Haaga-Helia
ammattikorkeakoulu Oy

Ilmatieteen laitoksen havaintoasemien huolto- ja ylläpito-toiminnan sähköisen raportoinnin kehittäminen

Jared Mäenpää

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2017

Haaga-Helia ammattikorkeakoulu Oy

Tiivistelmä

15.05.2017

Tekijä(t)
Jared Mäenpää.

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Tutkimussuunnitelmanotsikko
Ilmatieteen laitoksen havaintoasemien huolto- ja ylläpitotoiminnan sähköisen raportoinnin kehittäminen

Sivu- ja liitesivumäärä
38 + 2

Opinnäytetyössä tutkitaan, miten Ilmatieteen laitoksen havaintoasemien huolto- ja ylläpitotoiminnan sähköistä raportointia voitaisiin kehittää. Kehitystyössä hyödynnetään moderneja verkkotekniikoita selaimessa sekä sovelluksena useilla eri alustoilla.

Ilmatieteen laitoksen asemanhuolto- sekä edustavuusarviointiraportointi toteutetaan täyttämällä Excel-lomake, joka tulostetaan PDF-muotoon ja tallennetaan tiedostojärjestelmään. Tämä koetaan hankalaksi käyttää ja tietojen hyödyntäminen on hankalaa.

Toimeksiantajan tarpeita selvitetään tarvekartoituksella, joka toteutetaan kumpaakin raporttia koskien omana haastatteluna. Tuloksena saadaan selville tärkeimmät kehittämishaasteet, kuten Excel-lomakkeen käytön hankaluus sekä edustavuusarviointiraportissa valokuvien tallennus.

Sovelluksen prototyypin kehittämisessä hyödynnetään mm. seuraavia teknologioita; Node.js, Cordova, Bootstrap, jQuery, Vue. Työvälineinä käytetään MacOS- sekä Windows 10-käyttöjärjestelmiä sekä Visual Studio Code-editoria.

Havaintopalvelujen raportointisovelluksen prototyyppiä lähdetään kehittämään kahdessa osassa. Ensimmäisenä suunnitellaan sovelluksen SQLite-tietokantarakenne ja lisäksi otetaan kantaa rajapinnan PostgreSQL-tietokantarakenteeseen. Seuraavaksi suunnitellaan käyttöliittymä ja lisätään Cordova- ja Vue.js-toiminnallisuudet.

Opinnäytetyön tuloksena saadaan prototyyppi, joka havainnollistaa, miten uusi raportointisovellus voisi toimia ja mitä ominaisuuksia se sisältäisi.

Asiasanat
Selainsovelluskehitys, käytettävyys, tietovarasto

Sisällys

Termit ja lyhenteet

AJAX	Lyhenne englanninkielisestä termistä Asynchronous JavaScript And XML.
API	Lyhenne englanninkielisestä termistä Application Programming Interface, eli rajapinta, jota kutsumalla saadaan määritelty tulos.
CSS	Lyhenne englanninkielisestä termistä Cascading Style Sheets, jonka avulla HTML-elementit muotoillaan selaimessa.
DOM	Lyhenne englanninkielisestä termistä Document Object Model, jonka selain luo HTML-merkintäkielen perusteella.
EAV	Lyhenne englanninkielisestä termistä Entity Attribute Value, malli, jolla tietokantaan pystytään luomaan dynaamisia tietueita.
HAV	Havaintoasemien sisäinen hallintaverkkosivusto.
HTML	Lyhenne englanninkielisestä termistä Hypertext Markup Language, jota käytetään verkkosivujen kirjoittamisessa.
JavaScript	Ohjelmointikieli, joka suoritetaan tavallisemmin selaimessa ja jonka avulla pystytään manipuloimaan HTML/CSS-määritelmiä dynaamisesti.
JSON	Lyhenne englanninkielisestä termistä JavaScript Object Notation, joka on standardi tiedostomuoto.
NoSQL	Lyhenne englanninkielisestä termistä Not only SQL, eli perinteisestä relaatiomallista poikkeava tietokanta.
PHP	Lyhenne englanninkielisestä termistä PHP: Hypertext Preprocessor, ohjelmointikieli, jota suoritetaan palvelimella.

RDBMS	Lyhenne englanninkielisistä sanoista Relational Database Management System, eli relaatiotietokannan hallintajärjestelmä.
REST	Lyhenne englanninkielisestä termistä Representational State Transfer, joka on arkkitehtuurimalli.
RIA	Lyhenne englanninkielisestä termistä Rich Internet Application, joka tarkoittaa dynaamista verkkosivustoa.
SPA	Lyhenne englanninkielisestä termistä Single Page Application, tekniikka, jolla ohjelmoidaan työpöytäsovelluksen tavoin toimiva sovellus.
UI	Lyhenne englanninkielisestä termistä User Interface, joka tarkoittaa käyttöliittymää.
W3C	Lyhenne englanninkielisestä nimestä World Wide Web Consortium, joka on organisaatio joka vastaa HTML ja CSS standarteista.
XML	Lyhenne englanninkielisestä termistä Extensible Markup Language, joka on merkintäkieli rakenteellisen tiedon määrittelyyn.

1 Johdanto

Ilmatieteen laitos, jonka päätehtävänä on tuottaa sääpalveluita valtion ja kansalaisten käyttöön on perustettu vuonna 1838. Ilmatieteen laitos kuuluu liikenne- ja viestintäministeriön alaisuuteen ja sen toimintaa ohjaa Suomen laki. (Ilmatieteen laitos, 2016.)

Ilmatieteen laitoksen havaintoverkkoon kuuluu noin 400 havaintoasemaa. Jokaisella havaintoasemalla on oma tehtävänsä havaintoasemaverkostossa ja osa mittaakin esimerkiksi vain magneettisuutta ja osa puolestaan mittaa useaa eri suuretta kuten lämpötilaa, sateenmäärää ja pilvisyyttä. Havaintoverkostoa ylläpitää Ilmatieteen laitoksen havaintopalvelut-yksikkö. (Ilmatieteen laitos, 2016.)

Havaintopalvelut-yksikön, joka kuuluu sää- ja turvallisuus-tulosalueen alle, tehtäviin kuuluu havaintoverkoston ylläpitäminen, kehittäminen sekä tiedonkeruu havaintoasemilta. Kerätty data prosessoidaan, laatu-varmistetaan, jonka jälkeen se tuotetaan ilmastotietokantaan, joka on myös saatavilla avoimena rajapintana asiasta kiinnostuneille kansalaisille. (Ilmatieteen laitos, 2016.)

Tutkimuksessa selvitetään, kuinka nykyiset Excel-pohjaiset raportointityökalut voidaan muuttaa dynaamisiksi lomakkeiksi moderneilla selainsovellustekniikoilla.

2 Tavoitteet

Tutkimuksen tavoitteena on kehittää nykyisiä nykyiset raportointitoiminnot siten, että halutut metatiedot havaintoasemista saadaan kerättyä mahdollisimman sujuvasti ja kustannustehokkaasti. Metatietojen kerääminen mahdollisimman tarkasti ja reaaliaikaisesti on yhteydessä tuotetun datan laatuun. Jotta edellä mainittuun tavoitteeseen päästäisiin, tulee ensin saada vastaus alla esitettyihin tutkimuskysymyksiin.

Miten nykyisiä raportointitoimintoja käytetään?

> Selvitetään miten Ilmatieteen laitoksen havaintoasemien web-pohjaisia raportointitoimintoja aseman huoltoon, tarkastamiseen ja ylläpitoon liittyen käytetään.

Mitä nykyisissä raportointitoiminnoissa koetaan ongelmallisiksi?

> Selvitetään koetut ongelmakohdat, joiden johdosta havaintoasemien raportointitoiminnot eivät vastaa tarpeita ja haittaavat työn suorittamista.

Miten nykyisiä raportointitoimintoja voidaan kehittää saadun palautteen perusteella, jotta saavutetaan toivottu lopputulos?

> Määritellään kehittämiskohteet, joista määritellään keskeisimmät ja niistä toteutetaan ne osat, jotka ovat opinnäytetyöhön käytettävissä olevan ajan puitteissa järkeviä. Tutkimus keskittyy käyttöliittymään ja sen parantamiseen/uudelleenkehittämiseen, nojaten olemassa oleviin käytettävyysteorioihin ja hyödyntäen moderneja selainsovellustekniikoita, kuten SPA:ta.

Tutkimus on rajattu

- havaintoasemien huoltoon
- havaintoasemien tarkastuksiin (esim. meteorologinen katselmus)
- laitteiden raportointiin ja siitä syntyvään metatietoon

Tutkimus on jatkoa suuntautumisharjoittelujaksolleni Ilmatieteen laitoksella.

3 Havaintoasemien hallintajärjestelmä

Ilmatieteen laitoksella on sisäisessä käytössä oleva HAV-sivusto, jolla pystytään esimerkiksi tarkastelemaan asemilta kerättyjä havainnointitietoja, hallitsemaan tehtäviä ja muuttamaan asemien metatietoja. Nykyinen laitehallinta sekä erityisesti huoltoraportointi toimii osittain manuaalisesti. Esimerkiksi huoltoraportit luodaan Excel-tiedostosta, joka tallennetaan järjestelmään PDF-tiedostona.

Pdf-tiedostossa on ongelmansa, koska sitä ei pystytä lukemaan koneellisesti. Tällöin sen tietoja ei ole mahdollista hyödyntää HAV-sivuston kautta, vaan jokainen mahdollinen huoltoraportti täytyy käydä manuaalisesti lävitse, mikäli tietoja niistä tarvitaan. Ongelmana ovat myös muualle HAV-sivustoon mahdollisesti kirjattavat tiedot, jotka aiheuttavat lisätyötä.

HAV-sivusto koostuu olio-pohjaisesta PHP-ohjelmakoodista, jota suoritetaan Apache-palvelimella. Tietokantana toimii ilmastotietokanta, joka on Oracle-pohjainen. Verkkosivusto on toteutettu HTML5-, CSS3- sekä JavaScript-kielillä.

4 Tarvekartoitus

Tarvekartoituksen tarkoituksena on saada vastaukset kysymyksiin, kuka käyttää järjestelmää, mitä hän tekee, missä ja milloin (Dennis, 2006, 6).

Vaatimusten kartoituksessa käytettiin hyvin yksinkertaista kyselyä, jolla kartoitettiin nykyjärjestelmän tila, siinä koetut haasteet ja toivotut kehitysideat. Vastausten pohjalta koostettiin lista keskeisistä asioista, jonka pohjalta luotiin järjestelmän määritys.

Tarvekartoitus tehtiin teemahaastatteluna, joka on välimuoto avoimesta- ja lomakehaastattelusta, ja jossa aihepiiri tiedettiin ennalta (Hirsjärvi, 2009, 208).

4.1 Kysymykset ja tulokset

Kyselyitä laadittiin kaksi erilaista; huoltoraportointikysely (Liite 1. Määräaikaishuoltokysely) ja edustavuusarviointikysely (Liite 2. Aseman edustavuusarviointikysely). Sisällöltään kysymykset olivat kummassakin samat, mutta kohdennettuna raportin tyypin mukaisesti. Kysely toteutettiin haastatteluna.

Ensimmäiselle haastateltavalle kyselyssä oli yhteensä 6 tai 7 kysymystä raportointikohteesta riippuen ja seuraaville vain 3. Ensimmäisen haastateltavan kohdalla kartoitettiin, miten raportointi käytännössä tehdään ja mitä esi- ja loppuvaiheita siinä on. Lopuissa haastatteluissa keskityttiin vain nykyisen järjestelmän heikkouksiin, mitkä olisivat ensisijaisesti parannettavat kohdat ja mitä muita ideoita kehittämiseen haastateltavalla on.

Kyselyiden keskimääräinen kesto oli 0.5 h ja haastattelut pidettiin Ilmatieteen laitoksella. Edustavuusarviointikyselyssä haastateltiin 5 henkilöä ja määräaikaishuoltokyselyssä 6 henkilöä. Jokainen haastateltava valittiin sen perusteella, että heillä on kokoemusta aihealueesta.

4.1.1 Määräaikaishuollon raportointitarpeet

Määräaikaishuolto suoritetaan havaintoasemien laitteille ennalta määrättyjen aikajaksojen puitteissa, jolloin tehdään usein valmistajan ohjeiden mukainen huolto. Huoltoon voi kuulua muun muassa laitteen puhdistamista, kalibrointia tai kuluvien osien vaihtamista.

Erityisen huonona koettiin huollon yhteydessä tapahtuva Excel-lomakkeen täyttö, sillä tämä toistui lähes jokaisella haastateltavalla. Muutama kuitenkin piti Exceliä hyvin helppona käyttää, mutta myönsivät myös prosessin monimutkaisuuden useine vaiheineen. On-

gelmaisimpana koettiin se, että tietoja ei saatu suoraan tarvittaviin osiin järjestelmässä, kuten vikailmoitukseen, tehtävienhallintaan, laitehallintaan ja aseman tietoihin, joiden lisäksi Excel-lomake muunnettiin PDF-muotoon ja tallennettiin järjestelmään. Tästä kaikki haastateltavat olivat yhtä mieltä.

Virheiden mahdollisuus tuotiin voimakkaasti esille. Tällaisia ovat esimerkiksi laitteiden/antureiden sarjanumerot, koska Excel ei niitä laitetietokannasta tarkista. Tämän lisäksi kyseiset laitteet/anturit pitää muistaa siirtää laitehallinnassa oikealle asemalle, joka liian usein jää tekemättä ja mainittiin myös, että tämän johdosta saattaa asemalla olla laitteita, jotka puuttuvat laiterekisteristä.

Kävi myös ilmi, että Excel-lomake on suojattuna salasanalla, joka on vain yhden henkilön takana ja mahdolliset muutokset lomakkeeseen kestävät usein liian pitkään. Salasanasuojaus on asetettu estämään tahattomat muokkaukset vääriin kohtiin lomaketta sekä varmistamaan, että kaavat pysyisivät eheänä.

Toiveita huollon raportointikehitykseen esitettiin seuraavasti:

- sähköinen järjestelmä, joka olisi yhteydessä suoraan tarvittaviin tietoihin
- ei turhia kysymyksiä vaan mukaudutaan tehtävän tyypin mukaisesti
- laitteet haetaan automaattisesti tietokannasta
- työmäärän pienentäminen
- helppokäyttöisyys: ei saa olla hankalampi kuin Excel

Erityistä huomiota kyselyn perusteella tulee kiinnittää siihen seikkaan, että verkkoyhteys ei läheskään aina ole saatavilla kentällä – esimerkiksi ulkomeren luodoilla sijaitsevilla havaintoasemilla.

4.1.2 Aseman edustavuusarvioinnin raportointitarpeet

Havaintoaseman edustavuusarvioinnilla tarkoitetaan aseman ympäristön kartoittamista, kuten esimerkiksi maaperä, puiden sijainti ja korkeus.

Kaikki haastateltavat olivat samaa mieltä, että Excel-tiedosto on hankala käyttää ja altistaa virheille. Tietojen siirto Excelistä tarvittaviin tietokantoihin koettiin myös haastavana, koska samankaltaisia tietoja tarvitaan useampaan eri paikkaan. Erityisesti työaikaa kuluu valokuvien käsittelyyn, kuvia voi olla jopa 100 kappaletta per asema. Niiden nimeäminen

koettiin työlääksi prosessiksi, koska kuvat pitää nimetä jokaisen anturin kohdalla erikseen, moneen eri ilmansuuntaan.

Toivottiin myös sitä, että jo olemassa olevat tiedot asemista ja edellisistä lomakkeen täytöistä haettaisiin lomakkeisiin automaattisesti, koska nyt ne joudutaan kirjoittamaan joka kerta uudestaan, vaikka nämä tiedot eivät useinkaan muutu.

Toiveita havaintoaseman edustavuusraportoinnin kehitykseen esitettiin seuraavasti:

- valokuvien lisäämiseen drag & drop -tuki esim. horisonttiympyrään, jossa on määritellyt kulmat
- valokuvien automaattinen nimeäminen
- olemassa olevien tietojen hakeminen tietokannasta
- yhdellä kerralla tiedot tarvittaviin paikkoihin

4.1.3 Johtopäätös ja toimenpiteet

Aseman edustavuusarviointi- sekä määräaikaishuoltoraportti tulisi toteuttaa sähköisenä lomakkeena, jossa pyydettävät tiedot määräytyvät aseman mukaan ja johon pystytään tuomaan valmiiksi tarvittavat tiedot tietokannasta. Kehitystyössä tulee kuitenkin ottaa huomioon tilanne, jossa verkkoyhteyttä ei ole saatavilla. Tällöin datan tulisi olla synkronoituna tietokannasta lokaalisti laitteeseen.

Järjestelmän tulisi toimia sekä kannettavilla Windows-tietokoneilla että Android-tableteilla, joita nykyäänkin tarvitaan kenttätyössä. Tällä saavutetaan se, että ei tarvita erillistä laitetta sovelluksen käyttämiseen. Järjestelmän pitäisi myös mahdollistaa tulostettava lomake, mikäli jostain syystä ei ole mahdollista käyttää sähköistä versiota.

Modulaarisuus tulee ottaa huomioon sovelluksen toteutuksessa, jotta laajennettavuus uusille ominaisuuksille on tulevaisuudessa mahdollista. Sovellukseen ei kuitenkaan ole järkevää toteuttaa toimintoja, joiden käyttäminen on HAV-sivuston kautta järkevämpää, kuten esimerkiksi itse lomakkeiden luonti, tehtävienhallintajärjestelmän tehtävien luonnin tai vikailmoituksen tekeminen.

Mobiilisovellus toteuttaa siis seuraavaa:

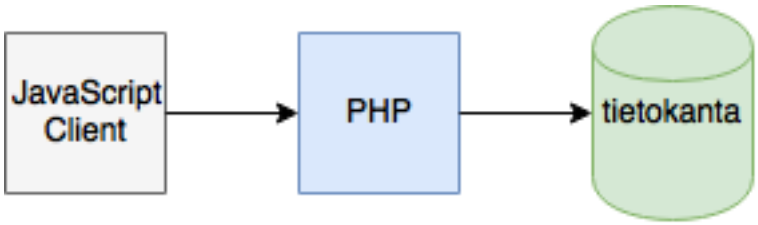
- lomakkeet
    - ympäristö- ja edustavuusarviointi
    - määräaikaishuolto

    - vikahuolto
- lomakkeiden generointi tietokannasta saatavan JSON-scheman mukaisesti.
- lomakkeiden tietojen tallennus ja muokkaaminen
- laitteiden siirrot, TEHA-tehtävien ja vikailmoitusten päivitys
- valokuvien ottaminen havaintoasemilta
- lomakkeissa hyödynnettävien tietojen synkronointi tietokannan kanssa
- aseman datatarkistus, kulkeeko data asemalta kantaan

5 Uuden raportointijärjestelmän vaatimuksia

Tässä luvussa kuvataan uuden raportointijärjestelmän teknisiä vaatimuksia, tietovarasto-vaihtoehtoja sekä käytettävyysvaatimuksia.

Raportointijärjestelmän toimintaympäristö koostuu yksinkertaisuudessaan asiakaspäässä suoritettavasta JavaScript-sovelluksesta, palvelimella suoritettavasta PHP-ohjelmasta sekä tietokannasta. (Kuva 1.)



Kuva 1: Toimintaympäristö

Kuten aikaisemmassa tutkimusosassa yhteenvetona voitiin päätellä, niin kehityskohteena on erityisesti koko Excel-lomakkeen muuttaminen sähköiseksi. EAV-tietokantarakenteella on mahdollista saada dynaaminen tuki kentille, jotka sidotaan havaintolaitteeseen, sekä itse raporttiin. Dynaamiset kentät voidaan vaihtoehtoisesti toteuttaa hyödyntämällä JSON-tukea, joka on saatavissa PostgreSQL-tietokannassa.

EAV:n hyötynä JSON-tietotyyppiin verrattuna olisi kuitenkin se, että datalle saadaan eheysvaatimukset tietyissä rajoissa. JSON-tietotyypissä tällaista vaihtoehtoa ei ole.

5.1 Tietokanta

Ilmatieteen laitoksella on käytössä Oraclen tietokannan versio 11. JSON-tietotyypin tuki on kuitenkin vasta versiossa 12.1.0.2 (Venzl, 2015), ja tuki helpottaisi dynaamisen lomakkeen luontia ja SQL-kyselyiden tekemistä. Päivitystä uudempaan ei kuitenkaan ole luvassa lähitulevaisuudessa. PostgreSQL-tietokanta on kuitenkin mahdollista saada uudemmalla, vähintään 9.5 versiolla, jossa on tuki JSON-tietotyypille.

Vaihtoehtona olisi myös XML-tietotyyppi, mutta se koetaan turhan runsassanaisena vaikkakin se on paikoitellen selkeämpi lukijalle kuin JSON-formaatti.

5.1.1 JSON-tietotyyppi

JSON-tietotyyppi on viime aikoina ilmestynyt useimpiin relaatiotietokantoihin (MySql, MariaDB, Oracle, PostgreSQL, SQL Server). Sen voidaan tallentaa joustavasti schemavapaata tietoa. Tämä on RMDBS-tietokantojen vastaus NoSQL-tietokannoille ja yhdistääkin molemman hyödyt.

JSON koostuu aaltosuluista, joilla merkitään objektia sekä hakasuluista, joilla merkitään taulukko. Muuttuja sekä arvo merkitään heittomerkkien sisään ja erotetaan toisistaan kaksoispisteellä (Esimerkki 1).

```
[
        {”product_name”:”Scoop”},
        {”product_name”:”Tray”}
]
```

Esimerkki 1. Tuotteet JSON-muodossa

5.1.2 XML-tietotyyppi

XML-tietotyyppi ei ole tuettu MySql 5.6, MariaDB 10 tai PostgreSQL 9.6-tietokannoissa, mutta Oracle-tietokannassa on sille tuki.

XML-tietotyypillä saavutettavat hyödyt ovat samat kuin JSON-tietotyypissä, mutta XML-formaatti itsessään on enemmän strukturoitu, kun taas JSON on puolestaan vapaammin kuvattava.

XML muodostuu tageista, joilla merkitään rakenne, joka muutetaan haluttuun tietomuotoon ohjelmallisesti. Alla olevasta muodostuisi täten taulukko, jossa on usea ”product” (Esimerkki 2).

```
<?xml version="1.0" encoding="UTF-8"?>
<products>
            <product>
                        < product_name>Scoop</product_name >
            </product>
            <product>
                        < product_name>Tray</ product_name>
```

```
        </product>
</products>
```
Esimerkki 2. Tuotteet XML-muodossa

5.1.3 EAV-malli

EAV-mallia hyödynnetään usein, kun tarvitaan dynaamisesti mukautuvia tietueita. EAV-mallilla saavutetaan usein relaatiotietokannan tuoma tiedon eheys, mutta samaan aikaan muodostaa tietokantarakenteesta monimutkaisen ja joudutaan tekemään vähintään kaksi pyyntöä tietokantaan, jotta saadaan koostettua haluttu tieto.

5.2 Käyttöliittymä

Tässä luvussa esitellään käytettävyysperiaatteita, joita sovelletaan uuden raportointijärjestelmän kehityksessä.

Käyttöliittymän selkeys on olennaisin osa käyttöliittymän suunnittelussa. Käyttöliittymän tulisi olla helposti lähestyttävä ja käytettävä sekä miellyttävä. Käyttöliittymän tulisikin olla rakennettu siten, että käyttäjä oivaltaa useimmat toiminnot visuaalisten vihjeiden avulla.

Käyttöliittymän yksinkertaisuus tuo puolestaan haasteita verkkosovelluksissa erityisesti siihen, miten kaikki toiminnot saadaan sijoitettua käyttöliittymään helposti löydettäviksi ja ilman, että siitä aiheutuu haittaa käytettävyydelle. Erityisesti tämä korostuu mobiililaitteissa, missä siirryttäessä työpöydästä mobiiliin, pieneen näyttöön, menetetään arvokasta ikkunatilaa.

Käyttöliittymäsuunnittelussa käytettävyys on ensiarvoisen tärkeätä. Jakob Nielsen on jakanut käytettävyyden viiteen laatukomponenttiin:

- Opittavuus (learnability), kuinka helposti käyttäjä suorittaa tehtävän ensimmäisellä kerralla
- Tehokkuus (effiency), kuinka tehokkaasti tehtävä suoritetaan uudestaan
- Muistettavuus (memorability), kuinka nopeasti sama tehokuustaso saavutetaan palattaessa tehtävään myöhemmin
- Virheet (erros), kuinka monta virhettä tapahtuu
- Miellyttävyys (satisfaction), kuinka miellyttävää sivustoa on käyttää

(Nielsen, 2012.)

6 Käytettävät ohjelmistot ja ohjelmointikielet

Tässä luvussa esitellään havaintopalveluiden raportointisovelluksessa käytettäviä tekniikoita.

6.1 HTML5 ja CSS3

W3C:n HTML5- ja CSS3-standardit toivat ilmestyessään useita uusia verkkosivustojen tekemistä helpottavia ominaisuuksia. HTML sekä CSS ovat olleet olemassa jo vuosista 1990 (Longman, 1998) sekä 1996 (Bos, 2016) ja niiden kehitys on ollut suhteellisen hidasta.

HTML5-merkintäkieli on uusin versio verkkosivujen merkintäkielestä, jonka avulla kuvataan sivuston rakenne käyttäen standardoituja tageja. HTML5 toi HTML-merkintäkieleen uusia rakennetta kuvaavia tageja, esimerkiksi section- ja article-tagien tuen. Nämä eivät kuitenkaan ole käyttäjän kannalta oleellisia, mutta esimerkiksi parantavat sivuston schematiikkaa (Mozilla Developer Foundation).

CSS on verkkosivujen ulkoasumäärittelyn standardi, jonka avulla pystytään tarkasti määrittelemään, miltä mikäkin elementti näyttää. CSS3 toi joukon uusia ominaisuuksia, joiden avulla verkkosivuston ulkoasusta saadaan entistä houkuttelevampi. Tyylimäärityksillä saadaan aikaiseksi entistä näyttävämpiä sivuja hyödyntämällä esimerkiksi animaatiotukea visuaalisten vihjeiden tueksi (Daliot, 2015).

HTML5 Webstorage-määritys on ehkä modernien selainsovellustekniikoiden kannalta oleellisin, koska tilatietoa ja muuta dataa voidaan säilöä selaimeen keksien sijasta, joka myös säilyy selaimen sulkemisen jälkeenkin. (Heilmann, 2010).

6.2 AJAX ja JavaScript

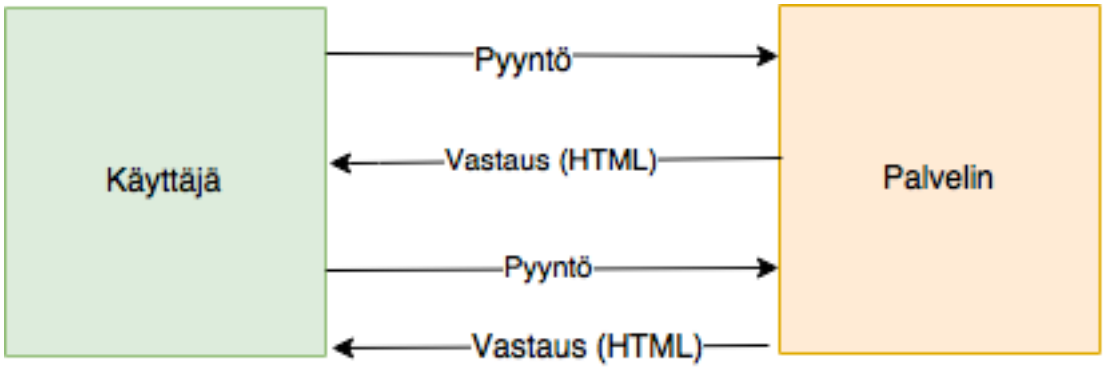
AJAX tekniikkaa käytetään siirtämään tietoja selaimen ja palvelimen välillä dynaamisesti, usein hyödyntäen JavaScriptiä. (Ullman, 2007).

JavaScript on oleellinen osa moderneja selainsovellustekniikoita, koska sen avulla voidaan muuttaa sivustoa dynaamisesti. Ongelmana on kuitenkin se, että esimerkiksi tietoturvan johdosta JavaScript saattaa olla estettynä selaimessa, jolloin saavutettu hyöty käytettävyyteen on olematon. Selaimet eivät myöskään tue kaikkia kielen ominaisuuksia (Kangax.github.io. 2017). Verkkosivustoa suunniteltaessa hyvä käytäntö on se, että sivus-

to toimii ilmankin JavaScriptiä ja palaa takaisin normaaliin sivunlatausmalliin, mikäli JavaScript ei toimi (Graceful Degradation). (Buckler, 2009).

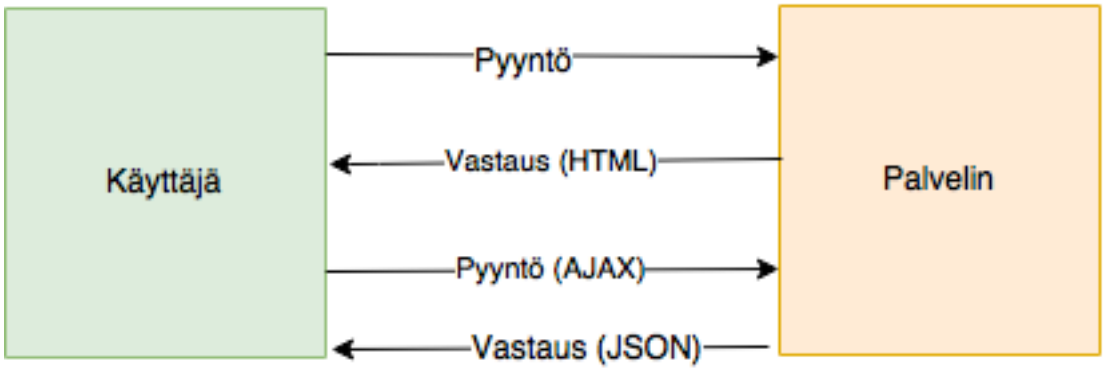
6.3 SPA

Perinteinen websovellus perustuu sivunlatauksiin, jossa pyydetään palvelimelta kokonainen websovellus uudestaan esimerkiksi seuraamalla linkkiä. (Kuva 2.)



Kuva 2: Selain - palvelin pyyntö (Wasson, 2013).

Single Page Application on JavaScriptillä ja usein Ajaxilla ja JSON-muotoisella datalla toteutettu kokonaisuus, jolla poistetaan sivulataukset ensimmäistä latausta lukuun ottamatta. Kokonaisuudessaan prosessi etenee siten, että AJAX-pyynnöllä haetaan tietoja palvelimelta ja saadaan vastauksena JSON-muotoinen data, jonka avulla koostetaan esimerkiksi uusi lomake sivulle ilman, että sivun muita osioita päivitetään mitenkään. (Kuva 3.)



Kuva 3: Ajax selain - palvelin pyyntö (Wasson, 2013).

Hyötynä SPA:n käyttämisestä on erityisesti toimintojen sulavuus, joka mahdollistaa verkkosovelluksen vaikuttamaan työpöytäsovellukselta, jossa vaste on usein välitön. Palvelimelta ei pyydetä koko sivua, vaan vain osa siitä ja tällöin myös vastaus on kooltaan pienempi ja siirtyy nopeammin (Kava, 2014, 9).

Aikaisemmassa suuntautumisharjoittelujaksossa kehitin yhden hallintaosion hyödyntämään SPA-mallia. Integroin tarvittavat toiminnot palvelinpuolella hyödyntäen PHP:ta jo silloin, joten tämän tekniikan käyttö ei vaadi taustajärjestelmään suurempia muutoksia AJAX-rajapintaa lukuun ottamatta.

6.4 Bootstrap

Bootstrap on Twitterin kehittämä CSS3-tyylikirjasto, jolla saa helposti yhtenäisen tyylin käyttöliittymälle (Bootstrap, 2017). Bootstrap on jo entuudestaan käytössä Ilmatieteen laitoksella, joten sen valinta tähän projektiin on looginen.

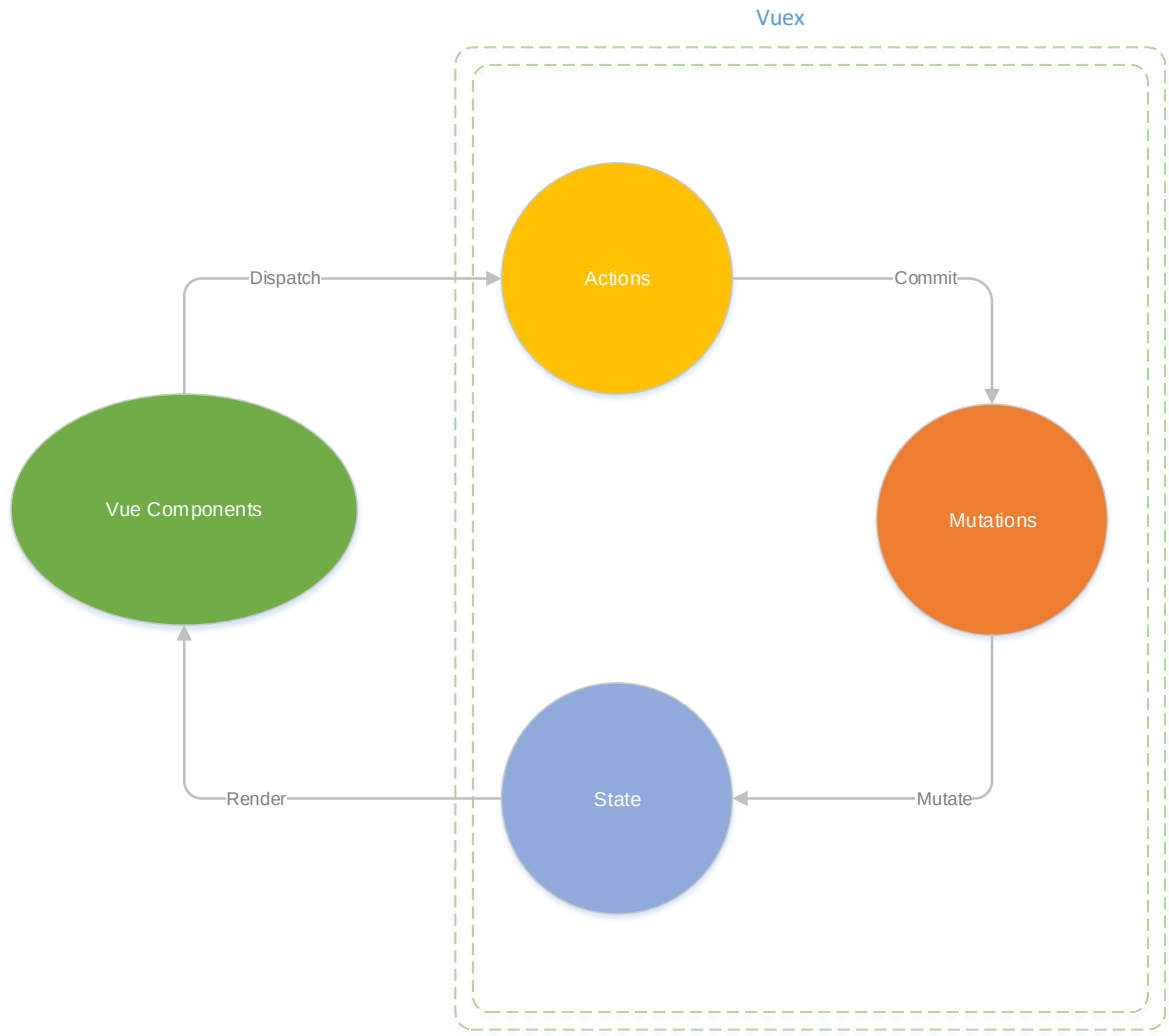
6.5 JQuery

JQuery on JavaScript-kirjasto, joka sisältää lukuisia metodeja. Niiden tarkoitus on helpottaa kehittäjän työtakkaa HTML-dokumentin käsittelyssä (jQuery, 2017). JQuery on jo entuudestaan käytössä Ilmatieteen laitoksella, joten myös sen valinta tähän projektiin on looginen.

6.6 VUE.JS

Vue.js (kehys), Vuex (tilanhallinta) ja Vue Router (reititys) muodostavat yhdessä kokonaisuuden, jonka avulla on mahdollista tehdä laajojakin SPA-sovelluksia. Itsessään Vue.js luo sivupohjista virtuaalisen DOM:n, josta vain tarvittavat päivitykset tehdään selaimen DOM-puuhun. Näin vältetään DOM käsittelyn hitaus ja saadaan tarvittava reaktiivisuus aikaiseksi. Vue.js on kokonaisuudessaan avoimen lähdekoodin projekti ja lisensoitu MIT-lisenssillä.

Vuex prosessimalli (Kuva 4.) kuvaa koko prosessin, jonka pohjalta sovelluksen tilaa (State) ja datavirtaa hallitaan. Ensimmäisenä kutsutaan metodeita (Actions), jotka suorittavat jonkin tehtävän. Metodin aiheuttamat muutokset suoritetaan (Mutations), tallennetaan tilanhallintaan (State) ja viimeisenä suoritetaan käyttöliittymäpäivitys (Vue Components).



Kuva 4: Vuex-prosessimalli (Goligo.io. 2017.)

Kuvassa 5 (Kuva 5) <template>-tagien sisällä määritellään sivupohja, joka muunnetaan Vue.js:n sisällä virtuaalidomiksi. <script>-tagien sisällä on itse ohjelma, jonka Vue.js suorittaa. <style>-tageissa määritellään mahdolliset tyylitiedostot.

```
<template>
  <div id="app">
    <router-view></router-view>
  </div>
</template>

<script>
import auth from './auth.js'

export default {
  data () {
    return {
      loggedIn: auth.check()
    }
  },
  created () {
  }
}
</script>

<style lang="sass">
@import "src/assets/scss/style.scss"
</style>
```

Kuva 5: Vue.js sivupohja

Vue.js on suhteellisen tuore tulokas ja React.js olisi vakiintuneempi, mutta React.js häviää vaihtoehtona olemalla monimutkaisempi (hitaampi omaksua) ja ainakin synteettisesti hitaampi (Vuejs.org, 2017) kuin Vue.js. Hitaus on kuitenkin sinällään kyseenalainen asia, koska testauksen on tehnyt vue.js. Verkossa kuitenkin on samankaltaisia havaintoja muiltakin (esim. Jorgé, 2016).

6.7 Node.js

Node.js on palvelinpään JavaScript-ajoympäristö, jota hyödynnetään sekä palvelimien että rajapintojen toteuttamiseen. Node.js sisältää saman V8-tulkin kuin Chrome-selainkin (Node.js, 2017). Node.js on lisensoitu avoimen lähdekoodin MIT-lisenssillä.

Tässä projektissa Node.js ympäristöä hyödynnetään kehitysympäristössä esim. Vue.js sovelluksen rakentamisessa Webpackin avustuksella sekä Cordovan käytössä.

6.8 NPM, Node Package Manager

NPM on keskeinen työkalu, jolla asennetaan kirjastoja Node.js-projektiin. NPM hoitaa versiohallinnan ja pakettien hakemisen pakettivarastoista (npmjs.org, 2017).

6.9 Webpack ja Babel

Webpack on moduulinen paketointityökalu, jolla voidaan paketoida JavaScript- ja HTML-tiedostoja yhdeksi paketiksi (webpack.js.com, 2017). Babel puolestaan kääntää es2015-standardin mukaiset JavaScriptit (kuten esimerkiksi import-komento) toimiviksi, jos niitä ei ole tuettuna selaimissa (Babel, 2017).

6.10 Cordova-kehitysympäristö ja sen asentaminen

Apache Cordova mahdollistaa natiivien sovellusten tekemisen Androidille, iOSille sekä Windowsille. Cordova paketoi HTML/CSS/JS-tiedostot sekä tarvittavat JavaScript-rajapinnat järjestelmään yhdeksi paketiksi. Apache Cordova on avoimen lähdekoodin projekti ja lisensoitu Apache 2.0-lisenssillä.

Cordovan käyttöönotto Frontend-kehittäjälle on hyvin tuttua, mikäli Node.js-ympäristö on tuttu, koska Cordovan työkalut ovat rakennettu Node.js:llä ajattavista JavaScript-skripteistä.

Tämän projektin kannalta Cordova on vaatimuksien mukainen. Cordova pystyy tuottamaan yhden koodipohjan, jota voidaan suorittaa eri käyttöjärjestelmissä sekä pelkässä selaimessa.

Asennus on tässä kuvattu Ubuntu 16.04. käyttöjärjestelmään. Cordovalla on kuitenkin tuki myös Windows ja Mac OS-järjestelmille, koska Cordovan vaatimukset kehitysympäristöstä rajautuu JDK 7, Android SDK sekä Node.js yhteensopiviin käyttöjärjestelmiin (Cordova Documents).

Asennetaan Java

```
sudo apt-get install default-jre, default-jdk
```

Javan lisäksi tarvitaan Android SDK, joka asennetaan lataamalla tarvittava paketti sivustolta https://developer.android.com/studio/index.html.

Asennetaan node.js

```
curl –sL https://deb.nodesource.com/setup_7.x | sudo -E bash -
sudo apt-get install –y nodejs
```

Asennetaan Webpack ja Babel, sekä muut tarvittavat paketit

```
sudo npm install –save-dev webpack babel-loader babel-preset-es2015 babel-core live-server raw-loader
```

Asennetaan Cordova

```
sudo npm install –g cordova
```

Luodaan sovelluksen rakenne haluttuun kansioon

```
cordova create hav-app com.fmi.hav Havaintoasemat
```

Siirrytään luotuun kansioon

```
cd hav-app
```

Lisätään webpack.config.js

```
nano webpack.config.js


var path = require('path');


module.exports = {
  entry: './app/main',
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'www/js')
  },
  cache: false,
  module: {
      loaders: [{
          test: /\.js$/,
          loader: 'babel-loader',
          query: {
```

```
            presets: ['es2015']
        }
    }, {
        test: /\.html$/,
        loader: 'raw-loader'
    }]
 }
};
```

Lisätään alustat

```
cordova platform add android --save
cordova platform add browser --save
```

Lisätään tarvittavat moduulit

```
cordova plugin add cordova-plugin-camera
cordova plugin add cordova-sqlite-storage
cordova plugin add cordova-plugin-device-orientation
cordova plugin add cordova-plugin-geolocation
cordova plugin add cordova-plugin-file
```

Paketoidaan halutut sovellukset

```
webpack
cordova build                # build all platforms
cordova build browser        # build browser
```

Pakettien päivittäminen

```
npm install –g cordova-check-plugins
cordova-check-plugins –update=auto
```

Ajetaan sovellus

```
webpack
cordova run android          # run android
cordova run browser          # run browser
```

6.11 Yhteenveto

Toteutustekniikoiden hyödyt ovat erityisesti siinä, että niiden käyttöön ei vaadita suurempia erityisosaamisia, joita nykypäivän Front End-kehittäjällä ei olisi jo valmiiksi. Cordova on ehkä haastavin teknologioista, mutta sillä vältetään se, että ei tarvitse tehdä jokaiselle tuetulle alustalle omaa natiivia sovellusta. Näin vaadittavat kehitystaidot pysyttelevät HTML/CSS/JS-tekniikoissa. Cordova myös sisältää valmiiksi tarvittavat JavaScript-rajapinnat käyttöjärjestelmiin, tuen esim. sensoreille ja kameroille.

Toinen iso hyöty web-tekniikkapohjaisessa sovelluksessa on tämän sovelluksen kohdalla erityisesti se, että lomakkeet saadaan vaivattomasti luotua dynaamisesti syötettävän JSON-tiedon perusteella, joka nopeuttaa muutoksien tekemistä lomakkeisiin ilman itse sovelluksen päivittämistä. Sovellus on mahdollista päivittää myös automaattisesti hakemalla palvelimelta muuttuneet JavaScript-, CSS- sekä HTML-tiedostot.

7 Suunnittelu

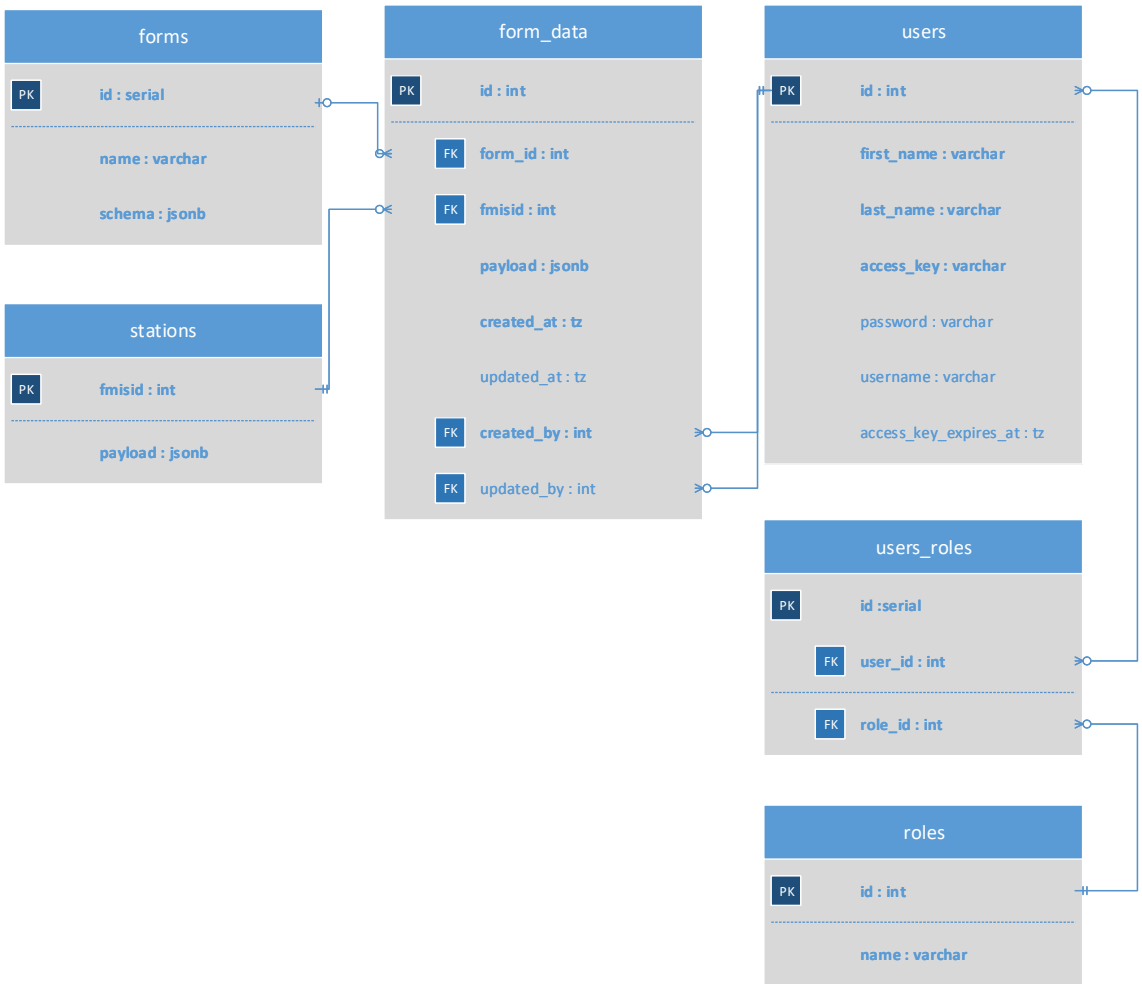
Seuraavassa on kuvattu havainnointipalvelujen uuden raportointisovelluksen suunnittelun vaiheet.

7.1 Tietokantasuunnittelu

Tietokantasuunnittelussa suunniteltiin sekä rajapinnan PostgreSQL-tietokannan, joka sijaitsee rajapinnan takana, rakenne, että itse sovelluksessa hyödynnettävän SQLite-tietokannan rakenne, joka toimii käytännössä välimuistina erityisesti offline-käyttöä ajatellen.

7.1.1 PostgreSQL-tietokannan relaatiokaavio

Tässä on kuvattu PostgreSQL-tietokannan mukainen relaatiokaavio ja sen viite-eheydet (Kuva 6.)



Kuva 6: PostgreSQL-tietokannan relaatiokaavio

7.1.2 PostgreSQL-tietokannan luontilauseet

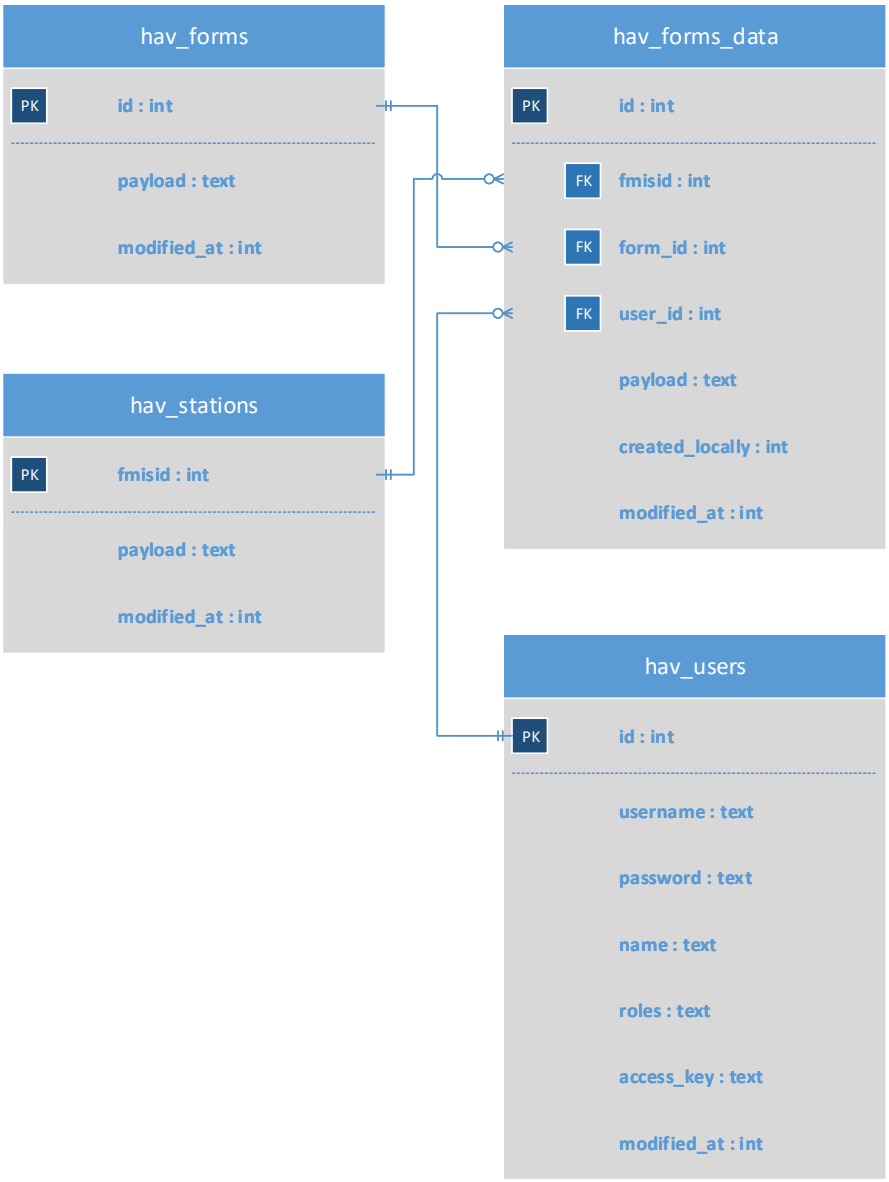
Alla on esitetty kuvan 6 mukaiset luontilausekkeen luontijärjestyksessä.

```
CREATE TABLE forms (
  id SERIAL NOT NULL PRIMARY KEY,
  name VARCHAR(128) NOT NULL UNIQUE,
  schema JSONB NOT NULL
);

CREATE TABLE stations (
  fmisid INT NOT NULL PRIMARY KEY,
  payload JSONB NOT NULL
);

CREATE TABLE users (
  id INT NOT NULL PRIMARY KEY,
  password VARCHAR(128) NOT NULL,
  username VARCHAR(128) NOT NULL UNIQUE,
  first_name VARCHAR(64) NOT NULL,
  last_name VARCHAR(64) NOT NULL,
  access_key VARCHAR(128) NOT NULL UNIQUE,
  access_key_expires_at TIMESTAMP WITHOUT TIME ZONE NOT NULL
);

CREATE TABLE roles (
  id INT NOT NULL PRIMARY KEY,
  name VARCHAR(64) NOT NULL UNIQUE
);

CREATE TABLE form_data (
  id SERIAL NOT NULL PRIMARY KEY,
  form_id INT NOT NULL REFERENCES forms (id) ON DELETE RESTRICT ON UP-
DATE CASCADE,
  fmisid INT NOT NULL REFERENCES stations (fmisid) ON DELETE RESTRICT ON
UPDATE CASCADE,
  payload JSONB NOT NULL,
  created_at TIMESTAMP WITHOUT TIME ZONE NOT NULL DEFAULT CUR-
```

```
RENT_TIMESTAMP,
  updated_at TIMESTAMP WITHOUT TIME ZONE,
  created_by INT NOT NULL REFERENCES users (id) ON DELETE RESTRICT ON
UPDATE CASCADE,
  updated_by INT REFERENCES users (id) ON DELETE RESTRICT ON UPDATE
CASCADE
);

CREATE TABLE users_roles (
  role_id INT NOT NULL REFERENCES roles (id) ON DELETE CASCADE ON UPDATE
CASCADE,
  user_id INT NOT NULL REFERENCES users (id) ON DELETE CASCADE ON UPDA-
TE CASCADE,
  PRIMARY KEY (role_id, user_id)
);
```

7.1.3 SQLite-tietokannan relaatiokaavio

Seuraavassa on kuvattu sovelluksen sisäinen SQLite-tietokannan relaatiokaavio (Kuva 7), johon tallennetaan rajapinnasta haetut tiedot offline-käyttöä ajatellen. Tiedot synkronoidaan rajapinnan kautta PostgreSQL-tietokannan kanssa automaattisesti verkkoyhteyden ollessa saatavilla.



Kuva 7: SQLite-tietokannan relaatiokaavio

7.1.4 SQLite-tietokannan luontilauseet

Alla on esitetty kuvan 7 mukaiset luontilausekkeet luontijärjestyksessä.

```
PRAGMA foreign_keys = ON;

CREATE TABLE hav_stations (
  fmisid INTEGER NOT NULL PRIMARY KEY,
  payload TEXT NOT NULL,
  modified_at INTEGER NOT NULL
```

```
);

CREATE TABLE hav_users (
  id INTEGET NOT NULL PRIMARY KEY,
  username TEXT NOT NULL UNIQUE,
  password TEXT NOT NULL,
  name TEXT NOT NULL,
  roles TEXT NOT NULL,
  access_key TEXT NOT NULL,
  modified_at INTEGER NOT NULL
);

CREATE TABLE hav_forms (
  id INTEGER NOT NULL PRIMARY KEY,
  payload TEXT NOT NULL,
  modified_at INTEGER NOT NULL
);

CREATE TABLE hav_forms_data (
  id INTEGER NOT NULL PRIMARY KEY,
  fmisid INTEGER NOT NULL,
  form_id INTEGER NOT NULL,
  user_id INTEGER NOT NULL,
  payload TEXT NOT NULL,
  created_locally INTEGER NOT NULL DEFAULT 1,
  modified_at INTEGER NOT NULL,
  FOREIGN KEY(fmisid) REFERENCES hav_stations(fmisid),
  FOREIGN KEY(form_id) REFERENCES hav_forms(id),
  FOREIGN KEY(user_id) REFERENCES hav_users(id)
);
```

7.1.5 Lomakekenttien schema

JSON schema on kehitetty kuvaamaan JSON-rakennetta ja siihen kohdistettuja vaatimuksia. Schema määrittelee tietueen tyypin, esimerkiksi alla on kyse objektista ja mikä on pakollista tai vapaaehtoista. (json-schema.org, 2017.)

Properties kohdassa puolestaan määritetään millaista ja minkä muotoista tietoa JSON sisältää.

Alla oleva JSON schema kuvaa sovelluksen dynaamisten lomakekenttien JSON-rakenteen, jonka avulla generoidaan lomake näytölle.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "form": {
      "type": "object",
      "properties": {
        "title": {
          "type": "string"
        }
      },
      "required": [
        "title"
      ]
    },
    "inputs": {
      "type": "array",
      "input": {
        "type": "object",
        "properties": {
          "name": {
            "type": "string"
          },
          "label": {
            "type": "string"
          },
          "type": {
            "type": "string"
          },
          "constraints": {
            "type": "object",
            "properties": {
              "presence": {
                "type": ”boolean”
              },
              "length": {
                "type": "object",
                "properties": {
                  "max": {
                    "type": "integer"
                  }
                },
                "required": [
                  "min"
                ]
              }
            },
            "required": [
              "presence",
```

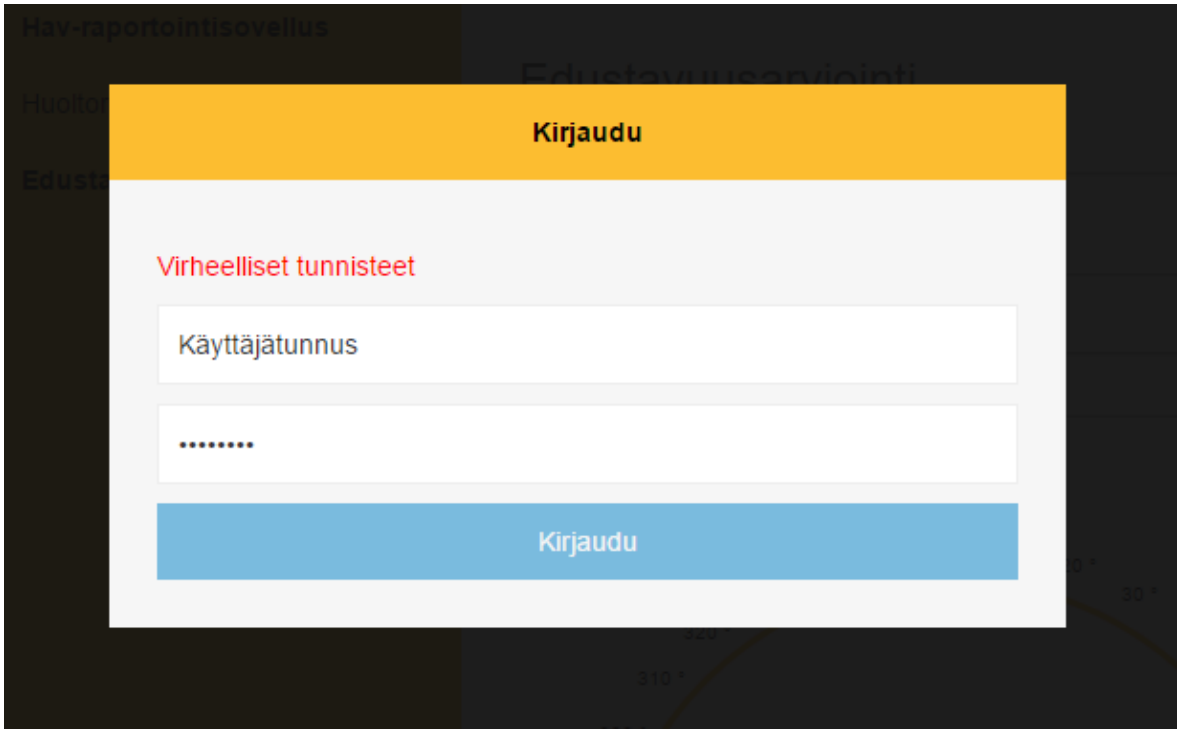
```
              "length"
            ]
          }
        },
        "required": [
          "name",
          "label",
          "type"
        ]
      }
    }
  },
  "required": [
    "form",
    "inputs"
  ]
}
```

7.2 Käyttöliittymäsuunnittelu

Sovelluksen typografia ja väri määrittyvät Ilmatieteen laitoksen graafisen ohjeistuksen mukaisesti. Fonttina on Arial ja päävärinä keltainen #FCBD30. Painikkeiden väriksi valitsin #7AADDE.

Ulkoasu on yksinkertaistettu ja hyödyntää Bootstrap-kirjastoa sovelluksen tyylittelyssä ja lomake-elementissä.

Sovelluksen käynnistyessä pyydetään kirjautumaan sovellukseen (Kuva 8), jonka onnistuessa siirrytään etusivulle (Kuva 9).

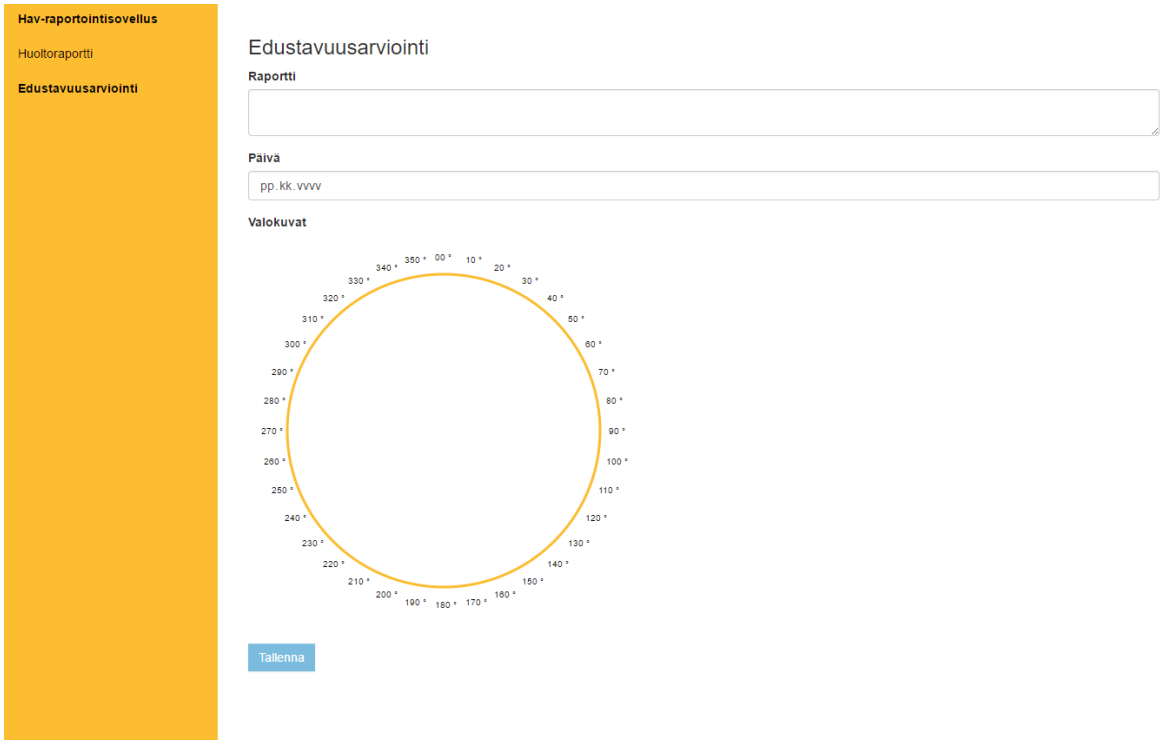


Kuva 8: Kirjautumisikkuna

Alla olevassa kuvassa (Kuva 9) on hyvin yksinkertainen aloitussivu ja siihen liittyvä staattinen valikko.

Kuva 9: Etusivu

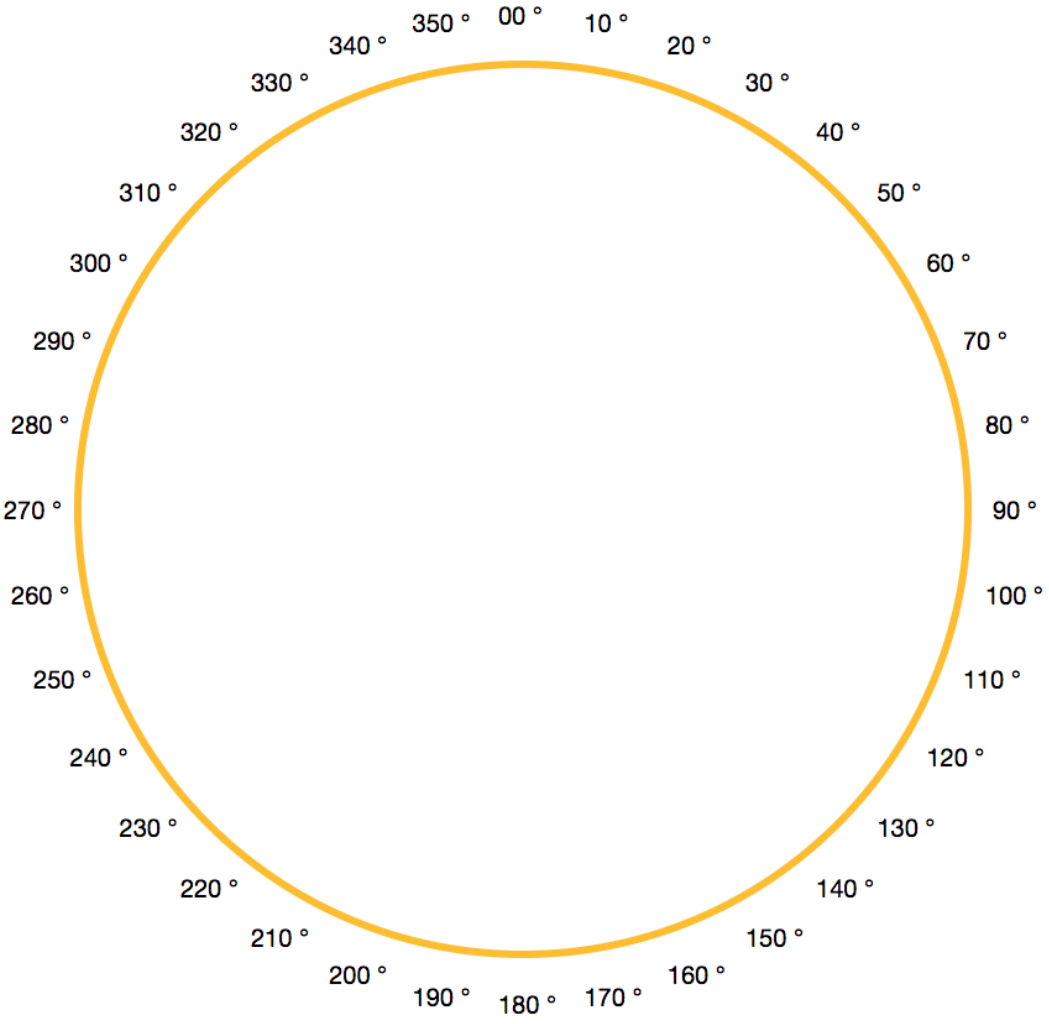
Lomakkeen ulkoasussa hyödynnetään Bootstrap-kirjaston tyylin mukaisia lomake-elementtejä (Kuva 10).



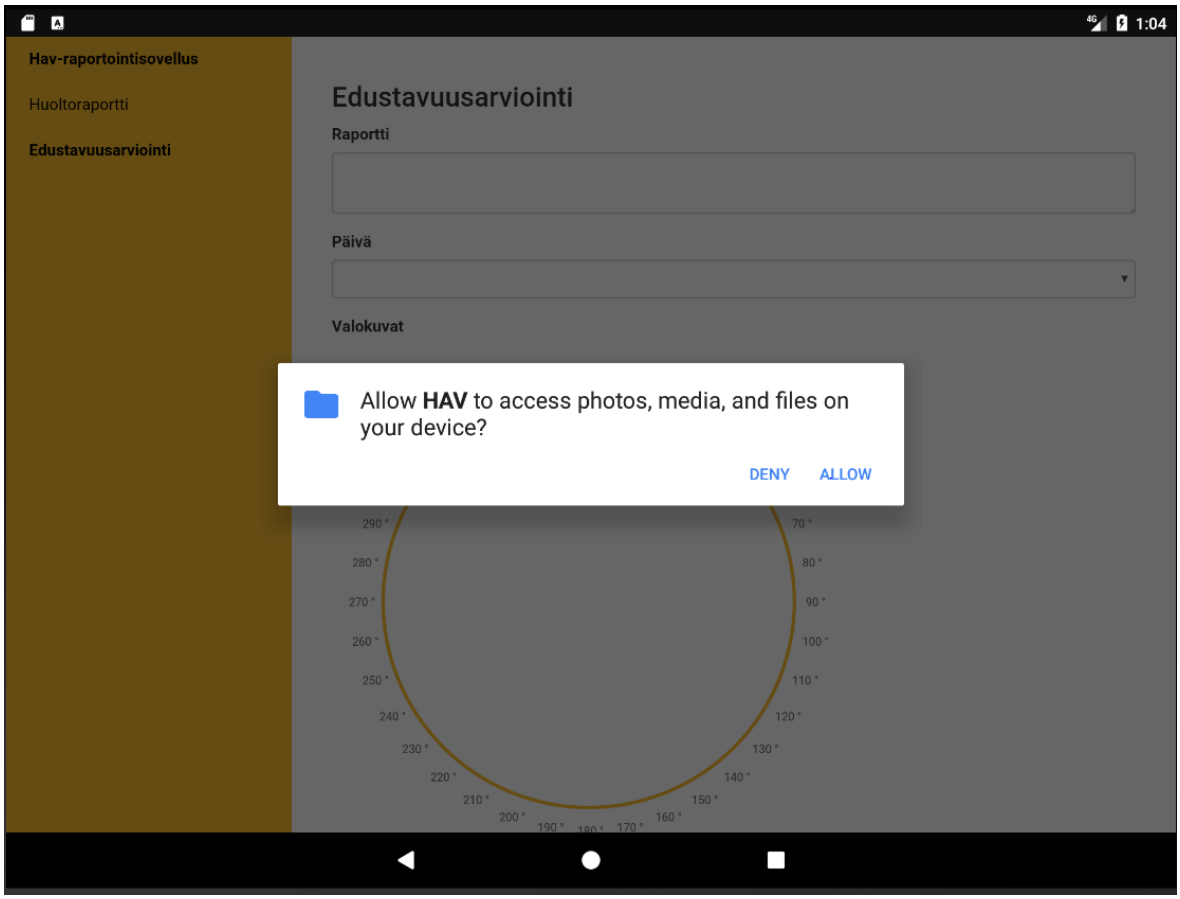
Kuva 10: Edustavuusarviointilomake

Teknisin lomake-elementeistä on 360 astetta näyttävä horisonttiympyrä, jolla jokaiseen asteeseen voidaan lisätä valokuva suoraan esim. tabletin kameralla otettuna tai jatkokehityksessä selaimen kautta drag & drop -menetelmällä (Kuva 11). Tabletilla astelukua klik-

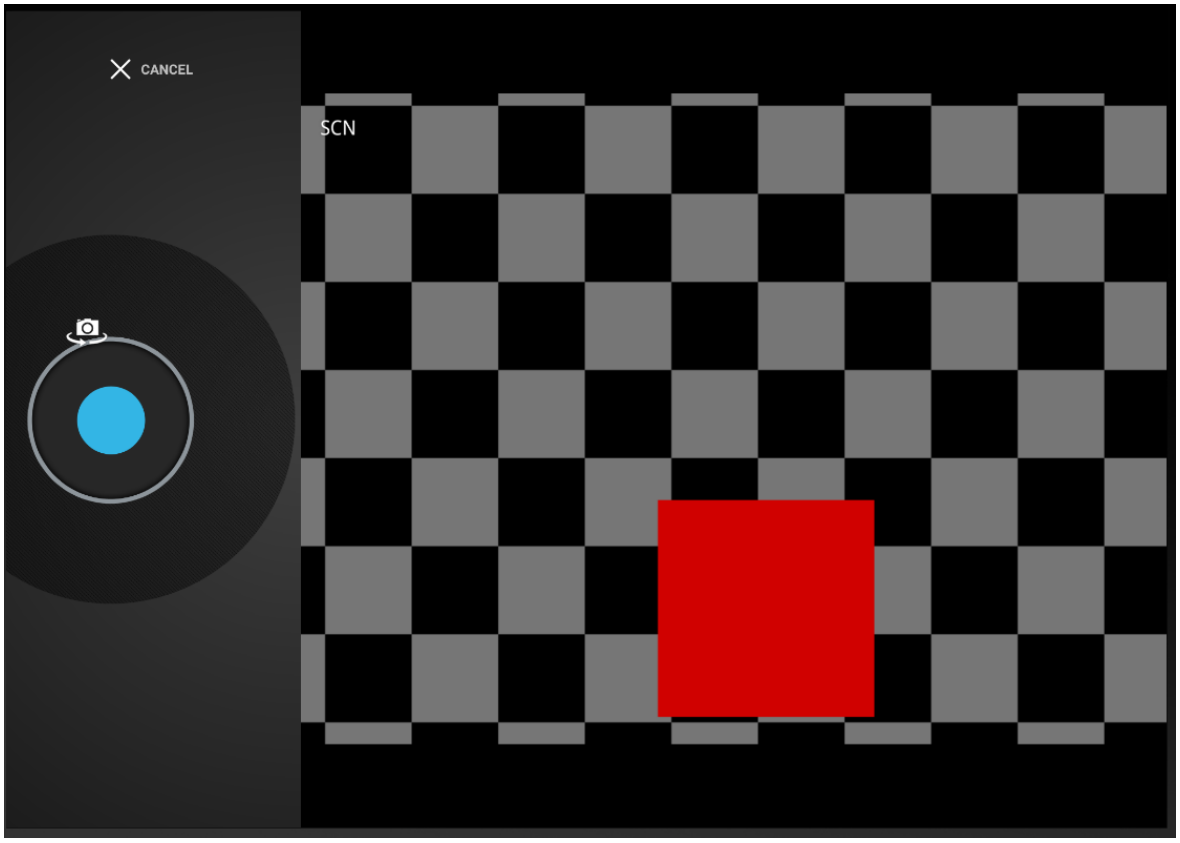
kaamalla aukeaa natiivi kamerasovellus (Kuva 13), jonka ottama kuva tallennetaan laitteen tiedostojärjestelmään odottamaan siirtoa rajapintapalvelimelle ja sidotaan tietokannassa oikeaan lomakkeeseen. Mikäli kameraa käytetään ensimmäistä kertaa, pyytää Android siihen luvan (Kuva 12).



Kuva 11: Valokuvien horisonttiympyrä -lomake-elementti



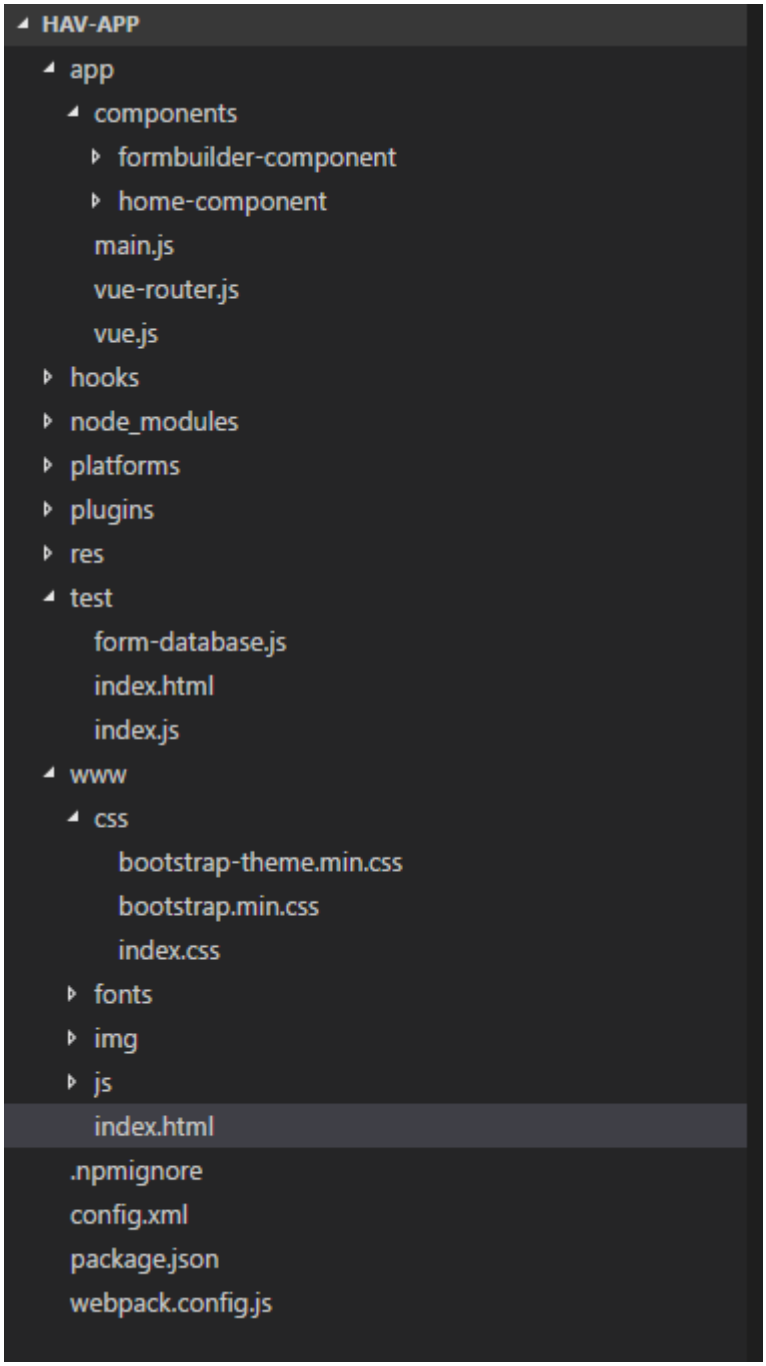
Kuva 12: Android pyytää kameran käyttöoikeutta



Kuva 13: Kamerasovellus Android-emulaattorissa

7.2.1 Sovelluksen kaaviot

Cordovan tiedostorakenne on hyvin yksinkertainen, sisältäen pienen määrän kansioita, jotka on jaettu Cordovan omiin kansioihin, Node.js node_modules-moduulikansioon sekä projektin toiminnallisuuden sisältäviin www-kansioon ja Vue.js-toiminnallisuuden app-kansioon (Kuva 14).

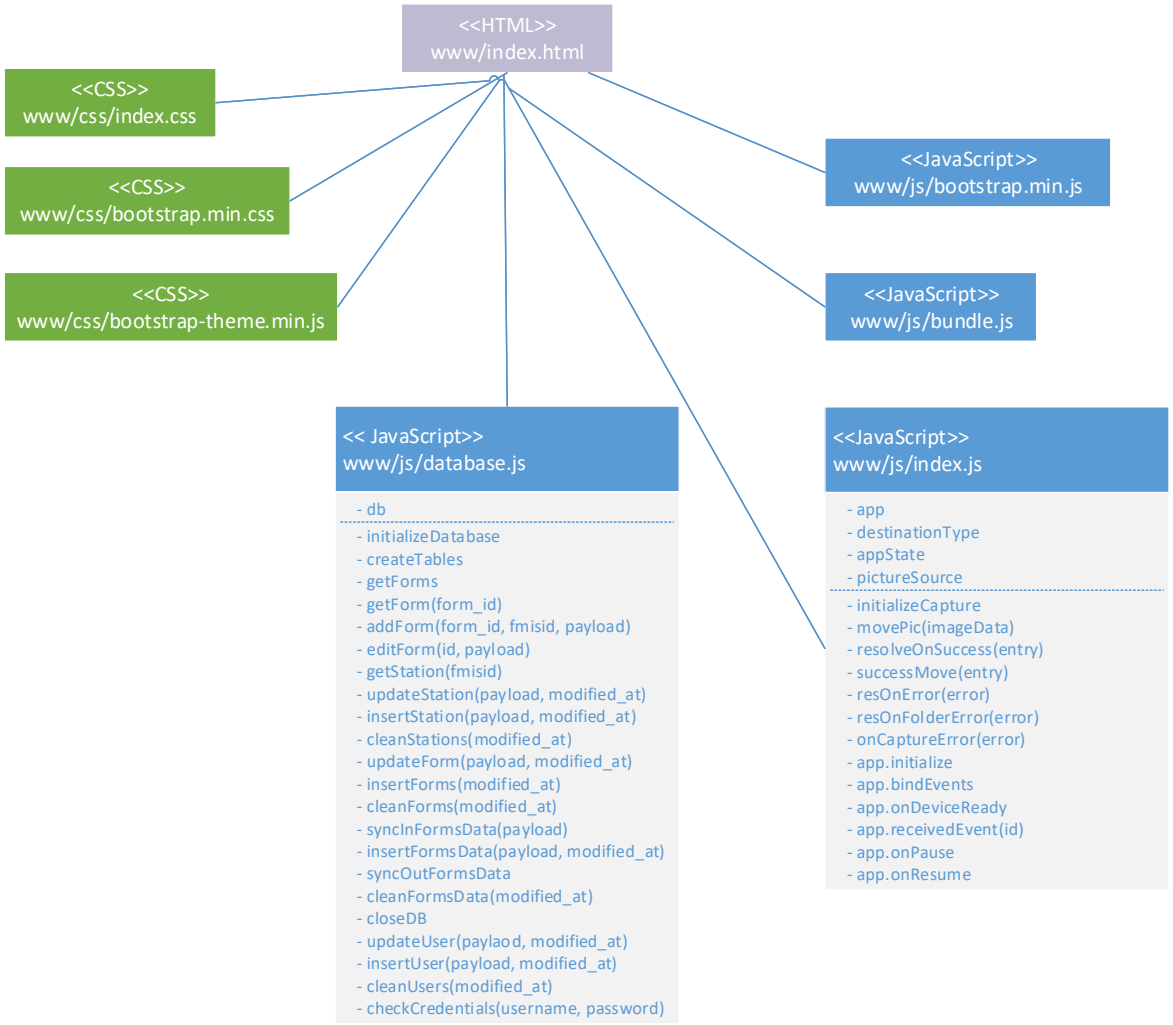


Kuva 14: Cordovan tiedostorakenne

7.2.2 Rakenne

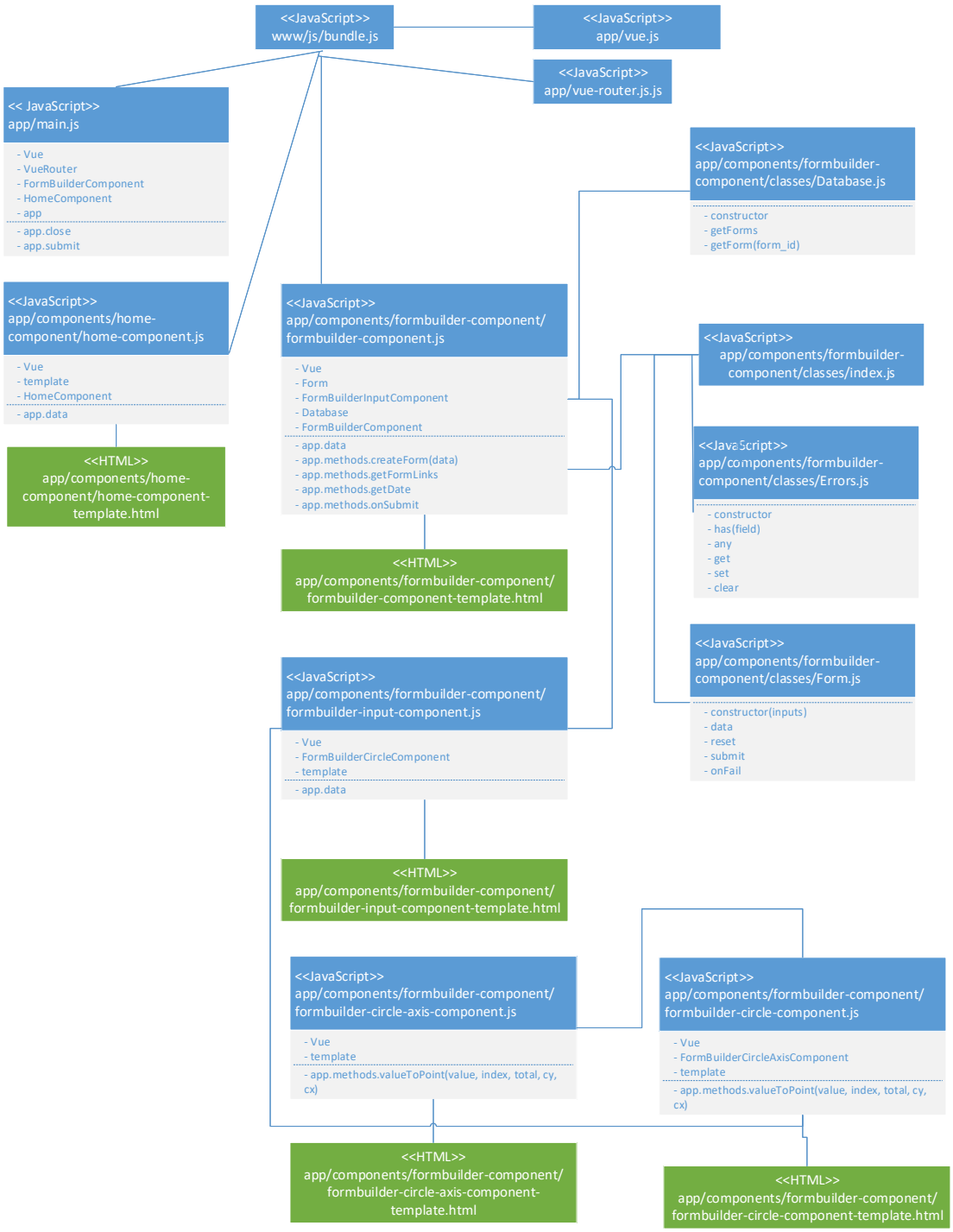
Kuva 13 kuvaa sovelluksen rakennetta. Sovelluksen pääkohta on index.html, joka sisällyttää itseensä CSS - sekä JavaScript-tiedostot. JavaScript-tiedostoissa on pääasiallinen sovelluslogiikka.

Index.js sisältää Cordovan vaatimat toiminnallisuudet, kuten kameran käsittelyn ja sovelluksen tilanseuraamisen. Database.js toteuttaa kaikki rajapinta- ja SQLite-toiminnallisuudet. Bundle.js on Webpackin koostama Vue.js-paketti (Kuva 15).



Kuva 15: Sovelluksen www-kansion rakenne

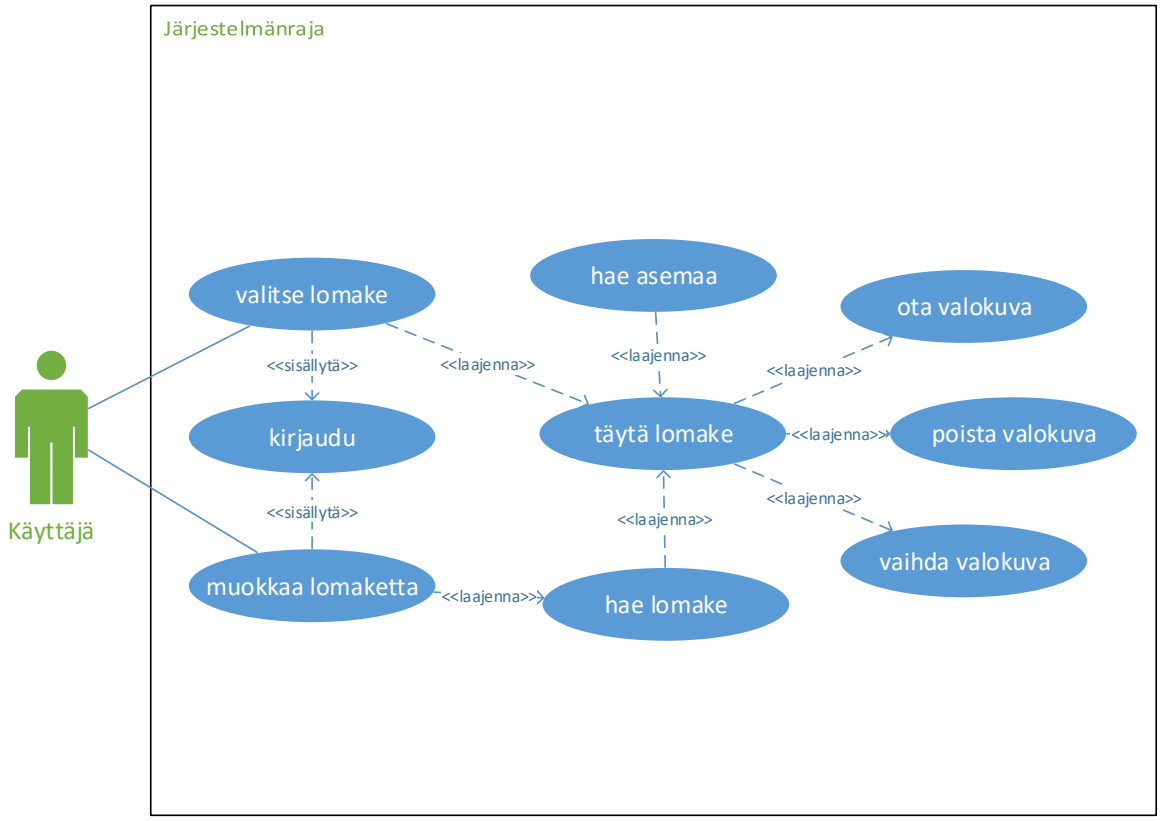
App-kansiossa olevat tiedostot (Kuva 16) toteuttavat Vue.js-toiminnallisuudet. Formbuilder-component toteuttaa lomakkeenrakentamisen ja Home-component puolestaan etusivun. Main.js hallitsee sivujen vaihtamisen, kirjautumisen sekä istunnontilaa. Vue.js sekä vue-router.js sisältävät vue.js tarvittavat toiminnot.



Kuva 16: Vue.js toiminnallisuuden rakenne

7.2.3 Käyttötapauskaavio

Alla on kuvattu sovelluksen käyttötapauskaavio (Kuva 17), jossa käyttäjä valitsee kirjautumisen jälkeen haluamansa lomakkeen, hakee siihen aseman ja täyttää sen. Mikäli lomakkeessa on horisonttiympyrä -lomake-elementti, myös kuvan ottaminen, poistaminen tai muuttaminen on mahdollista.



Kuva 17: Käyttötapauskaavio

8 Prototyypin toteutus

Prototyyppi sisältää dynaamisesti luotavat lomakekentät, mutta ei itse rajapintaa. Valmiustietojen synkronointi palvelimen kanssa on ohjelmoitu siihen pisteeseen, että rajapinnan lisääminen olisi mahdollista.

Prototyyppi hakee olemassa olevat lomakkeet sivuvalikkoon, josta navigoimalla generoidaan pyydetty lomake. Edustavuusarviointilomakkeella lisäksi horisonttiympyrä -lomake-elementillä otetut kuvat tallennetaan tiedostojärjestelmään.

9 Pohdinta

Opinnäyteprojektissa kehitettiin prototyyppi sovelluksesta, jolla saataisiin mahdollistettua nykyisten Excel-taulukoiden sijaan sähköinen järjestelmä havaintoaseman huolto- ja edustavuusarviointiin. Uuden järjestelmän hyödyt toimeksiantajan kannalta ovat manuaalisen työnmäärän vähentäminen ja laadukkaamman metatiedon kerääminen, jota voidaan hyödyntää tehokkaasti.

Opinnäytetyöprojektin aikana kohtasin haasteita selvittäessäni, kuinka moderneja verkkotekniikoita hyödynnetään nykypäivänä. Enää ei riitä pelkästään jQuery-kirjasto vaan tarvitaan useita työkaluja pelkästään kehittämisen tueksi. Yksinkertainenkin asia voi vaatia Node.js-ajoympäristön sekä lukemattoman määrän erilaisia paketteja kuten Babelin, Webpackin jne. Osaltaan tämä helpottaa kehitystyötä, mutta paikoitellen huomasin, että ongelmien selvittämiseen meni runsaasti aikaa eri pakettienversioiden yhteensopivuusongelmien johdosta.

Eri SPA-kehyksien vertailu ja tutkiminen olivat mielekästä, vaikka paikoitellen tuntuikin loputtomalta tehtävältä. Lopulta kuitenkin päädyin Vue.js-kehykseen johtuen sen yksinkertaisuudesta verrattuna Reactiin. Vue.js on konseptiltaan hyvin helposti omaksuttava, eikä sen sisäistämiseen mennyt kovinkaan kauaa. Suurin haaste oli sovelluksen tilan ylläpitämisessä, koska lomakkeen tiedot eivät säilyneet lähettämisen jälkeen, vaikka olisi ”pitänyt”. Ongelma kuitenkin ratkesi muuttaessani lomaketta Cordova yhteensopivaksi.

Sovelluksen kehittämisessä keskityin käyttäjäystävällisyyteen, joka koettiin ongelmaksi nykytilanteessa. Ulkoasussa hyödynnän pääosin Twitterin Bootstrap-tyylikirjastoa, joka mielestäni on hyvinkin selkeä käytettävyyttä ajatellen. Lomake-elementit ovat Bootstrapin mukaisia, mutta painikkeet muutin yksinkertaisemmiksi.

Kesken projektia, kun aloin yhdistää Vue.js toiminnallisuutta Cordovan toiminnallisuuteen huomasin, että Webpackin ja Babel tekniikoiden käyttäminen on käytännössä pakollista ja otin ne mukaan. Samalla kuitenkin rikoin horisonttiympyrä -lomake-elementin kameratoiminnallisuuden, joka tulee korjattavaksi jatkokehitykseen.

Itse projektin päätteeksi törmäsin mielenkiintoiseen näkökulmaan JavaScriptin suhteen ja olen aika pitkälle samaa mieltä: JavaScript-ekosysteemi kehittyy valtavaa vauhtia ja minkä opit viimeviikolla ei välttämättä ole enää merkityksellinen tällä viikolla. Omissa projekteissani olen miettinyt Node.js-ympäristön hyödyntämistä rajapintaan, mutta mitä enemmän asiaa mietin niin, sitä varmemmin pysyn Javan tai PHP:n parissa, mikäli rajapinta vaatii

monimutkaisuutta. Toki JavaScript ohjelmointikielenä kehittyy jatkuvasti, mutta koko ekosysteemi on toistaiseksi liian sekava, ainakin minulle, joka olen tottunut toisenlaiseen ohjelmointitapaan palvelinpuolella.

Toimeksiantajalle projektista syntyi prototyyppi, joka hyödyntää moderneja selaintekniikoita: JavaScript, CSS3 ja HTML5 sekä Cordova-ajoympäristöä, jonka avulla saadaan luotua natiiveja sovelluksia eri käyttöjärjestelmille. Suurin hyöty käytettyjen tekniikoiden johdosta on toimeksiantajalle ehdottomasti se, että projektin saattaminen loppuun ei vaadi eri ohjelmointikielien hallitsemista, jotta sovellus voidaan toteuttaa useisiin eri käyttöjärjestelmiin.

Projektin loppuun saattamisessa tuotantoon toimeksiantajan tulisi kehittää rajapinta, jonka kanssa sovellus keskustelee ja viimeistellä itse sovellus toimimaan yhteistyössä rajapinnan kanssa. Lähtisin itse toteuttamaan projektia eteenpäin toteuttamalla lomakkeeseen puuttuvat toiminnot, kuten virheidentarkistuksen sekä tallennuksen. Korjaisin myös horisonttiympyrä -lomake-elementin ja tiedostojärjestelmään tallentamisen sijaan tallentaisin kuvat base64-enkoodattuina SQLite-tietokantaan helpomman hallittavuuden johdosta. Seuraavaksi siirtyisin suunnittelemaan ja toteuttamaan prototyypin rajapinnasta, jotta kokonaisuudesta saataisiin täysin toimiva. Näin kokonaisuutta päästäisiin arvioimaan toiminnallisuuden perusteella ja päättämään sen jatkokehittämisestä tuotantoon asti.

Askeleet tuotantoon:

- Lomakkeeseen tallennus ja muokkaustoiminto sekä asemat
- Horisonttiympyrä -lomake-elementin toiminnallisuus kuntoon
- Lomakkeeseen virheentarkistus
- Rajapinnan suunnittelu
- Rajapinnan proto
- Rajapinnan ja sovelluksen yhdistäminen
- Puuttuvien ominaisuuksien toteuttaminen
- Testaaminen käytännössä
- Tuotanto

Ilmatieteen laitoksella on tarkoituksena lähteä jatkokehittämään järjestelmästä toimivaa prototyyppiä ja todennäköisesti olen sitä itse tekemässä kesällä 2017.

Lähteet

Babel. 2017. Luettavissa: https://babeljs.io. Luettu: 16.5.2017.

Bos, Bert. 2016. A brief history of CSS until 2016. Luettavissa: https://www.w3.org/Style/CSS20/history.html. Luettu: 23.5.2017.

Buckler, Graig. 2009. Progressive Enhancement and Graceful Degradation: an Overview. Sitepoint. Luettavissa: https://www.sitepoint.com/progressive-enhancement-graceful-degradation-basics/. Luettu: 4.10.2016.

Bootstrap. 2017. Luettavissa: http://getbootstrap.com/. Luettu: 16.5.2017.

Coligo.io. 2017. Build Vuex by Building a Notes App. Luettavissa: https://coligo.io/learn-vuex-by-building-notes-app/. Luettu: 17.5.2017.

Cordova Documents. Installing the Requirements. Luettavissa: https://cordova.apache.org/docs/en/latest/guide/platforms/android/index.html#installing-the-requirements. Luettu: 30.3.2017.

Dennis, Wixom, Roth. 2006. Systems Analysis and Design. Third edition. John Wiley and Sons co. Luettu: 30.3.2017.

Daliot, Amit. 2015. Functional Animation In UX Design. SmashingMaganzine. Luettavissa: https://www.smashingmagazine.com/2015/05/functional-ux-design-animations/#visual-hint. Luettu: 4.10.2016.

Heilmann, Christian. 2010. Local Storage And How To Use It On Websites. SmashingMaganzine. Luettavissa: https://www.smashingmagazine.com/2010/10/local-storage-and-how-to-use-it/. Luettu: 4.10.2016.

Hirsjärvi, Sirkka. 2009. Tutki ja kirjoita. Tammi. Luettu: 30.3.2017.

Ilmatieteen laitos. 2017. Luettavissa: http://ilmatieteenlaitos.fi/ilmatieteen-laitos. Luettu: 6.3.2017.

JavaScript Combability table. Luettavissa: http://kangax.github.io/compat-table/es6/. Luettu: 4.10.2016.

Jorgé. 2017. Vue.js vs React: Which one? React-etc.net. Luettavissa: http://react-etc.net/entry/vue-js-vs-react. Luettu: 17.5.2017.

Json Schema. 2017. Luettavissa: http://json-schema.org. Luettu: 6.3.2017.

JQuery. 2017. Luettavissa: https://jquery.com/. Luettu: 16.5.2017.

Kangax.github.io. 2017. Luettavissa: http://kangax.github.io/compat-table/es6/. Luettu: 17.5.2017.

Kava, Akash. Atom.js - CREATING RICH INTERNET WEB APPLICATIONS. Luettavissa: https://s3.amazonaws.com/webatoms/Documentation.pdf. Luettu: 4.10.2016.

Longman, Addison. 1998. A history of HTML. Luettavissa: https://www.w3.org/People/Raggett/book4/ch02.html. Luettu: 23.5.2017.

Material.io. Material design - Guidelines. Google. Luettavissa: https://material.io/guidelines. Luettu: 30.3.2017.

Mozilla Developer Foundation. Luettavissa: https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Using_HTML_sections_and_outlines. Luettu: 4.10.2016.

Nielsen, Jakob. 2012. Usability – 101. Luettavissa: https://www.nngroup.com/articles/usability-101-introduction-to-usability/. Luettu: 4.10.2016.

Node.js. Luettavissa: https://nodejs.org/en/. Luettu: 30.3.2017.

npmjs.org, 2017. Luettavissa: http://npmjs.org. Luettu: 30.3.2017.

Ullman, Chris. 2007. Beginning Ajax. John Wiley & Sons, Inc. Luettavissa: http://www.wrox.com/WileyCDA/Section/id-303217.html. Luettu: 30.3.2017.

Venzl, Gerald. 2015. Series: JSON within the Oracle Database - Introduction Luettavissa: https://blogs.oracle.com/developer/entry/json_support_in_oracle_database. Luettu: 30.3.2017.

Vuejs.org. 2017. Comparison with Other Frameworks. Luettavissa: https://vuejs.org/v2/guide/comparison.html. Luettu: 17.5.2017.

Wasson, Mike. 2013. ASP.NET - Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET. Microsoft Inc. Luettavissa: https://msdn.microsoft.com/en-us/magazine/dn463786.aspx. Luettu: 30.3.3017.

Webpack. 2017. Concepts. Luettavissa: https://webpack.js.org/concepts. Luettu: 16.5.2017.

What is NPM? 2017. Luettavissa: https://docs.npmjs.com/getting-started/what-is-npm. Luettu: 6.3.2017.

Liitteet

Liite 1. Määräaikaishuoltokysely

Haastateltava
Päivä

1. Miten asemanedustavuusraporttitäytetään?
2. Mitenraporttija siihenliittyvätmuutliedotviedäänjärjestelmään?
3. Mitenlaitepoistetaanasemalta?
4. Mitenlaitelisätäänasemalle?
5. Minkäkoetsuurimmaksihankaluudeksinykytilanteessa?
6. Mitkätoiminnotolisierityisentärkeääsaadaparemmiksi?
7. Mitämuitakehitysehdotuksiasinullaonliittyenhuoltoraportointiinjalaitehallintaan?

Liite 2. Aseman edustavuusarviointikysely

Haastateltava
Päivä

1. Mitä esitietoja tarvitaan ennen edustavuusarvioinnin tekoa?

2. Miten aseman edustavuusraportti täytetään?

3. Miten raportti ja siihen liittyvät muut tiedot viedään järjestelmään?

4. Minkä koet suurimmaksi hankaluudeksi nykytilanteessa?

5. Mitkä toiminnot olisi erityisen tärkeää saada paremmiksi?

6. Mitä muita kehitysehdotuksia sinulla on liittyen aseman edustavuusraporttiin?

Liite 3. Sovelluskoodit (salassa pidettävä)