOS-alustalle on keskusteltu.

Ominaisuuksien saralla sovellus on edennyt positiiviseen suuntaan. Valitettavasti nämä uudet ominaisuudet ovat rajatut tämän opinnäytetyön ulkopuolelle. Tulevaisuudessa olisi tärkeää keskittyä varsinkin aluskasvuston määrän selvittämiseen ja mittaustavan kehittämiseen. Nämä ominaisuudet ovatkin jo saaneet hieman tuulta alleen aikaisten versioiden muodossa ja niitä on esitelty projektin osakkaille onnistuneesti.

LÄHTEET

AJAX – Asynchronous JavaScript and XML. 2008. Ohjelmointiputka. Viitattu 10.8.2013. http://www.ohjelmointiputka.net/oppaat/opas.php?tunnus=ajax

Ajax (programming). 2013 . Wikipedia – vapaa tietosanakirja. Viitattu 10.8.2013. http://en.wikipedia.org/wiki/Ajax_(programming)

Android (operating system). 2013. Wikipedia – vapaa tietosanakirja. Viitattu 19.7.2013. http://en.wikipedia.org/wiki/Android_(operating_system)

Android software development. 2013. Wikipedia – vapaa tietosanakirja. Viitattu 19.7.2013. http://en.wikipedia.org/wiki/Android_software_development

AngularJS. 2013. Wikipedia – vapaa tietosanakirja. Viitattu 16.10.2013. http://en.wikipedia.org/wiki/AngularJS

Application Fundamentals. N.d. Android Developers. Viitattu 19.7.2013. http://developer.android.com/guide/components/fundamentals.html

Avola, G. 2012. Creating apps with PhoneGap: Lessons learned. Viitattu 19.7.2013. http://www.adobe.com/devnet/phonegap/articles/creating-apps-with-phonegap-lessons.html

Backbone.js. 2013. Wikipedia – vapaa tietosanakirja. Viitattu 7.10.2013. http://en.wikipedia.org/wiki/Backbonejs

Bailey, D. 2011. Backbone.js Is Not An MVC Framework. Viitattu 7.10.2013. http://lostechies.com/derickbailey/2011/12/23/backbone-js-is-not-an-mvc-framework/

Bluetooth – Android Developers. N. d. Android Developers. Viitattu 22.7.2013. http://developer.android.com/guide/topics/connectivity/bluetooth.html

Bluetooth. 2013. Wikipedia – vapaa tietosanakirja. Viitattu 22.7.2013. http://en.wikipedia.org/wiki/Bluetooth

CoffeeScript. 2013. Wikipedia – vapaa tietosanakirja. Viitattu 16.9.2013. http://en.wikipedia.org/wiki/CoffeeScript

Dalvik (software). 2013. Wikipedia – vapaa tietosanakirja. Viitattu 19.7.2013. http://en.wikipedia.org/wiki/Dalvik_(software)

Florence, R. 2011. A Case Against Using CoffeeScript. Viitattu 16.9.2013. http://ryanflorence.com/2011/case-against-coffeescript/

Foundation (framework). 2013. Wikipedia – vapaa tietosanakirja. Viitattu 11.9.2013. http://en.wikipedia.org/wiki/Foundation_(framework)

Glonass. 2013. Wikipedia – vapaa tietosanakirja. Viitattu 26.10.2013. http://fi.wikipedia.org/wiki/Glonass

GPS. 2013. Wikipedia – vapaa tietosanakirja. Viitattu 26.10.2013. http://fi.wikipedia.org/wiki/GPS

How Ajax Works. 2008. Web Designer Depot. Viitattu 10.8.2013. http://www.webdesignerdepot.com/2008/11/how-ajax-works/

Johnson, J. 2013. Mobile First Design: Why It's Great and Why It Sucks. Viitattu 22.7.2013. http://designshack.net/articles/css/mobilefirst/

jQuery Mobile. 2013. Wikipedia – vapaa tietosanakirja. Viitattu 11.9.2013. http://en.wikipedia.org/wiki/JQuery_Mobile

jQuery. 2013. Wikipedia – vapaa tietosanakirja. Viitattu 11.9.2013. http://en.wikipedia.org/wiki/JQuery

Koordinaattijärjestelmä. 2013. Wikipedia – vapaa tietosanakirja. Viitattu 26.10.2013. http://fi.wikipedia.org/wiki/Koordinaattij%C3%A4rjestelm%C3%A4

Model-view-controller. 2013. Wikipedia – vapaa tietosanakirja. Viitattu 5.10.2013. http://en.wikipedia.org/wiki/Model-view-controller

Osmani, A. 2012. Journey Through The JavaScript MVC Jungle. Viitattu 5.10.2013. http://coding.smashingmagazine.com/2012/07/27/journey-through-the-javascript-mvc-jungle/

PhoneGap. 2013. Wikipedia – vapaa tietosanakirja. Viitattu 19.7.2013. http://en.wikipedia.org/wiki/PhoneGap

Pilgrim, M. N. d. Dive Into HTML5. Viitattu 11.8.2013. http://diveintohtml5.info/index.html

Plugin Development Guide. N. d. PhoneGap Documentation (2.6.0). Viitattu 22.7.2013. http://docs.phonegap.com/en/2.6.0/guide_plugin-development_index.md.html

Rand-Hendriksen, M. 2013. Responsive design for all screens. Viitattu 22.7.2013. http://blog.lynda.com/2013/03/21/responsive-design-for-all-screens/

Ranganathan, A. 2010. Beyond HTML5: Database APIs and the Road to IndexedDB. Viitattu 11.8.2013. http://hacks.mozilla.org/2010/06/beyond-html5-database-apis-and-the-road-to-indexeddb/

Ranganathan, A., Wilsher, S. 2010. Firefox 4: An early walk-through of IndexedDB. Viitattu 11.8.2013. https://hacks.mozilla.org/2010/06/comparing-indexeddb-and-web-database/

Responsive web design. 2013. Wikipedia – vapaa tietosanakirja. Viitattu 22.7.2013. http://en.wikipedia.org/wiki/Responsive_web_design

Single-page application. 2013. Wikipedia – vapaa tietosanakirja. Viitattu 10.8.2013. http://en.wikipedia.org/wiki/Single-page_application

Takada, M. N. d. Single page apps in depth. Viitattu 10.8.2013. http://singlepageappbook.com/

Trestima Oy. 2013. Viitattu 18.7.2013. https://www.trestima.com/

Twitter Bootstrap. 2013. Wikipedia – vapaa tietosanakirja. Viitattu 11.9.2013. http://en.wikipedia.org/wiki/Twitter_Bootstrap

Unobtrusive JavaScript. 2013. Wikipedia – vapaa tietosanakirja. Viitattu 22.7.2013. http://en.wikipedia.org/wiki/Unobtrusive_JavaScript

Vastaranta, M., Holopainen, M., Kaartinen, H. & Hyyppä, H. & Hyyppä J. 2009. Uudistuneet metsien maastomittaustarpeet. Metsätieteen aikakausikirja 4/2009. Saatavissa: http://www.metla.fi/aikakauskirja/full/ff09/ff094370.pdf

WGS84. 2013. Wikipedia – vapaa tietosanakirja. Viitattu 26.10.2013. http://fi.wikipedia.org/wiki/WGS84