Komponentin suunnittelu ja toteutus

YLVA-järjestelmä: Häiriintyvät kohteet -moduuli

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Hämeenlinnan korkeakoulukeskus
Tietojenkäsittelyn koulutusohjelma

Syksy, 2020

Jimi Koppinen

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutusohjelma
Hämeenlinnan korkeakoulukeskus

Tekijä Jimi Koppinen **Vuosi** 2020

Työn nimi Komponentin suunnittelu ja toteutus: YLVA-järjestelmä – Häiriintyvät kohteet -moduuli

Työn ohjaaja/t Lasse Seppänen

TIIVISTELMÄ

Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa Ympäristönsuojelun valvonnan sähköiseen asiointijärjestelmään asiakkaan tilaama Häiriintyvät kohteet -moduuli sekä dokumentoida suunnittelu- ja toteutusprosessi. Häiriintyvät kohteet -moduuli on osa laajempaa rekisteröinti-ilmoitusta, josta säädetään ympäristönsuojelulaissa. Työn tarkoituksena on tuottaa uutta julkista tietoa modernin web-asiointipalvelun toteuttamisesta julkishallinnolle. Toimeksiantajana toimi Alfame Systems Oy.

Ympäristönsuojelun valvonnan sähköinen asiointijärjestelmä – YLVA on suunnattu ympäristönsuojelulain nojalla lupa-, ilmoitus- ja rekisteröintivelvollisille sekä jätelain nojalla ilmoitus- ja rekisteröintivelvollisille asiakkaille. Palvelun käyttäjiä ovat muun muassa toiminnanharjoittajat, kuntien ympäristöviranomaiset ja ELY-keskusten asiantuntijat.

Teoriaosuuteen kerättiin aineistoa internetlähteistä, ohjelmointitekniikoiden virallisista dokumentaatioista sekä alan kirjallisuudesta. Käytännön osuudessa kuvattiin komponentin toteutusta kuvakaappausten ja koodiesimerkkien avulla. Suunnittelun käytännön osuudessa hyödynnettiin määrittelypalaverimuistioita sekä asiakkaan avainhenkilöiden haastatteluita. Työn lopussa arvioitiin toteutettua komponenttia ja peilattiin lopputuotetta tutkimuskysymyksien valossa.

Työn lopuksi toteutettiin asiakkaan tilaama modernilla web-ohjelmointitekniikalla toteutettu single page application -moduuli asiakkaan asiointijärjestelmään.

Avainsanat Ohjelmistokehitys, sähköinen asiointi, web-ohjelmointi, ohjelmistosuunnittelu, ReactJS

Sivut 45 sivua, ei liitteitä

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology
Hämeenlinna University Centre

Author Jimi Koppinen **Year** 2020

Subject Planning and development of a digital form component – YLVA system

Supervisors Lasse Seppänen

ABSTRACT

The goal for the thesis was to plan and develop a form module for the client's digital transaction service named Ympäristönsuojelun valvonnan sähköinen asiointijärjestelmä. The goal was to document the process. The name of the module is Häiriintyvät kohteet and it is a part of a larger registration notice process regulated by the environmental protection law. One of the objectives was to provide new public information on developing a modern web service for public administration.

Ympäristönsuojelun valvonnan sähköinen asiointijärjestelmä – YLVA is a system that is targeted for entities that are required under the environmental protection law to file an environment permit or for customers who are obliged to make a registration notice. Users of the system include for example operators, environmental authorities of municipalities and the experts of the centre for economic development.

The theoretical framework was compiled from the official documentation of programming techniques and frameworks. Theoretical part also includes sources from the internet and literary sources regarding the framework. Legislative literature was also used and an interview with one of the project's key persons was conducted. The development and planning process were documented by using code snippets and screenshots of the user interface of the web service in the practical section of the thesis. In the final chapter the completed module was evaluated in the light of the research questions. In the end, a ready single page application form module was provided for the client using modern web application techniques. The client for the thesis was Alfame Systems Oy.

Keywords Web application, Software development, electronic services, ReactJS

Pages 45 pages, no appendices

SANASTOA

API	Ohjelmointirajapinta
Back-end	Sovelluksen taustaprosessit ja liiketoimintalogiikka
CSS	Cascading style sheet, kaskadiset tyyliohjeet
DX	Developer experience, UX:n vastine sovelluskehittäjälle
DOM	dokumenttioliomalli, tapa kuvata rakenteisen dokumentin, kuten HTML:n rakenne puuna, jonka eri olioita voi hakea, tutkia tai manipuloida esim. JavaScriptin avulla
Front-end	Sovelluksen käyttöliittymäkerros
REKI	Ympäristönsuojelulain mukainen rekisteröintimenettely
Scrum	Projektinhallinnan viitekehys, jota käytetään yleisesti ketterässä ohjelmistokehityksessä
SPAv2	Aluehallinnon sähköinen asiointipalvelu. Käytetään myös nimellä sähköinen palvelualusta. Tätä palvelua käyttämällä toiminnanharjoittajat täyttävät ja lähettävät lomakkeita viranomaisille käsiteltäväksi.
YLVA	Ympäristönsuojelun valvonnan sähköinen asiointijärjestelmä. YLVAlla viitataan tässä työssä käsittelyjärjestelmään, jota viranomainen käyttää saapuneiden lomakkeiden käsittelemiseen.
UX	Sovelluksen tai käyttöliittymän käyttökokemus

SISÄLLYS

1 JOHDANTO

Työn tavoitteena oli suunnitella ja toteuttaa käyttöliittymäkomponentti KEHA-keskukselle (Suomen valtion virasto, joka tuottaa ELY-keskusten ja TE-toimistojen kehittämis- ja hallintopalveluja) alihankintana Alfame Systems Oy:n toimesta osaksi jo toimivaa Ympäristönsuojelun valvonnan sähköistä asiointijärjestelmää eli YLVA-järjestelmää.

YLVA-järjestelmää kehitetään jatkuvasti. Kesällä 2020 YLVA-järjestelmään kehitetään uusia ominaisuuksia, joista yksi on ympäristönsuojelulain mukaisen rekisteröintimenettelyyn liittyvien lomakkeiden digitalisointi. Asiakasorganisaation tilaama komponentti on lomakemoduuli, joka tulee osaksi REKI-lomakekokonaisuutta, jolla rekisteröintivelvollinen ilmoittaa rekisteröintimenettelyn mukaisesta toiminnastaan sähköisesti. Viranomainen taas käsittelee vastaavan lomakkeen käsittelyjärjestelmässä. Tilatun moduulin työnimi on Häiriintyvät kohteet.

Lomakemoduulin suunnittelussa ja toteutuksessa tulee ottaa huomioon jo olemassa olevassa tietojärjestelmässä oleva järjestelmäarkkitehtuuri sekä muut tekniset ratkaisut. Tämän vuoksi esimerkiksi käyttöliittymäkirjastona tullaan käyttämään ReactJS -JavaScript -kirjastoa sekä siihen liittyviä lisäosia. Prosessointi- ja back end -viitekehyksenä käytetään .NET Frameworkia ja siihen liittyviä apukirjastoja.

Opinnäytetyössä kuvataan suunnittelu- ja toteutusprosessia erilaisin kaavioin, koodiesimerkein, haastattelu- ja kokousmuistiinpanoin sekä kuvakaappauksin. Suunnittelu- ja toteutusprosessissa otetaan huomioon komponentin uudelleenkäyttömahdollisuudet YLVA-tietojärjestelmässä. Työn toteutuksessa ja suunnittelussa pyritään käyttämään sovelluskehitykseen liittyvää abstraction principle -periaatetta. (Pierce, 2002, s. 340.) Periaatteen ideana on pyrkiä välttämään tarpeetonta koodin kopioimista ohjelmiston kehittämisessä. Työn tavoitteena on kuvata suunnittelu- ja toteutusprosessia mahdollisimman avoimesti, jotta opinnäytetyö tarjoaa uutta tietoa modernin web-palvelun toteuttamisesta julkishallinnon organisaatioille.

Työ vastaa seuraaviin kysymyksiin:

- Mitä asioita pitää ottaa huomioon olemassa olevan, osittain vanhentuneella tekniikalla toteutetun järjestelmän ja uuden komponentin yhteensopivuudessa?
- Mitä lisäarvoa tuottavia uusia tekniikoita voidaan hyödyntää komponentin toteutuksessa?
- Miten rakennetun komponentin osia voidaan uudelleenkäyttää tässä järjestelmässä tai muissa asiakkaan järjestelmissä?

2 YLVA-TIETOJÄRJESTELMÄ

Ympäristönsuojelun valvonnan sähköinen asiointijärjestelmä YLVA on viranomaisten käyttämä tietojärjestelmä. Ympäristö.fi-sivustolla kerrotaan YLVAsta seuraavasti: ”YLVA on suunnattu ympäristönsuojelulain nojalla lupa-, ilmoitus- ja rekisteröintivelvollisille sekä jätelain nojalla ilmoitus- ja rekisteröintivelvollisille asiakkaille. Palvelua käyttävät muun muassa toiminnanharjoittajat, kuntien ympäristöviranomaiset ja ELY-keskusten asiantuntijat.” (Ympäristöhallinnon yhteinen verkkopalvelu, 2013). Tässä luvussa käsitellään YLVA-tietojärjestelmää ja siihen liittyvien muiden järjestelmien kokonaisuutta. Luvussa käydään myös läpi lainkohtia, jotka asettavat reunaehtoja ja vaatimuksia ohjelmiston kehittämiselle.

2.1 Sähköisen asiointijärjestelmän yleiskuvaus

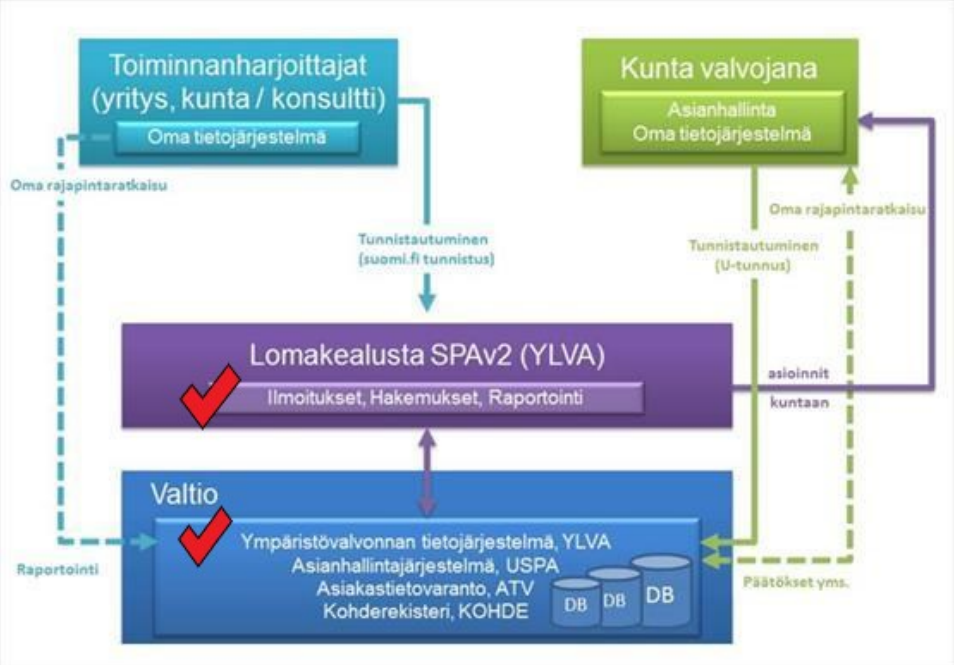
Kuva 1. YLVA ja siihen liittyvien muiden tietojärjestelmien kokonaisuus

YLVA on yksi osa ympäristöhallinnon järjestelmäkokonaisuutta, johon kuuluvat perusYLVAn lisäksi maaperäntilan tietojärjestelmä, jätehuoltorekisteri ja eLUPA (sähköinen vesi- ja ympäristölupa). Kyseessä on siis erittäin laaja järjestelmäkokonaisuus, joka koostuu useasta eri osa-alueesta. Kuvasta 1 selviää tietojärjestelmien kokonaisuus.

Järjestelmä voidaan jakaa kahteen pääosa-alueeseen. Ensimmäinen osa-alue on Aluehallinnon asiointipalvelu (https://sahkoinenasiointi.ahtp.fi/fi) jota muun muassa toiminnanharjoittajat käyttävät toimintansa raportoimiseen sekä ympäristölupien hakemiseen. Toinen osa-alue on käsittelyjärjestelmä, jota eri viranomaisvalvojat käyttävät eri toimenpiteiden, kuten

ympäristölupien päätösten tekemiseen. Näiden lisäksi kokonaisuuteen integroituu myös muita järjestelmiä kuten USPA-asianhallintajärjestelmä.

Opinnäytetyön aiheena oleva lomakemoduuli tulee olemaan osa ympäristönsuojelulain mukaisen rekisteröintimenettelyyn liittyvien lomakkeiden digitalisointia. Lomakemoduulia tullaan hyödyntämään sekä asiointijärjestelmän että käsittelyjärjestelmän puolella. Kuvaan 2 on merkattu punaisella tähdellä ne kohdat järjestelmäkokonaisuudessa, johon moduuli toteutetaan.



Kuva 2. Kuva tietojärjestelmän tietovirroista (Ympäristöhallinnon yhteinen verkkopalvelu, 2017)

2.2 Viranomaisvaatimukset ja lainsäädännön vaikutus suunnitteluun

Opinnäytetyötä varten haastateltiin Hämeen ELY-keskuksen projektipäällikköä Päivi Laurilaa, joka toimii myös YLVA-projektin projektipäällikkönä. Hän kertoi viranomaisvaatimuksista sekä lainsäädännöstä, jotka ohjaavat vahvasti YLVAn uusien ominaisuuksien kehittämistä. Projektipäällikkö Päivi Laurilan (haastattelu 28.5.2020) mukaan viranomaistoiminnassa kaiken kehitystyön pohjalla on lainsäädäntö. Hänen mukaansa suurimmassa roolissa ohjelmiston kehittämis- ja määrittelytyössä ovat ympäristönsuojelulaki ja sen säädökset. Laurilan mukaan säädökset ja asetukset asettavat omat vaatimuksensa YLVA-tietojärjestelmän kehittämiseen.

Tässä luvussa eritellään komponentin suunnitteluun ja toteuttamiseen liittyvät keskeisimmät lainkohdat. Luvussa käsitellään eritoten rekisteröintimenettelyä ja sen vaikutusta REKI-lomakkeiden suunnitteluun.

Rekisteröintimenettelystä säädetään ympäristönsuojelulaissa. Siinä mainitaan valtioneuvoston asetukset ympäristön pilaantumisen ehkäisemiseksi eräissä toiminnoissa. Pykälässä käydään läpi toiminnot/toimialat, jota asetus koskee. Tässä asetuksessa määrätään toiminnanharjoittajien rekisteröintivelvollisuudesta. Esimerkiksi uusi asfalttiasema on velvollinen rekisteröimään toimintansa ennen toimintansa aloittamista. (Ympäristönsuojelulaki 527/2014 § 10)

Ympäristönsuojelulaissa käsitellään toiminnan sijoituspaikan valintaa seuraavasti: ”Ympäristön pilaantumisen vaaraa aiheuttava toiminta on mahdollisuuksien mukaan sijoitettava siten, että toiminnasta ei aiheudu pilaantumista tai sen vaaraa ja pilaantuminen voidaan ehkäistä. Toiminnan sijoituspaikan soveltuvuutta arvioitaessa on otettava huomioon toiminnan:

1) luonne, kesto, ajankohta ja vaikutusten merkittävyys sekä pilaantumisen todennäköisyys ja onnettomuusriski;
2) vaikutusalueen herkkyys ympäristön pilaantumiselle;
3) merkitys elinympäristön terveellisyyden, ja viihtyisyyden kannalta;
4) sijoituspaikan ja vaikutusalueen nykyinen ja oikeusvaikutteisen kaavan osoittama käyttötarkoitus;
5) muut mahdolliset sijoituspaikat alueella.” (Ympäristönsuojelulaki 527/2014 § 11)

Yllä olevassa käydään läpi kohteen sijoituspaikan valintaa. Sijoituspaikan valinta -osio tullaan nimeämään uudella digitalisoidulla REKI-lomakkeella Häiriintyvät kohteet -osioksi. Siihen viitataan tässä opinnäytetyössä myös Häiriintyvät kohteet -moduulina.

Ympäristönsuojelulaissa kerrotaan toiminnan rekisteröinnistä seuraavasti: ”Tämän lain liitteessä 2 säädetystä ympäristön pilaantumisen vaaraa aiheuttavasta toiminnasta on tehtävä rekisteröinti-ilmoitus kunnan ympäristönsuojeluviranomaiselle ympäristönsuojelun tietojärjestelmään rekisteröintiä varten.” (Ympäristönsuojelulaki 527/2014 §116) Viranomaiset ovat tätä varten rakentaneet YLVA-tietojärjestelmän. Tätä rekisteröintiä varten REKI-lomakkeet suunnitellaan ja toteutetaan.

Ympäristönsuojelun tietojärjestelmästä kerrotaan ympäristönsuojelulaissa seuraavaa: ” Ympäristöä ja siihen vaikuttavia toimintoja koskevia tietoja varten on ympäristönsuojelun tietojärjestelmä. Sitä käytetään ympäristönsuojeluun liittyvien tietojen hallintaan ja käsittelyyn, ympäristölainsäädännön valvonnan toteuttamiseen, ympäristön tilan seurantaan sekä ympäristöön liittyvään tutkimukseen ja suunnitteluun.” (Ympäristönsuojelulaki 527/2014 § 222) Tällä tietojärjestelmällä viitataan muun muassa YLVA-tietojärjestelmään.

”Rekisteröitäviä toimintoja ovat ympäristönsuojelulain liitteessä 2 mainitut polttoaineteholtaan alle 50 MW:n energiantuotantolaitokset, nestemäisten polttoaineiden jakeluasemat, asfalttiasemat, kemialliset pesulat

ja orgaanisia liuottimia käyttävät toiminnat sekä kiinteät betoniasemat ja betonituotetehtaat.” (Ympäristöhallinnon yhteinen verkkopalvelu, 2013) Nämä toiminnat eivät ole ympäristönsuojelulain mukaisesti ympäristölupavelvollisia. Kyseessä on rekisteröintimenettely. Tätä rekisteröintimenettelyä kuvataan tässä opinnäytetyössä ja asiakasorganisaatiossa REKI-menettelyksi.

3 KETTERÄT MENETELMÄT VAATIMUSMÄÄRITTELYSSÄ JA OHJELMISTOKEHITYKSESSÄ

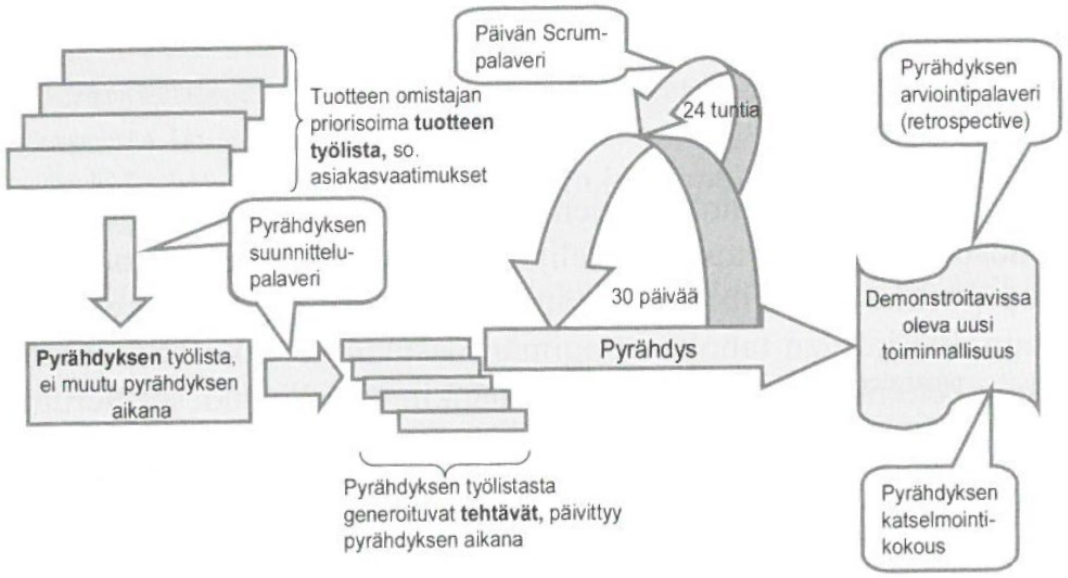
Ohjelmistokehityksessä käytetään usein jotakin projektinhallintamenetelmää tukemaan varsinaista kehitystyötä. YLVAn kehitystyön osalta sovelletaan ketteriä menetelmiä, joista etenkin Scrum-prosessia sovelletaan suurelta osin.

Yli 60-prosentissa epäonnistuneista ohjelmistoprojekteista syynä on vaatimustenhallintaan liittyvät puutteet (Haikala & Mikkonen, 2011, s. 61). Tämän takia on erittäin tärkeää, että vaatimusmäärittely ohjelmistoprojektissa tehdään huolellisesti. Esimerkiksi vesiputousmallissa ja muissa perinteisemmissä projektinhallintamalleissa vaatimusmäärittely saatetaan tehdä jopa erillisenä projektina ja on näin ollen suhteellisen raskas prosessi. Tällaisissa projekteissa vaatimusten muutosprosessin hallitseminen on tärkeää.

Ketterissä menetelmissä uusia muutoksia vaatimuksiin saattaa tulla projektin elämänkaaren aikana useastikin. Tällaisissa tilanteissa muutokset käsitellään uusina vaatimuksina ja asetetaan tuotteen työlistalle. Pyrähdyksen suunnittelu -tapahtumassa tuoteomistaja esittelee tuotteen työlistaa ja Scrum-tiimi voi taas valita näitä uusia vaatimuksia pyrähdyksen työlistalle. Näitä tehtäviä toteutetaan seuraavassa pyrähdyksessä. Scrumissa tuoteomistaja on vastuussa vaatimustenhallinnasta. (Haikala ym., 2011, s. 49)

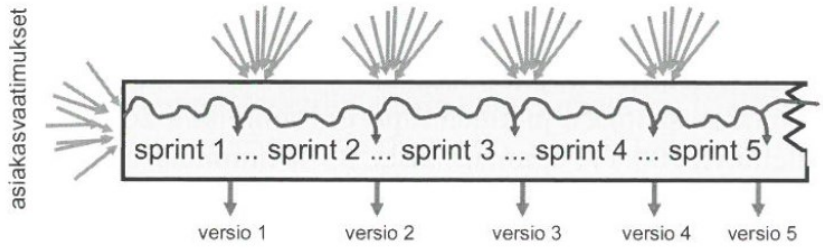
Ketterien menetelmien soveltaminen esimerkiksi vaatimusten määrittelyssä voi ketteröittää prosessia ja voi parantaa asiakastyytyväisyyttä. Usein perinteisen vaatimusmäärittelyn haasteena on, ettei liiketoiminnan johto osaa määritellä, mitkä ohjelmiston ominaisuudet ovat oikeasti tärkeitä ja tämän vuoksi perinteiset prosessit ovat jäykkiä. (Auer ym., 2013, s. 27)

Ketterissä menetelmissä projekti jaetaan usein pieniin osaprojekteihin, kuten Scrumissa pyrähdyksiin, ja uusia ominaisuuksia pystytään toteuttamaan nopeallakin aikataululla asiakkaan nähtäväksi. Lopputulos saatetaan nähdä nopeasti määrittelyn jälkeen. Tämä mahdollistaa nopean palautteenannon käyttäjiltä kehittäjille ja näin ollen säästää aikaa ja kustannuksia. (Myllymäki, Hinkka, Hirvensalo & Hämäläinen, 2015, s.86)



Kuva 3. Kuvaus Scrum-prosessista (Haikala & Mikkonen, 2011, s. 48)

Kuvassa 3 on tiivis kuvaus Scrum-viitekehityksen mukaisesta pyrähdyksestä ja siihen liittyvistä muista asioista, kuten palaverit ja kokoukset. Pyrähdys alkaa suunnittelupalaverilla, johon osallistuvat kaikki Scrumiin kuuluvat: tuoteomistaja, Scrum-mestari ja Scrum-tiimi. Siinä valitaan pyrähdyksen aikana toteutettavat tehtävät. Pyrähdyksen aikana, joka päivä, käydään läpi Scrum-palaveri, jossa jokainen tiimiläinen kertoo, mitä on tehnyt edellisen päivän aikana, mitä aikoo tehdä tulevan päivän aikana ja jos tekemiselle on jotain esteitä. Scrum-mestarin tehtävänä on poistaa tiimin mahdolliset esteet. Pyrähdyksen lopussa on katselmointikokous, jossa esitellään uusia toiminnallisuuksia asiakkaalle. Tämän jälkeen pidetään pyrähdyksen arviointipalaveri, jossa arvioidaan sitä, mikä meni pyrähdyksessä hyvin ja sitä, mitä voidaan kehittää. Seuraavaksi aloitetaan uusi pyrähdys ja prosessi alkaa alusta.



Kuva 4. Asiakasvaatimukset Scrum-mallissa (Haikala & Mikkonen, 2011, s. 51)

Kuvassa 4 on Haikalan ja Mikkosen näkemys Scrum-mallissa tulevista asiakasvaatimuksista. Scrum-filosofian mukaan pyrähdyksen/sprintin aikana ei sprintin työlistaa ei muuteta, vaikka uusia vaatimuksia tulisi. Tämän

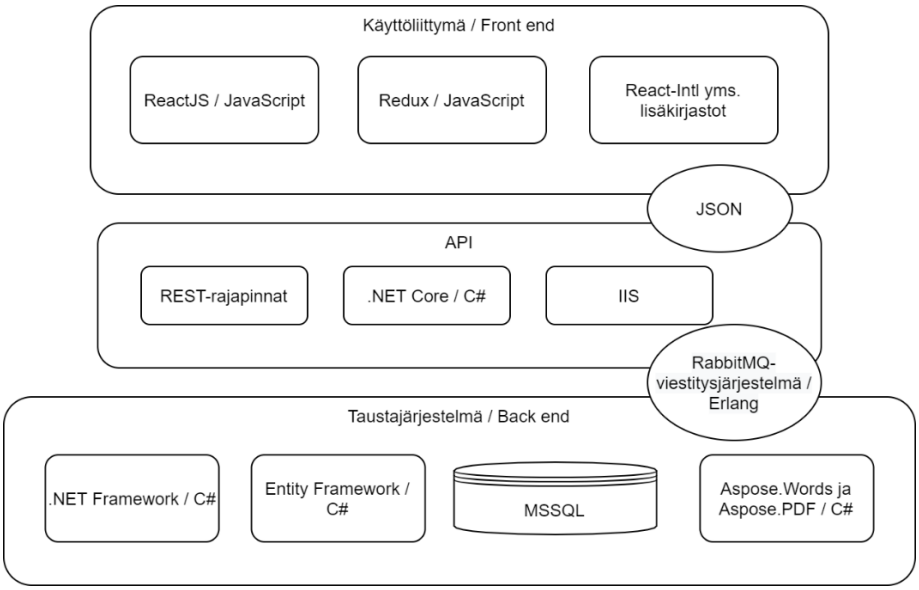
tarkoituksena on taata kehittäjätiimille työrauha ja jotta tiimi voi sitoutua alkuperäisen pyrähdyksen tavoitteen saavuttamiseen. Oikeassa elämässä tilanne ei kuitenkaan aina näin ole. Aika ajoin voi olla asioita, joita täytyy vain nostaa pyrähdykseen mukaan asiakkaan vaatimuksesta. Onhan asiakas kuitenkin se, jolle tuotetta tehdään.

4 KÄYTETTÄVÄT TEKNIIKAT

Toteutettavassa komponentissa käytetään useita eri ohjelmistotekniikoita ja ohjelmointikieliä. Seuraavaksi luetellaan toteutuksessa käytettävät tekniikat ja osittain syyt niiden valintojen takana. Suurin syy tekniikkavalintojen taustalla on se, että suurimpaan osaan järjestelmän kirjoittamisessa on käytetty jo tiettyä kieltä. Nämä valinnat on tehty jo aiemmin, kun koko asiointialustan (https://sahkoinenasiointi.ahtp.fi/fi) arkkitehtuuria on suunniteltu. Sähköistä palvelualustaa (SPAv2) käyttävät myös monet muut yhteisöt, kuten TUKES ja TE-toimistot. Täytyy siis ottaa huomioon, että käytettävät tekniikat ovat yhteneväisiä järjestelmän toimivuuden ja jatkuvuuden kannalta. Myöhemmässä vaiheessa tehdään arvio siitä, että voisiko komponentin toteuttamisessa hyödyntää jossain määrin uusia tekniikoita ja toisiko sellaisten käyttö lisäarvoa asiakkaalle.

Sekä käsittely- että asiointijärjestelmässä hyödynnetään SPA (Single-page-application) tekniikkaa. SPA tarkoittaa www-sivua, joka toimii yhden palvelimen tarjoaman sivun kontekstissa. Nettisivu näyttää yleensä sovellukselta. Sivu ei siis sisällä palvelimen tarjoamia useampia sivuja. Kuitenkin SPA voi sisältää useita sivunäkymiä, joiden välillä liikutaan sivun navigaation kautta. Eri näkymiä pyydetään selaimessa suoritettavan sovelluskehyksen sisäisen reitityslogiikan kautta. Vaikka URL osoiterivillä muuttuu, niin palvelimelta ei tarjoilla uutta sivua vaan käytetään samaa jo ladattua sivua.

Kuvassa 5 kuvataan hyvin yleisellä tasolla tärkeimmät käytettävät tekniikat painottaen käyttöliittymän osuutta, joka on tässä työssä suurimmassa osuudessa. Vaikka käsittelyjärjestelmä ja asiointijärjestelmä ovat kaksi erillistä järjestelmää, niin ne hyödyntävät suuressa määrin samoja tekniikoita. Tästä on synergiaetua jatkokehityksen kannalta.



Kuva 5. Päätekniikat

4.1 **JavaScript**

Toteutettavassa moduulissa käytetään tekniikkana JavaScriptiä, jota käytetään pääasiassa käyttöliittymässä käytettävien ReactJS-komponenttien koodaamisessa. Jonkin verran käytetään myös niin sanottua ”puhdasta” JavaScriptiä ilman mitään ulkoista lisäkirjastoa. Tämä on kuitenkin satunnaista.

JavaScript on olio-orientoitunut skriptauskieli, jota käytetään nettisivujen muuttamiseksi interaktiiviseksi. Interaktiivisuudella tarkoitetaan esimerkiksi animaatioita, painettavia nappeja, popup-menuja ja muita toiminnallisuuksia, joita voidaan lisätä nettisivuille. JavaScriptiä käytetään pääosin asiakaspuolen eli internet-selaimen päässä. Sitä käytetään jonkin verran myös palvelinpuolen kielenä. (MDN Web Docs, 2020)

4.2 **ReactJS**

Käyttöliittymäkomponenttien toteutuksessa käytetään pääasiallisesti ReactJS-kirjastoa. ReactJS on alun perin Facebookin kehittämä JavaScript-kirjasto käyttöliittymien kehittämistä varten (W3Schools, 2020).
Reactista käytetään joskus myös nimeä React.js tai ReactJS. React on avointa lähdekoodia. Sen tehtävänä on muuttaa applikaatiossa tai ohjelmistossa käytettävä tietorakenne loppukäyttäjälle näkyvään muotoon. (Honkanen, 2017)

Reactia käytetään tällä hetkellä hyvin useissa suurissa yrityksissä. Muun muassa Facebook, Instagram, Paypal, Uber, Netflix ja Twitter käyttävät Reactia tällä hetkellä käyttöliittymissään. Reactia käytetään PWA (progressive web application) ja SPA (single-page-application) sovellusten tekemiseen. Single page application on sovellus, joka perustuu liiketoimintalogiikaltaan selaimen päähän ja muodostuu yhdestä HTML-dokumentista usean sijaan. Loppukäyttäjä ei siis oikeasti navigoi sivulta toiselle, vaikka selainrivillä URL-osoite muuttuukin. (Spec India, Medium.com, 2020)

Reactilla pystytään rakentamaan monistettavia uudelleenkäytettäviä käyttöliittymäkomponentteja, joita voidaan kuvata esimerkiksi lohkoiksi. Näitä lohkoja käyttämällä voidaan rakentaa toiminnallinen käyttöliittymä. Näitä komponentteja voidaan ajatella funktioina. Funktioita voidaan uudelleenkäyttää ja niistä voidaan koota suurempia funktioita tarvittaessa. Erotuksena ”normaaleihin” funktioihin nähden on se, että komponentit voivat sisältää komponentin oman sisäisen tilan (state). Tila on JavaScript-olio, joka sisältää komponentin omaa dataa. (freeCodeCamp, 2017).

4.2.1 JSX-syntaksi

Reactin syntaksia kutsutaan JSX:ksi. JSX on XML:n ja HTML:n tapainen syntaksiltaan. JSX:n avulla voidaan kirjoittaa React-komponentteja hyödyntäen XML:stä tuttua puurakennetta. Vaikka JSX muistuttaa HTML:ää ja XML:ää, niin se on JavaScriptiä. (reactjs.org, n.d.)

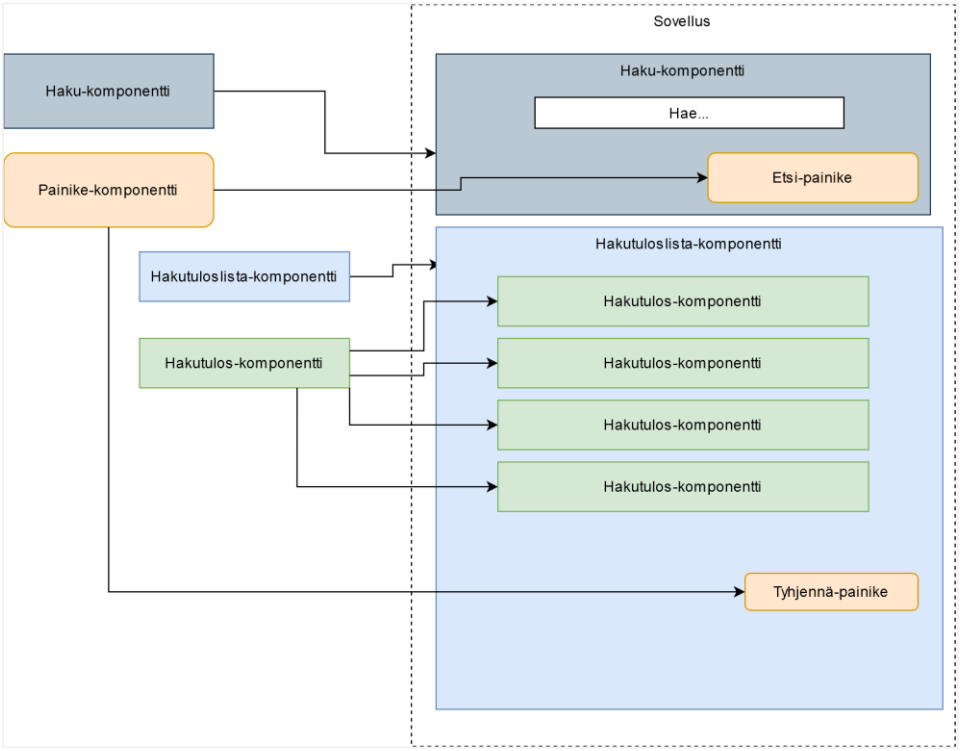
Kuvassa 6 olevassa esimerkissä on esitetty hyvin yksinkertainen React-komponentti. Return-lausekkeen sisällä oleva HTML:n näköinen syntaksi on JSX:ää. Siinä on div-elementti, jonka sisällä on h1-elementti, sekä p-elementti. P-elementin sisällä on kaarisulut, jonka sisällä on JavaScript-lauseke. Kaarisulkujen sisällä voi olla mikä tahansa JavaScript-lauseke.

```
import React from 'react';

const name = 'Jimi Koppinen';

class App extends React.Component {
  render() {
    return (
      <div>
        <h1>Tämä on JSX-syntaksiesimerkki</h1>
        <p>Hei, {name}</p>
      </div>
    );
  }
}

export default App;
```

Kuva 6. Yksinkertainen React-komponentti

4.2.2 Uudelleenkäytettävyys ja propsit

Kaaviokuvassa 7 kuvataan React-komponenttien uudelleenkäytettävyyttä korkealla tasolla. Esimerkkinä on hyvin yksinkertainen hakusovellus. Painikekomponenttia käytetään esimerkissä kaksi kertaa. Haku-komponentin sisällä sitä käytetään etsi-toiminnon aloittamiseen ja Hakutuloslista-komponentin sisällä sitä käytetään tyhjentämään hakutulokset. Hakutuloslista-komponentin sisällä on lukuisia Hakutulos-komponentteja. Hakutulos-komponentti on tyypillinen esimerkki React-komponenttien uudelleenkäyttämisestä. Hakutulos-komponentti näyttää käyttöliittymässä uniikkia dataa, vaikka pohjana on yhteinen hakutulos-komponentti. Tämä uniikin datan näyttäminen tapahtuu properties/propseja hyödyntämällä.



Kuva 7. Esimerkki React-komponenttien uudelleenkäytettävyydestä

Jokaisella React-komponentilla on props-ominaisuus. Tämän ominaisuuden tarkoituksena on kerätä syötettyä dataa komponentin käyttöä varten. Aina, kun JSX-ominaisuus kiinnitetään React-elementtiin, niin saman niminen ominaisuus kiinnitetään myös props-ominaisuuteen. (Chiarelli, 2018, s. 44). Props eroaa komponentin sisäisestä tilasta siten, että propsit ovat vain-luku muodossa ja niitä ei voi muuttaa.

Props-ominaisuutta hyödynnetään kuvan 7 tyylisissä hakutulos-komponentin uudelleenkäytöissä, kun dataa halutaan siirtää emokomponentilta lapsikomponentille. Hakutuloslista-komponenttiin on liitetty hakutulos-komponentteja. Näille hakutulos-komponenteille voidaan syöttää dataa propseina. Esimerkiksi JSX-lauseke {this.props.hakutulos.itemName} viittaisi hakutulos-komponentin nimeen, joka tulostetaan käyttöliittymälle. Esimerkiksi haettaessa sanalla ”kirja” ensimmäisen hakutuloksen nimi voisi olla ”Harry Potter” ja toisen hakutuloksen nimi taas ”Taru sormusten herrasta”, vaikka kyseessä on sama pohjakomponentti.

4.2.3 Elinkaarimetodit ja luokkapohjaiset komponentit

React-komponentteja voidaan kirjoittaa funktionaalisiksi komponenteiksi tai luokkapohjaisiksi komponenteiksi. (reactjs.org, n.d.) YLVAssa käytetään pääosin luokkapohjaisia komponentteja. Samoin myös Häiriintyvät kohteet -moduulin osalta. Funktionaaliset komponentit voivat olla syntaksiltaan hieman yksinkertaisempia, mutta ennen React 16.8. versiota

funktionaalisissa komponenteissa ei ole ollut mahdollista käyttää muun muassa tilanhallintaan tarkoitettuja hooks-funktioita.

YLVAssa on käytössä React-versio 15.6.2. joten siinä käytetään luokkapohjaisia komponentteja. Luokkapohjaisilla komponenteilla on käytössään elinkaarimetodeja komponentin eri elinkaarien vaiheisiin. Esimerkiksi ComponentDidMount()-metodia kutsutaan silloin, kun komponentti liitetään DOM-malliin. (reactjs.org, n.d.) ComponentDidMount-metodissa yleensä tehdään komponenttiin liittyviä datahakuja. Elinkaarimetodeja on muun muassa ComponentDidUpdate, ComponentDidUnmount ja ComponentWillReceiveProps. Tässä opinnäytetyössä käytetään ainoastaan ComponentDidMount-metodia.

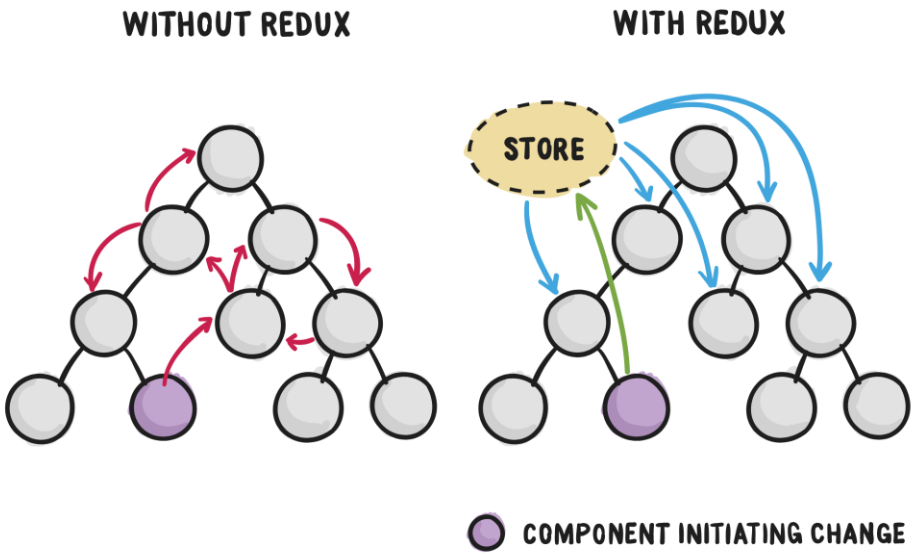
4.3 **Redux**

Redux on JavaScript -kirjasto, jota käytetään sovelluksen globaalin tilan (state) hallintaan käyttölliittymien ohjelmoinnissa. Reduxia käytetään usein yhdessä muiden kirjastojen, kuten Reactin kanssa. Vaikkakin Reduxia käytetään yhdessä Reactin kanssa, niin ne eivät ole toisiinsa sidottuja kirjastoja, vaan tarvitsevat usein väliohjelmiston (middleware) toimiakseen keskenään.

React-sovelluksen ohjelmointi onnistuu ilman Reduxia, ja tällöin Reactkomponenttien omia tiloja voidaan hyödyntää ohjelmoinnissa ja datan tallettamisessa komponenttiin. Redux otetaan usein käyttöön silloin, kun jokin komponentti tarvitsee jakaa oman tilansa toisen komponentin kanssa. Reduxissa puhutaan globaalista tilasta ja storesta, jota kaikki Reduxia käyttävät komponentit voivat hyödyntää. (Knüssel, 2017)

Kun React-sovellus kasvaa isoksi ja monimutkaiseksi niin datan liikuttaminen komponentilta toiselle voi olla haastavaa, mikäli ei käytetä globaalia tilaa tai Reduxia tilanhallintaan. Tällöin dataa täytyy ”vyöryttää” komponentilta toiselle ja vyöryttäminen komponentilta toiselle on erittäin haasteellista, jos kommunikoivien komponenttien välissä on lukuisia muita komponentteja, jotka eivät tarvitse vyörytettävää dataa. Tällaisissa tapauksissa Reduxin käyttö voi olla kannattavaa (Hoque, 2018).

Kuvassa 8 on kuvattu datan jakamista ilman Reduxia ja Reduxin kanssa. Purppuralla kuvattu pallo kuvastaa komponenttia, joka kutsuu tilaa sisältävää komponenttia. Mikäli käytössä ei ole Reduxia tai globaalia tilanhallintaa, niin dataa joudutaan kuljettamaan usean välikäden kautta. Kun taas Redux on käytössä, niin tällöin kutsutaan globaalia tilaa, jota myös säilöksi kutsutaan. Tällöin kutsuva komponentti saa haluamansa tilan suoraan säilöstä ilman, että sitä tarvitsee kuljettaa välikäsien kautta. Mikäli data muuttuu, niin samalla globaali tila päivittyy kaikille sitä tarvitsevien komponenttien ulottuville.



Kuva 8. Komponenttipuu Reduxin kanssa ja ilman Reduxia (Medium.com, 2017)

YLVAn tapauksessa puhutaan hyvin laajasta ja monimutkaisesta kokonaisuudesta, jolloin Reduxia hyödynnetään miltei jokaisessa toteutettavassa moduulissa.

4.4 Muut JavaScript-apukirjastot

Projektissa käytetään lukuisia apukirjastoja. Kaikkia kirjastoja ei luetella, mutta moduulin toteutuksen kannalta tärkeimmät avataan tässä kappaleessa. React-Intl on JavaScript-kirjasto, jota käytetään Reactin yhteydessä käännöksiä varten. React-Intliä käyttää käännöstekstilähteinään JSON-muotoisia käännöstiedostoja. Jokaisessa YLVA-moduulissa on oma käännöstiedostonsa sekä suomeksi että ruotsiksi. React-Intliin viitataan käytännön toteutusosassa.

Yksi käytettävistä JavaScript-lisäkirjastoista on Ramda. Ramdaa käytetään tässä komponentissa helpottamaan JavaScript-olioiden ja muiden tietomallien käsittelyä funktionaalisen paradigman avulla. (ramdaJS.com, n.d.). Olioiden käsittely ilman JavaScript-lisäkirjastoa voi olla haastavaa ja koodin lukeminen haastavaa muille kehittäjille.

Reactin ja Reduxin yhteenliittämisessä tarvitaan React-Redux-apukirjastoa ja sieltä etenkin connect-funktiota, jonka avulla Reduxin globaali tila saatetaan React-komponenttien käytettäväksi. React-Redux kirjaston avulla globaalia tilaa voidaan myös päivittää. (react-redux.js.org, n.d.). React-Reduxista puhuttaessa viitataan usein middleware-ohjelmistoksi.

Näiden lisäksi käytetään Redux Form -apukirjastoa. Redux Formia käytetään lomakedatan tallentamiseen ja hallintaan Reduxin globaaliin tilaan. (redux-form.com, n.d.). Redux Formin käyttöön viitataan käytännön toteutuksessa.

4.5 .NET / C#

C# on Microsoftin kehittämä olio-ohjelmointikieli. C#-kieltä ajetaan .NET-viitekehyksessä. C#-kieltä käytetään web-sovellusten, työpöytäsovellusten, mobiilisovellusten, pelien ja muiden sovellusten kirjoittamiseen. (W3Schools.com, 2020)

Kuten kuvan 5 yhteydessä huomataan, C#-kieltä käytetään YLVAssa pääosin back-end-kehittämiseen. Suurin osa liiketoimintalogiikasta on kirjoitettu C#-kieltä käyttäen. Taustajärjestelmäprojekteissa käytetään .NET-viitekehystä ja ohjelmointirajapintaprojektissa käytetään .NET-core-viitekehystä. Viitekehysten erotus johtuu pääosin siitä, että sekä ohjelmointirajapintaprojekti että .NET-core ovat uudempia kuin .NET-viitekehys ja taustajärjestelmäprojektit. Tämä työ keskittyy pääosin Front-end -kehittämiseen, jonka vuoksi .NET viitekehystä ja C#-kieltä ei ole tässä osiossa kuvattu laajemmin.

4.6 RabbitMQ

RabbitMQ on Erlang-ohjelmointikieltä käyttävä avoimen lähdekoodin viestitysjärjestelmä. (RabbitMQ, n.d.) RabbitMQ välittää viestejä eri järjestelmistä toisille. Esimerkiksi aluehallinnon asiointipalvelusta välitetään viesti YLVA-käsittelyjärjestelmälle, kun asiakas lähettää lomakkeen asiointipalvelun kautta.

4.7 Aspose.Words for .Net

Aspose.Words on edistynyt ohjelmointirajapinta, jota käytetään Word-tiedostojen luomiseen, muokkaamiseen, tulostamiseen ja konvertoimiseen uuteen muotoon. (Aspose, n.d.)

Asposea käytetään pääosin asiakirjadokumenttien luontiin, kun asiointilomake on täytetty. Luotu dokumentti lähetetään asianhallintajärjestelmään, jossa sitä jatkokäsitellään. Asiakas saa myös ladattua dokumentin itselleen PDF-muodossa, kun se on luotu Asposen rajapinnan avulla.

4.8 JSON Schema

JSON Schema on tehokas työkalu JSON-datan rakenteen validointiin. (JSON Schema, n.d.) YLVAssa JSON Scheman validointia käytetään, kun lomakedataa lähetetään eteenpäin taustaprosessien käsiteltäväksi.

Taustaprosessit tarkistavat JSON Shceman avulla, että lähetettävä JSON-data on rakenteellisesti oikeassa muodossa. Toteutusvaiheessa tästä saadaan käytännön esimerkki.

4.9 **Tietokantayhteydet ja Entity Framework**

Tietokantana projektissa käytetään Microsoftin SQL Serveriä ja tietokannan kanssa kommunikointiin käytetään Microsoftin Entity Frameworkia. Microsoft SQL Server on järjestelmä relaatiotietokantojen hallintaan. SQL Server perustuu SQL-tietokantakyselykieleen. (sqlservertutorial.net, n.d.) Entity Framework taas on ORM-viitekehys .NET-ohjelmien käyttöön. Entity Frameworkin avulla voidaan käsitellä tietokannan sisältämää dataa luokkapohjaisesti ilman, että tarvitsee tietää missä sarakkeessa mikäkin tieto sijaitsee. Entity Frameworkia voidaan käyttää Database first -mallina, jolloin tietokanta luodaan ensiksi ja luokat vasta tämän jälkeen. Toinen vaihtoehto on käyttää ns. Code first -mallia, jolloin dataluokat kirjoitetaan C#-kielellä, jonka perusteella tietokanta luodaan. YLVAssa käytetään Database first -mallia. (entityframeworktutorial.net, n.d.)

5 MODUULIN SUUNNITTELU

Toimeksiannon toteuttajan, Alfame Systemsin ja asiakkaan, KEHA-keskuksen kesken on järjestetty suunnittelupalaveri 1.5.2020 Ympäristönsuojelulain mukaisen rekisteröinnin eli REKI-asioinnin suunnittelusta. Palaverissa käytiin läpi asiakkaan kanssa asioinnin sähköistämistä ja lomakkeiden digitalisoimista ja siihen liittyviä toimenpiteitä. Asiakkaan edustajat olivat jo sisäisessä palaverissaan tehneet sisäistä määrittelytyötä ja miettineet valmiiksi millaisia vaatimuksia heillä on moduulin ja asioinnin toteutukselle.

5.1 Vaatimusmäärittely asiakkaan kanssa

Koska projektissa käytetään ketteriä menetelmiä sovelluksen suunnittelussa ja toteutuksessa, niin suunnitteluun käytettävä aika pidettiin suhteellisen lyhyenä itse määrittelypalaverissa. Tarkoituksena on käydä asiakkaalle läpi moduulin toimintaa Scrum-viitekehyksen mukaisesti pyrähdyksen katselmointipalaverissa (sprint review), jotta saadaan mahdollisesti asiakkaalta palautetta moduulin toiminnallisuudesta. Katselmointipalaverissa saadun palautteen jälkeen tarkistetaan kehitystyön tilanne ja tarvittaessa muutetaan toiminnallisuutta asiakkaan palautteen mukaisesti.

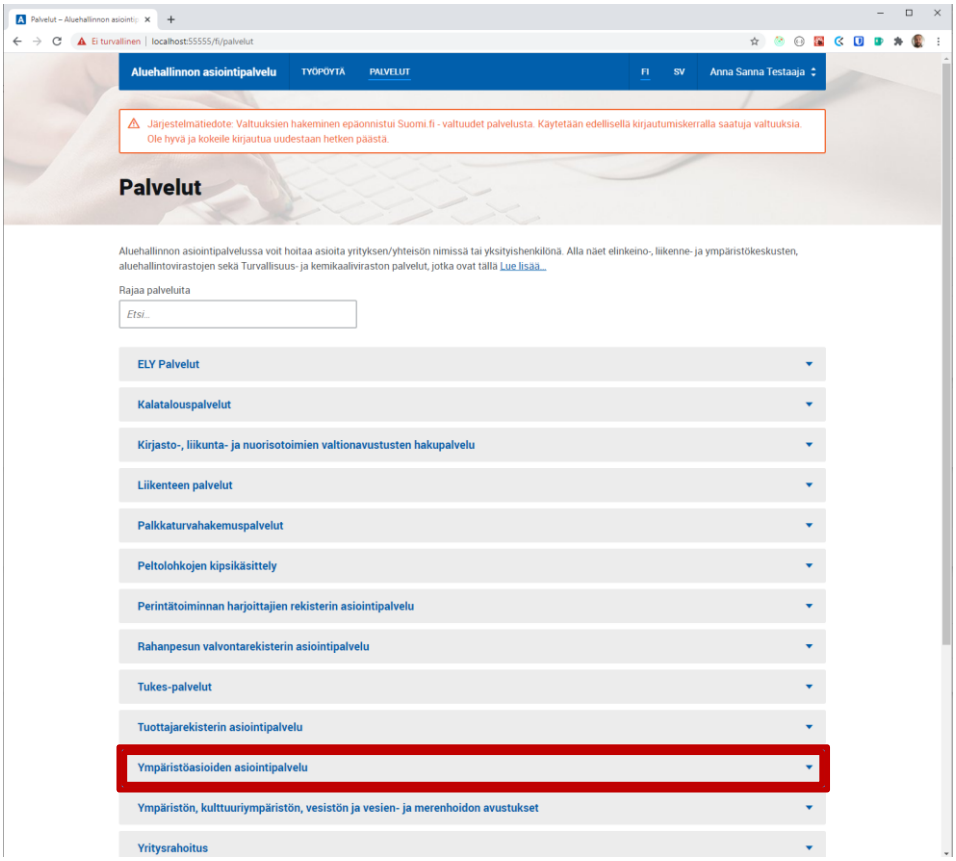
Asiakkaan yhteyshenkilöihin on helppo ja nopea ottaa yhteyttä esimerkiksi Teamsilla tai sähköpostilla, jos tarvitaan tarkempaa määrittelyä kehitystyön tueksi. Tämä on yksi osa ketterää kehitystä Alfame Systemsillä.

Kehitystiimissä Alfame Systemsillä on opinnäytetyön tekemisen hetkellä neljä henkilöä, joilla on usean vuoden kokemus YLVAn ja SPAv2-alustan kehittämisessä. Tämän vuoksi asiakas ei anna kovinkaan tarkkoja raameja kehitystyölle, vaan luottaa kehitystiimin asiantuntemukseen. Kehitystiimi arvioi sprint planningissa REKI-asiointiin liittyvien moduulien työmäärät ja toimittaa tiedon työmääristä asiakkaalle. Kun toimeksiantosopimus on sovittu asiakkaan ja toimeksiannon toteuttajan välille, kehitystyö aloitetaan.

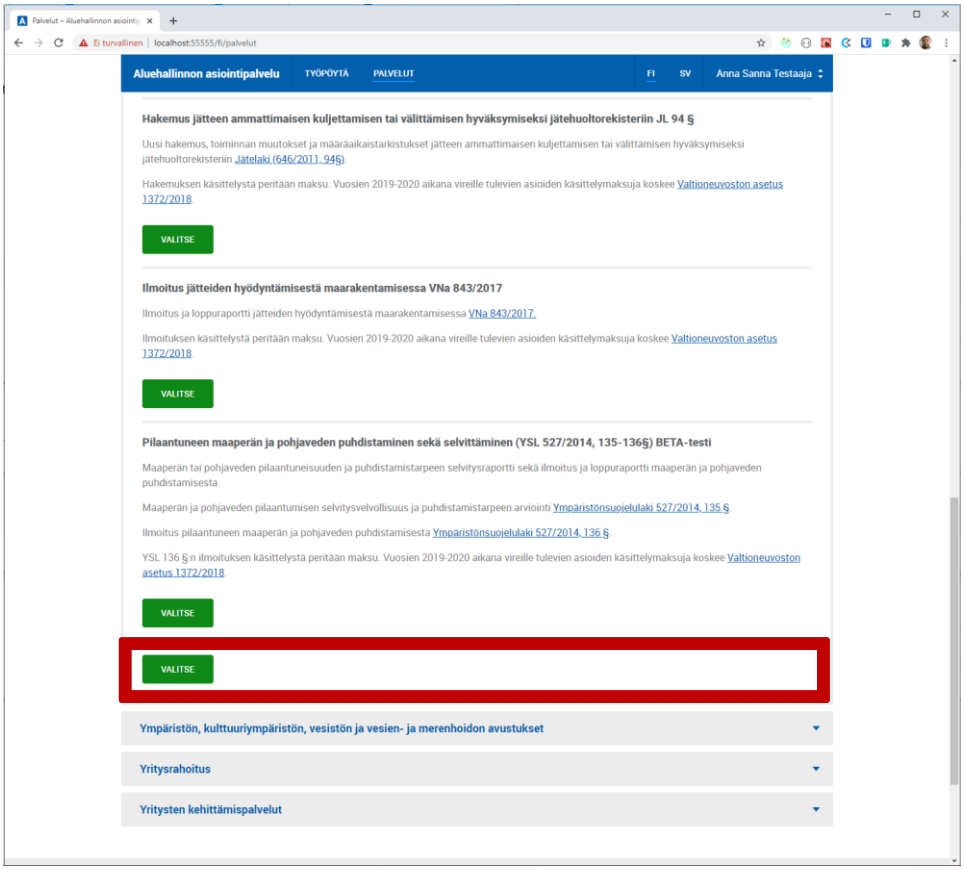
5.2 Suunniteltava moduuli osana asiointiprosessia

Suunniteltava Häiriintyvät kohteet -moduuli on osa REKI-asiointia. Ennen lomakkeen digitalisointia lomake on löydettävissä ymparisto.fi -palvelusta. Toiminnanharjoittaja täyttää lomakkeen manuaalisesti ja lähettää lomakkeen joko postitse tai sähköpostin liitteenä kunnan ympäristönsuojeluviranomaiselle. Tarkoituksena on sähköistää rekisteröinti-ilmoituksen tekeminen niin, että lomake löytyy aluehallinnon asiointipalvelusta ympäristöasioiden palveluiden alta.

Kuvissa 9 ja 10 nähdään ympäristöasioiden asiointipalvelun sijainti Palvelut-sivulla. Kuvakaappaukset ovat kehitysympäristöstä Chrome-selaimella. Kuvaan on merkattu punaisella tulevan REKI-asioinnin sijainti. Asiakkaan toiveiden mukaisesti sijainti on lisätty asiointityypeistä alimmaiseksi.



Kuva 9. Aluehallinnon asiointipalvelun palvelut-sivu ja REKI-asioinnin sijainti



Kuva 10. REKI-asioinnin sijainti ympäristöasioiden asiointipalvelussa

Asiakkaan toiveena oli myös, että REKI-asioinnin käyttöliittymän käytettävyys on hyvällä tasolla. Tämän vuoksi lomakkeen sähköiseen optimointiin käytettiin huomattavasti resursseja. Tarkoituksena oli tehdä asioinnista intuitiivista ja helppoa. Aikaisemmassa lomakkeessa näitä asioita ei juurikaan otettu huomioon.

Asiakas oli valmiiksi määritellyt REKI-asioinnin profiloinnin ja asettanut raamiehdot sen toteuttamiseen. Uuden määrittelyn jälkeen REKI-lomakkeen täyttämisprosessi ja järjestys hieman muuttuu. Aikaisemmin jokaiselle toimialalle eli asfalttiasemille, betoniasemille ja betonituotetehtaille, keskisuurille energiatuotantolaitoksille, polttonesteiden jakeluasemille ja orgaanisia liuottimia käyttävälle toiminnalle on ollut oma lomakkeensa. Nyt täytettävä lomake tullaan toteuttamaan niin, että asioija/asiakas valitsee toimialan, jolla toimii ja hänelle näytetään toimialaspesifiset lomakemoduulit sekä yhteiset moduulit, jotka ovat useissa asioinneissa käytössä.

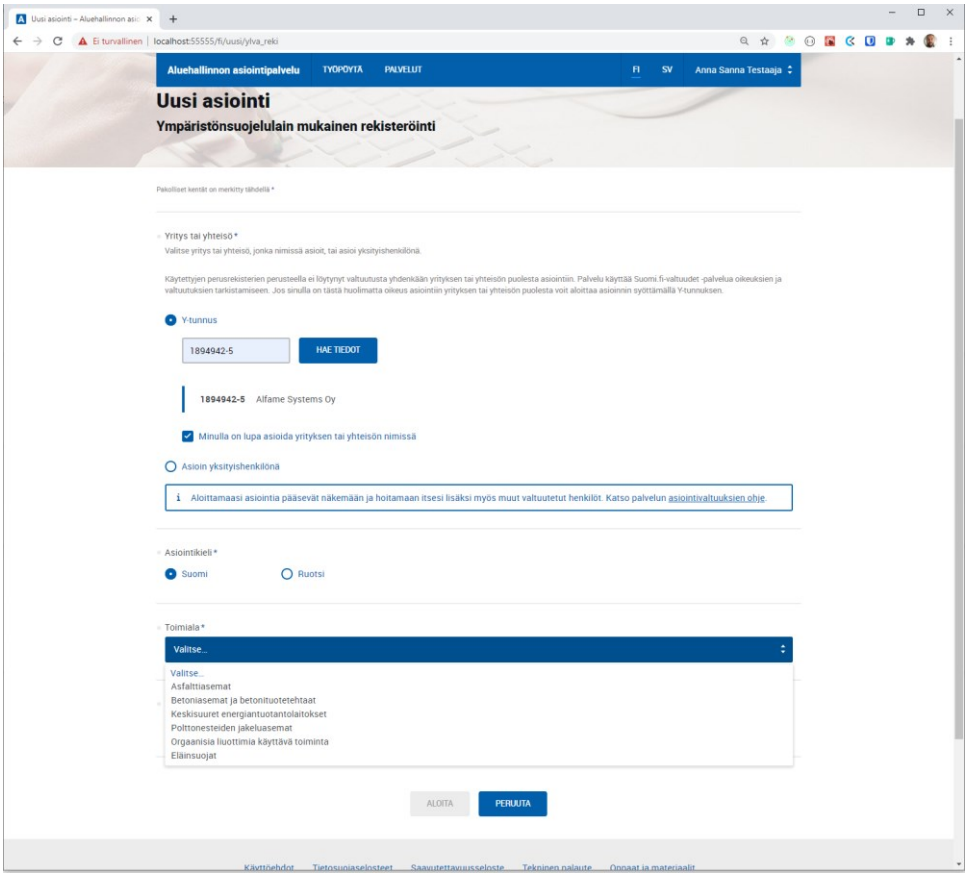
Asiakas on määritellyt REKI-asioinnin profiloinnin seuraavasti. Asiointeja on kahta tyyppiä: suppea ja laaja. Suppeassa asioinnissa pysähdytään vaiheeseen 3 ja laajassa täytetään myös vaihe 4.

Asioinnin vaiheet:

1. Valitaan toimiala
2. Syy rekisteröinti-ilmoitukselle
3. Kysytään toiminnanharjoittajan yhteystiedot ja kohteen perustiedot
4. Täytetään toimialakohtainen ilmoitus

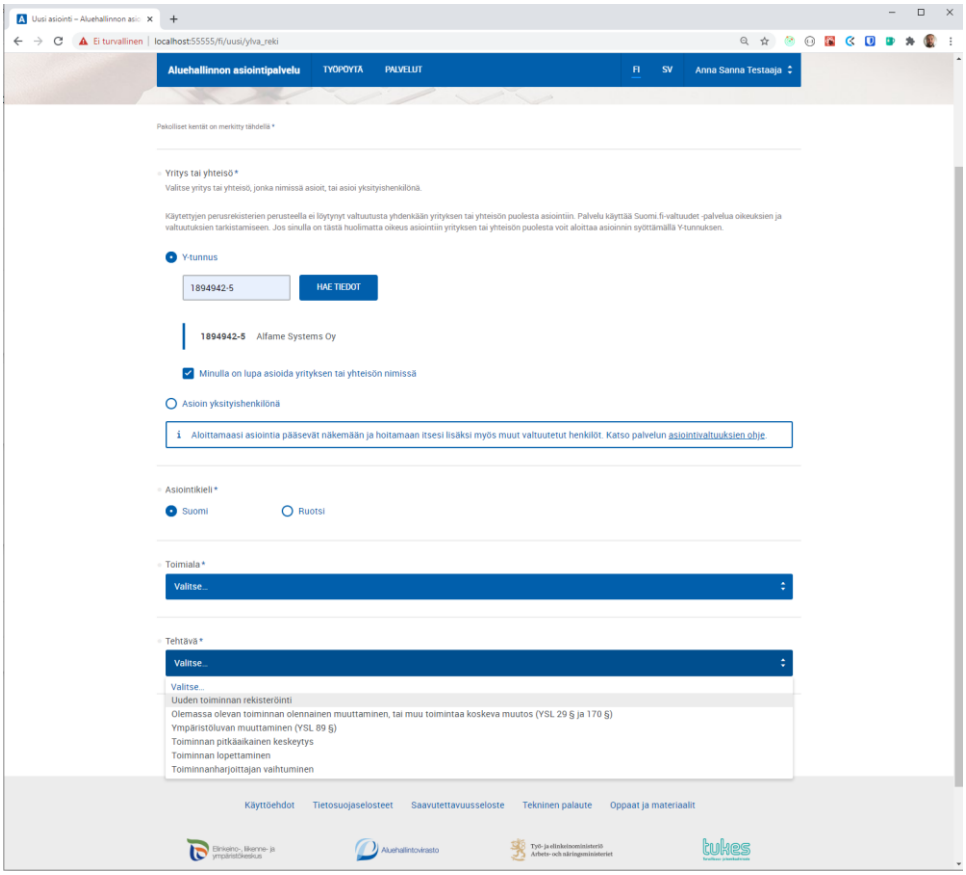
Vaiheessa 1 valitaan yksi seuraavista toimialoista: Asfalttiasemat, Betoniasemat ja betonituotetehtaat, Keskisuuret energiantuotantolaitokset, Polttonesteiden jakeluasemat, Orgaanisia liuottimia käyttävä toiminta, Eläinsuojat.

Vaiheessa 2 käyttäjältä kysytään, miksi ilmoitusta ollaan tekemässä. Valinta ohjaa lomaketta joko laajaan tai suppeaan asiointiin. Vaihtoehtoja on kuusi kappaletta. Laajaan asiointiin johtavat nämä valinnat: uuden toiminnan rekisteröinti, olemassa olevan toiminnan olennainen muuttaminen, tai muu toimintaa koskeva muutos (YSL 29 § ja 170 §) tai ympäristöluvan muuttaminen. Suppeaan asiointiin johtavat seuraavat valinnat: toiminnan pitkäaikainen keskeytys, toiminnan lopettaminen ja toiminnanharjoittajan vaihtuminen. Suppeassa asioinnissa lomake täytetään 3. vaiheeseen asti.



Kuva 11. Kuvakaappaus REKI-asioinnin profilointinäkymästä

Kuvassa 11 nähdään REKI-asioinnin profilointinäkymä, joka on Häiriintyvät kohteet -moduulin toteutusvaiheessa valmis. Pudotusvalikossa näkyvät valittavana olevat toimialat, jonka mukaisesti toimialakohtaiset moduulit tulevat näkyviin asioinnin aloittamisen jälkeen.



Kuva 12. Kuvakaappaus tehtävän valinnasta

Kuvassa 12 nähdään, että käyttöliittymässä on valittavissa syy, miksi ilmoitus tehdään. Tämän valinnan myötä aloitetaan joko suppea tai laaja asiointi. Häiriintyvät kohteet -moduulia voidaan hyödyntää kaikkien toimialojen asioinnissa ja nimenomaan laajassa asioinnissa.

Asiakas on toimittanut kuvankaappauksen siitä miltä moduulin tulisi pääpiirteittäin näyttää osana asiointiprosessia. Ohjelmistokehittäjille on kuitenkin annettu osittainen suunnitteluvapaus toteutuksen osalta. Alla olevassa kuvassa nähdään, että Häiriintyvät kohteet -moduulissa käyttäjän tulee ilmoittaa kaikki häiriölle alttiit kohteet sekä muut herkät kohteet, jotka sijaitsevat alle 500 m etäisyydellä häiriötä aiheuttavasta toiminnasta. Kuvasta huomataan, että kyseessä on staattinen täytettävä taulukko.

Kaikki häiriölle alttiit kohteet sekä muut herkät kohteet, jotka **sijaitsevat alle 500 m etäisyydellä** häiriötä aiheuttavasta toiminnasta:

Kohde*	Kohteen nimi, kiinteistötunnus tai käyntiosoite	Etäisyys asfaltti-asemasta (m)	Merkintä laitoksen sijaintikartalla (liite A)
Asuinkiinteistö			
Loma-asunto			
Koulu tai päiväkoti			
Leikkikenttä			
Sairaala			
Virkistysalue			
I tai II luokan pohjavesialue			
Pohjavedenottamo			
Natura 2000 -alue			
Muu luonnonsuojelukohde			
Muu häiriölle altis kohde			

* Kohteet on merkittävä myös sijaintikarttaan (liite A)

☐ Lähiseudulla sijaitsee muita ympäristöä kuormittavia toimintoja, mitä? Etäisyys asemasta m

Sijaitseeko samalla kiinteistöllä muita toimintoja, jotka eivät liity rekisteröitävään/ilmoitettavaan toimintaan?
☐ Ei
☐ Kyllä, mitä?

Kuva 13. Asiakkaan määrittelypalaverissa toimittama kuva Häiriintyvät kohteet -lomakkeen osiosta

Jotta moduulista saadaan mahdollisimman toimiva sähköisen asioinnin kannalta, niin staattisen taulukon sijaan tullaan hyödyntämään dynaamista taulukkoa ja FormTable -React-komponenttia, jonka avulla voidaan lisäämään kuvan 13 mukaisia kohteita ilman, että ne ovat kaikki heti nähtävillä. FormTable-komponentin käyttö selkeyttää käyttöliittymän käyttäjäkokemusta huomattavasti. FormTable-komponenttiin tulee pudotusvalikko, josta käyttäjä voi valita hänen tilanteeseensa sopivan kohteen sekä täyttää tiedot valinnan jälkeen. Näin käyttäjälle ei näytetä epärelevanttia tietoa käyttöliittymässä ja käyttökokemus säilyy selkeänä.

Moduulin alareunasta löytyvän tekstin ”Lähiseudulla sijaitsee muita ympäristöä kuormittavia toimintoja, mitä” jälkeen tullaan käyttämään ehdollista renderöintiä. Tämä tullaan toteuttamaan valintanapilla. Eli kun käyttäjä klikkaa valintanappia, niin lisätekstiä tulee näkyviin. Käyttäjälle näytetään vain hänelle relevanttia tietoa käyttöliittymässä. Mikäli lähiseudulla ei ole muita ympäristöä kuormittavia toimintoja, niin täyttökenttää ei näytetä käyttäjälle.

Varsinainen ulkoasuohjeistus on määritetty KEHA-keskuksen ja dedikoidun SPAv2-tiimin ja UX-designerien toimesta entuudestaan. Varsinaisiin CSS-tiedostoihin ei juurikaan tehdä muutoksia. React-komponenteissa voidaan käyttää inline-stylingiä, eli määritellään pienet tyylimuutokset suoraan JSX-elementtiin.

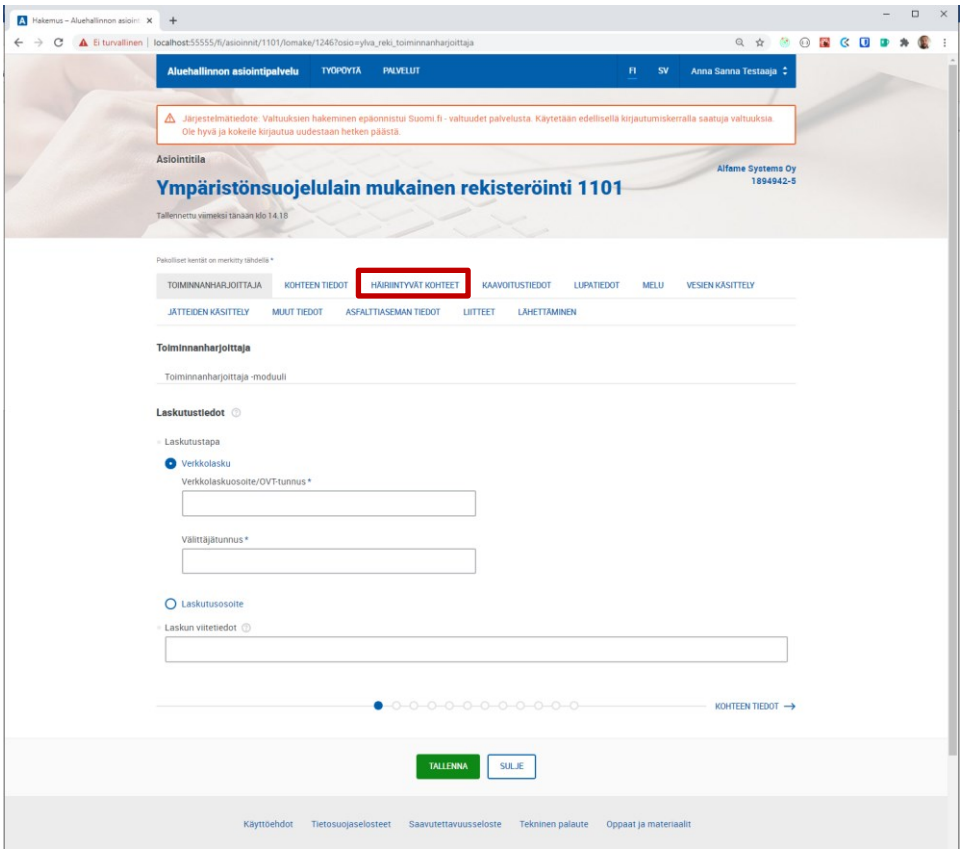
6 MODUULIN KÄYTÄNNÖN TOTEUTUS

Tässä luvussa kuvataan moduulin käytännön toteutusta. Toteutuksessa esitetään kuvakaappauksia käyttöliittymästä sekä koodiesimerkkejä. REKI-asioinnissa esimerkkitoimialana käytetään asfalttiasemaa ja asiointina uuden toiminnan rekisteröintiä. Asiointiyrityksenä käytetään Alfame Systems Oy:ta, jolla ei todellisuudessa ole toimintaa asfalttiasemana. Kaikki data on kehittäjän koneella paikallisesti, joten tietoja ei välitetä kehitysympäristöstä missään välissä palvelimelle.

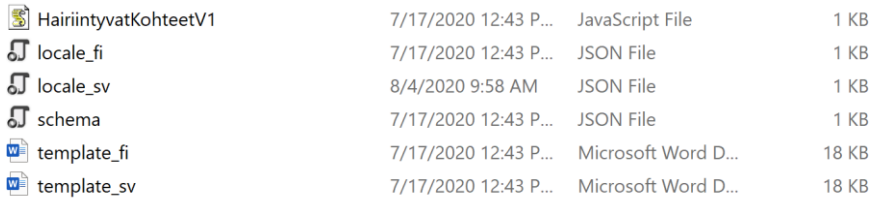
Aluksi kuvataan moduulin lähtötilannetta, kun ohjelmakoodia ei ole juurikaan kirjoitettu ja tiedostojen sisältö on suppea. Osion lopuksi käydään läpi valmiin moduulin koodia ja sen toiminnallisuuksia. Lopuksi selitetään valmiin moduulin toiminnallisuuksia käyttöliittymänäkymän avulla.

6.1 Moduulin lähtötilanteen kuvaus

Kuvassa 14 on näkyvissä uuden toiminnan rekisteröinti -asioinnin välilehdet sekä moduulit. Häiriintyvät kohteet moduuli on ympyröity punaisella laatikolla.



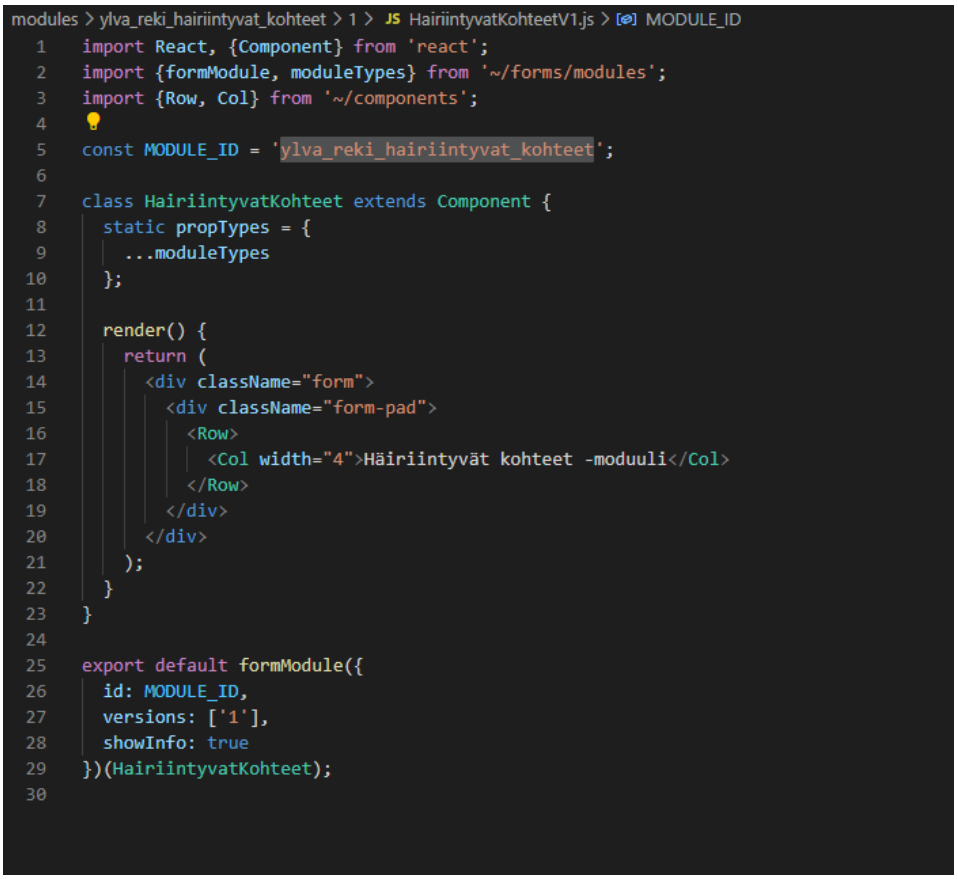
Kuva 14. Uuden toiminnan rekisteröinnin välilehdet ja moduulit.



HairiintyvatKohteetV1	7/17/2020 12:43 P...	JavaScript File	1 KB
locale_fi	7/17/2020 12:43 P...	JSON File	1 KB
locale_sv	8/4/2020 9:58 AM	JSON File	1 KB
schema	7/17/2020 12:43 P...	JSON File	1 KB
template_fi	7/17/2020 12:43 P...	Microsoft Word D...	18 KB
template_sv	7/17/2020 12:43 P...	Microsoft Word D...	18 KB

Kuva 15. Moduulin aloitusvaiheen tiedostot

Kuvassa 15 on nähtävillä kaikki sellaiset tiedostot, joita vähintään tarvitaan, että SPAv2-moduuli toimii tarkoituksen mukaisesti. Ensimmäisenä on HairiintyvatKohteetV1.js -tiedosto. Se sisältää pääkomponentin, jossa hyödynnetään tekniikkana Reactia ja jonka ympärille moduuli rakentuu. Tämän jälkeen on kolme JSON-tiedostoa. locale_fi-tiedostosta löytyy moduulin suomenkieliset tekstisisällöt ja locale_sv-tiedostosta löytyy taas ruotsinkieliset tekstisisällöt, joita React-Intl hyödyntää käyttöliittymän käännöksissä. Schema-tiedostosta löytyy JSON-Schema tiedosto, jota käytetään JSON-datan validointiin. Template_fi-tiedostosta löytyy PDF-tulostusta varten moduulin täytettyjen lomakekenttien tiedot. Aloitushetkellä suurin osa tiedostoista on tyhjiä.



modules > ylva_reki_hairiintyvat_kohteet > 1 > JS HairiintyvatKohteetV1.js > MODULE_ID

```
1  import React, {Component} from 'react';
2  import {formModule, moduleTypes} from '~/forms/modules';
3  import {Row, Col} from '~/components';
4
5  const MODULE_ID = 'ylva_reki_hairiintyvat_kohteet';
6
7  class HairiintyvatKohteet extends Component {
8    static propTypes = {
9      ...moduleTypes
10   };
11
12   render() {
13     return (
14       <div className="form">
15         <div className="form-pad">
16           <Row>
17             <Col width="4">Häiriintyvät kohteet -moduuli</Col>
18           </Row>
19         </div>
20       </div>
21     );
22   }
23 }
24
25 export default formModule({
26   id: MODULE_ID,
27   versions: ['1'],
28   showInfo: true
29 })(HairiintyvatKohteet);
30
```

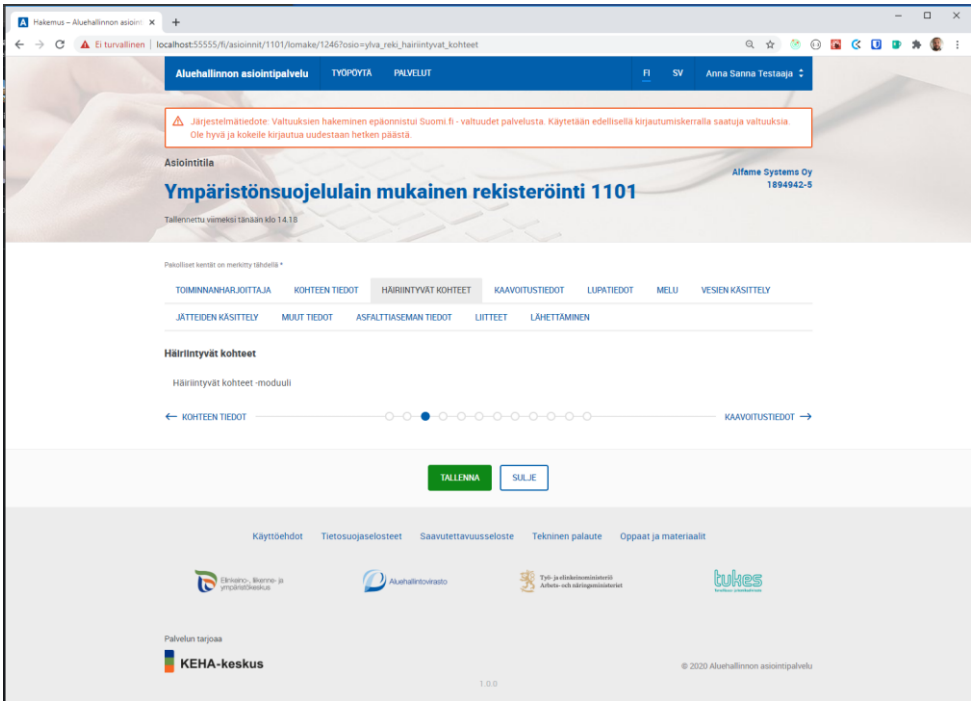
Kuva 16. Moduulin pääkomponentin JavaScript-tiedoston sisältö

Kuvassa 16 nähdään pääkomponentin lähtötilanne. Riveillä 1–3 tuodaan tarvittavat komponentit ja moduulit pääkomponentin käytettäväksi. Ensiksi tuodaan React-kirjasto ja Component-komponentti käytettäväksi,

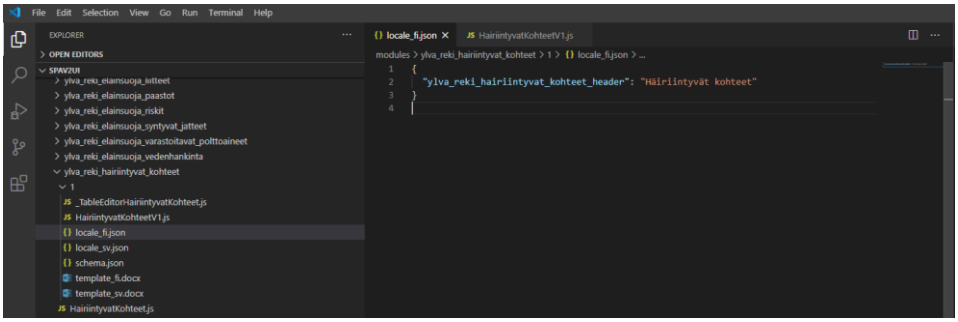
jonka lisäksi formModule ja moduleTypes -käyttöön. Tämän lisäksi komponentin lähtövaiheessa Row- ja Col -komponentteja.

Row- ja Col -komponentit hyödyntävät itsessään React-Bootstrap-kirjastoa, jonka lisäksi komponentteihin on lisätty SPAv2-spesifistä toimintalogiikkaa. Niiden tarkoitus on hyödyntää Bootsrapista tuttua grid-järjestelmää sekä käyttöliittymän optimointia eri kokoisille päätteille. W3Schoolsin verkkosivuilla kuvataan kattavasti Bootstrapin grid-järjestelmän ominaisuuksia, luokkia ja sääntöjä. (W3Schools, 2020).

Kyseessä on luokkapohjainen komponentti, jolla on tässä vaiheessa vain yksi ja ainoa React-luokan pakottama metodi, render. Render palauttaa sen mitä halutaan komponentin näyttävän käyttöliittymässä, kun se liitetään DOMiin. Kuvassa 17 nähdään, että render-metodin palauttama JSX-syntaksi näyttää käyttöliittymässä ainoastaan tekstin ”Häiriintyvät kohteet -moduuli”. Div-elementeille on asetettu CSS-määrittelyä varten className-ominaisuudet. Reactissa käytetään HTML:stä poiketen className-syntaksia class-syntaksin sijaan. CSS-luokkien käyttämisellä saadaan varmistettua Sähköisen asiointipalvelun yhtenäinen UX-ilme.

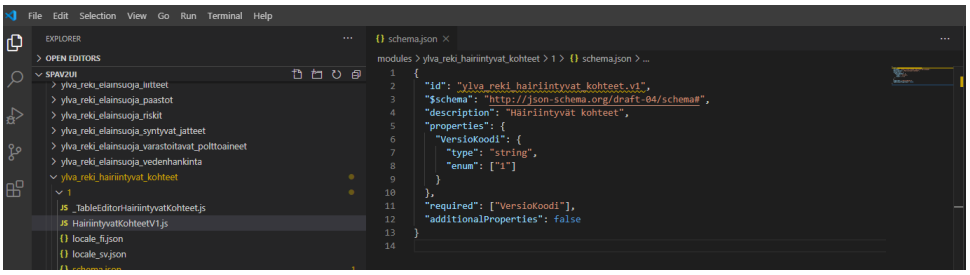


Kuva 17. Aloitusvaiheen käyttöliittymän näkymä



Kuva 18. locale_fi.json -tiedoston sisältö aloitusvaiheessa

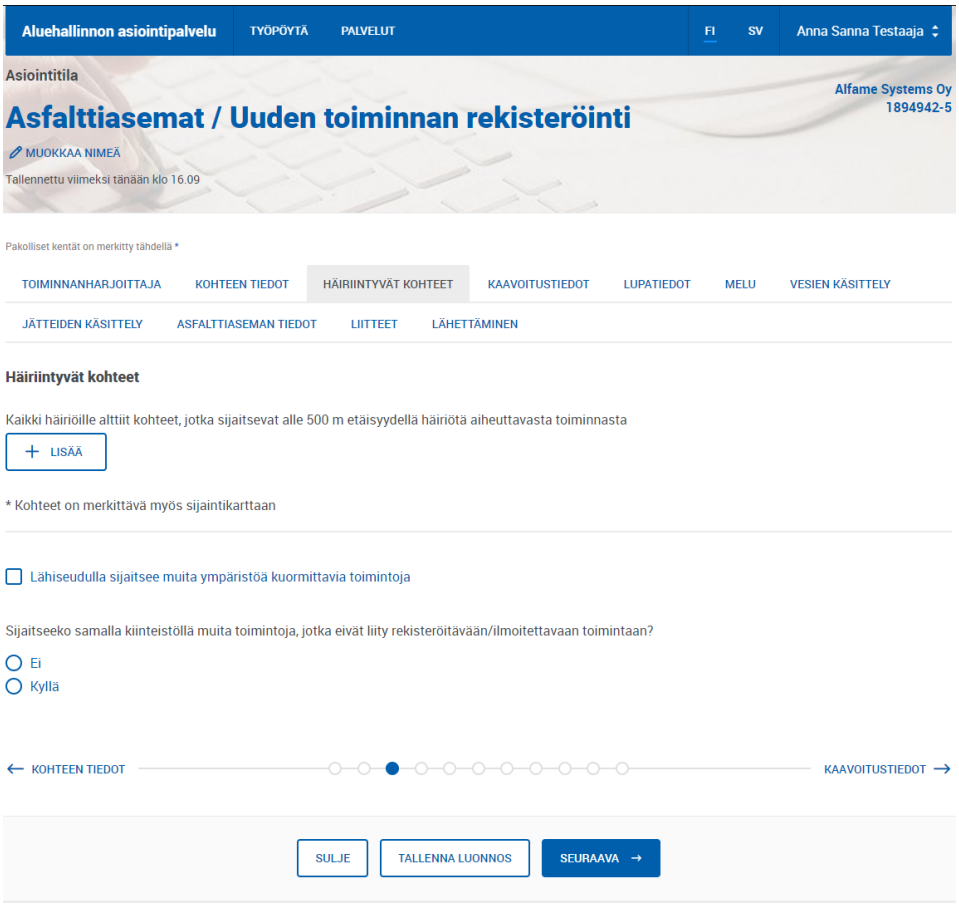
Kuvassa 18 on nähtävillä, että käännöstiedostossa on ainoastaan yksi käännös. Kyseessä on moduulin otsikko, joka näkyy kuvassa 14. Tiedostossa olevan JSON-olion avain on ylva_reki_hariintyvat_kohteet_header ja käännöstekstinä on ”Häiriintyvät kohteet”. Avain toimii yksilöivänä tekijänä komponentissa, jonka avulla React-intl-apukirjasto osaa tunnistaa oikean käännöstekstin.



Kuva 19. Schema.json -tiedoston sisältö aloitusvaiheessa

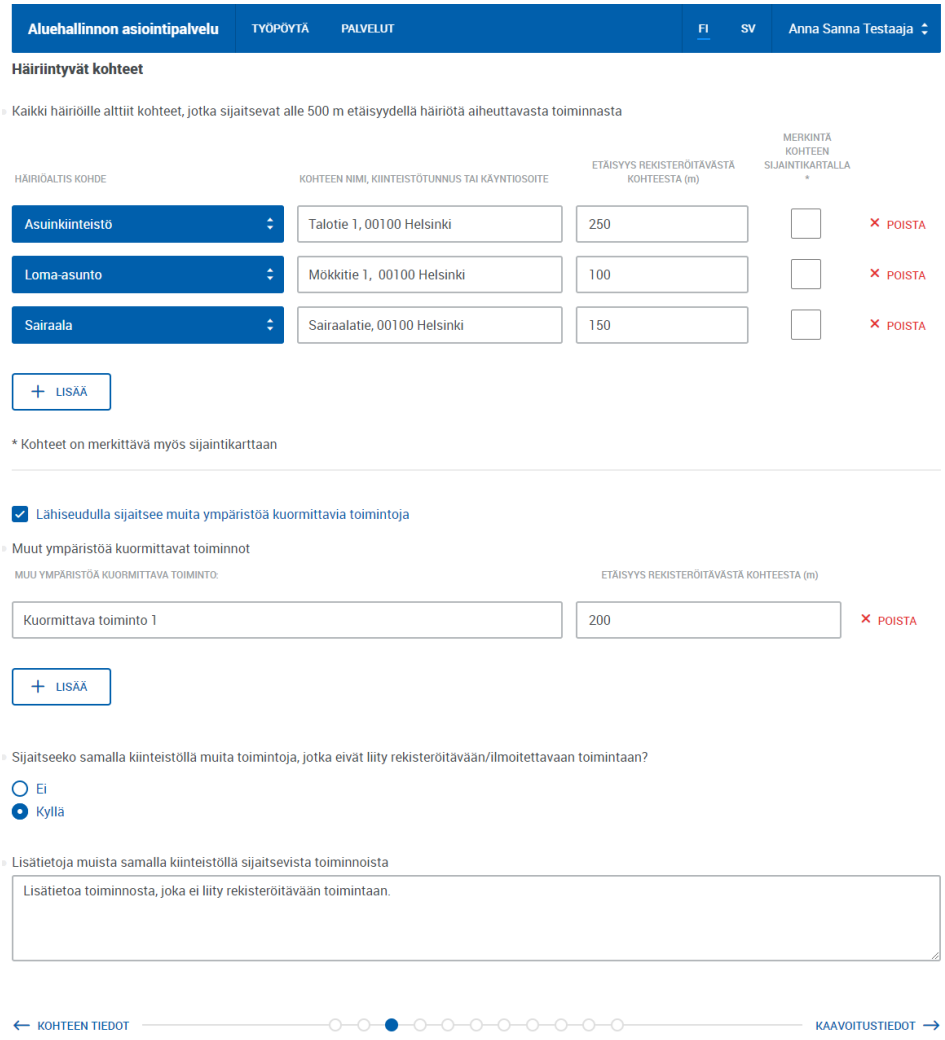
Kuvassa 19 on kuvattu Schema.json -tiedoston sisältöä moduulin aloitusvaiheessa. Schemalla on oma tunnisteensa ”id”, jonka perusteella Schema yhdistetään oikeaan moduuliin.

6.2 Valmiin moduulin kuvaus



Kuva 20. Selaimen näkymä, moduulin tiedot täyttämättä

Kuvassa 20 on näkyvissä Häiriintyvät kohteet -moduulin käyttöliitty-mänäkymä Chrome-selaimella. Tässä vaiheessa lomakkeelle ei ole vielä täytetty mitään tietoja.



Kuva 21. Selaimen näkymä, moduulin tiedot täytettynä

Kuvassa 21 on nähtävillä valmis lomakemoduuli, johon on täytetty tietoja valmiiksi. Ylin taulukkokomponentti on toteutettu dynaamisesti ja kuvassa lomakkeelle on syötetty kolme häiriöaltista kohdetta, joten vain kolme riviä taulukosta on nähtävillä. Alkuperäisessä staattisessa taulukossa kaikki häiriintyvät kohteet olivat nähtävillä. Tarvittaessa käyttäjän on mahdollista lisätä rivejä taulukkoon ilman rajoituksia. Mikäli esimerkiksi rekisteröitävän kohteen läheisyydessä on kaksi asuinkiinteistöä, niin käyttäjä voi lisätä niitä useamman. Alkuperäisellä lomakkeella ei ollut tätä mahdollisuutta. Näin ollen käyttäjäkokemuksen voidaan nähdä parantuneen vanhasta lomakkeesta.

Seuraava täytettävä taulukko tulee näkyville käyttöliittymään ainoastaan, jos ”Lähiseudulla sijaitsee muita ympäristöä kuormittavia toimintoja” -valintaruutu on valittuna. Tämän jälkeen käyttäjän on mahdollista täyttää tietoja mahdollisista muista ympäristöä kuormittavista toiminnoista. Tämän jälkeen käyttäjän on mahdollista valita valintaruutu, joka määrittää näytetäänkö seuraavaa täytettävää tekstikenttää ja sen otsikkoa lainkaan. Näin näytetään käyttäjälle vain relevanttia sisältöä käyttöliittymässä.

```
import React, {Component} from 'react';
import {formModule, moduleTypes} from '~/forms/modules';
import R from 'ramda';
import {
  FormInput,
  LabeledCheckbox,
  LabeledTextarea,
  PureLabel,
  RadioGroup,
  LabeledSelect,
  createSelectOptions,
  FormTable,
  NumberPicker,
  cellRenderer
} from '~/forms/inputs';
import PropTypes from 'prop-types';
import {getFormLanguage} from '~/selectors/form';
import {isCodeDataLoading, getCodeData} from '~/selectors/ylva';
import {fetchCodes} from '~/actions/ylva';
import {bindActionCreators} from 'redux';
import {Row, Col} from '~/components/';
import ConditionalRender from '~/components/ConditionalRender';
import {FormattedMessage} from 'react-intl';
import FormTableCheckbox from '../../ylva_yleiset/_FormTableCheckbox';
import {FieldArray} from 'redux-form';
```

Kuva 22. Häiriintyvät kohteet -pääkomponentti 1/6

Kuvassa 22 nähdään varsinaista JavaScript-koodia, jota on käytetty Häiriintyvät kohteet -komponentin rakentamiseen. Kyseinen koodi on JavaScriptin ja Reactin oman JSX-syntaksin sekoitusta. Pääkomponentin koodin rivien määrä on 225, jonka vuoksi kuvakaappauksia on useampia. Kuvassa 22 näytetään kootusti muut JavaScript-komponentit ja moduulit, jotka on tuotu pääkomponentin käyttöön. Ilman näitä tuonteja komponentti ei toimisi halutulla tavalla. Tuotua koodia avataan myöhemmin lisää.

```
const MODULE_ID = 'ylva_reki_hairiintyvat_kohteet';
const TABLE_ID1 = 'HairiintyvatKohteetTaulukko';
const TABLE_ID2 = 'MuutKuormittavatKohteetTaulukko';
const TABLE_NAME1 = `${MODULE_ID}${TABLE_ID1}`;
const TABLE_NAME2 = `${MODULE_ID}${TABLE_ID2}`;

const muutLiittymattomatToiminnotOptions = createSelectOptions(
  ['ei', 'kylla'],
  MODULE_ID,
  'MuutLiittymattomatToiminnotKoodi',
  false
);
const defaultOption = [{value: '', label: 'general_select_ellipsis'}];

const affectedTargetsTableProps = (moduleId, targetTypeOptions) => ({
  headerData: [
    {text: `${MODULE_ID}KohteenTyyppiTunnus`, width: '30%'},
    {text: `${MODULE_ID}HairiintyvanKohteenNimi`, width: '30%'},
    {
      text: (
        <div>
          <span>
            <FormattedMessage id={`${moduleId}EtaisyysKohteestaNumero`} />
          </span>
          <span className="ylva-table-header-units">
            <FormattedMessage id={`${moduleId}MetriYksikko`} />
          </span>
        </div>
      ),
      width: '20%',
      textAlign: 'center',
      raw: true
    },
    {text: `${MODULE_ID}SijaintiKartallaKytkin`, textAlign: 'center', width: '10%'}
  ],
  cellRenderers: [
    cellRenderer('KohteenTyyppiTunnus', LabeledSelect, {options: targetTypeOptions, raw: true}),
    cellRenderer('HairiintyvanKohteenNimi'),
    cellRenderer('EtaisyysKohteestaNumero', 'number', NumberPicker),
    cellRenderer('SijaintiKartallaKytkin', FormTableCheckbox)
  ],
  title: TABLE_NAME1,
  moduleId
});
```

Kuva 23. Häiriintyvät kohteet -pääkomponentti kuva 2/6

Kuvassa 23 alustetaan lomakemoduulissa käytettävien taulukkojen tietoja. Ensiksi alustetaan muuttujat TABLE_ID1, TABLE_ID2, TABLE_NAME1 ja TABLE_NAME2. Kahteen viimeksi mainittuun muuttujaan tullaan viittaamaan jatkossa useassa kohdissa. Nämä muuttujat yksilöivät sen, mistä taulukosta on kyse, kun taulukkoihin viitataan muualta ohjelmasta ja komponentin sisältä.

Tämän jälkeen alustetaan muuttuja muutliittymattomatToiminnotOptions, joka pitää sisällään kutsun createSelectOptions-funktiolle. createSelectOptions on apufunktio, joka on tuotu pääkomponentille forms/inputs-kansiosta. Sen tarkoituksena on luoda valintavaihtoehdot sellaiseen muotoon, että mm. LabeledSelect ja muut valintavaihtoehtoja sisältävät komponentit toimivat sen tuottamalla datalla. Se ottaa sisäänsä taulukon vastausvaihtoehtoja, moduulin id:n sekä sen lomakekentän nimen, jossa sitä käytetään. Tämän jälkeen alustetaan defaultOption, jota käytetään myöhemmässä vaiheessa pudotusvalikon oletusvaihtoehtona.

Tämän jälkeen alustetaan affectedTargetsTableProps funktio. Tässä funktiossa luodaan Häiriintyvät kohteet -taulukon propsit eli ominaisuudet, jotka välitetään propsina alaspäin niitä käyttäville komponenteille.

Funktiolle annetaan parametrina moduulin id ja TargetTypeOptions. TargetTypeOptions sisältää pudotusvalikon valintavaihtoehdot.

Funktion sisällä alustetaan myös HeaderData, josta löytyy olioista koostuva taulukko. Se sisältää Häiriintyvät kohteet -taulukon otsikot. Kohdekomponentti odottaa tiettyjä ominaisuuksia, jonka mukaisesti otsikot näytetään käyttöliittymässä. Tässä tapauksessa otsikoita on neljä kappaletta. Text-ominaisuus sisältää viittauksen lokalisointitiedostoihin, josta haetaan kulloinkin valitun kielen mukainen käännösteksti. Width-ominaisuus taas määrittää otsikon leveyden suhteessa taulukon leveyteen.

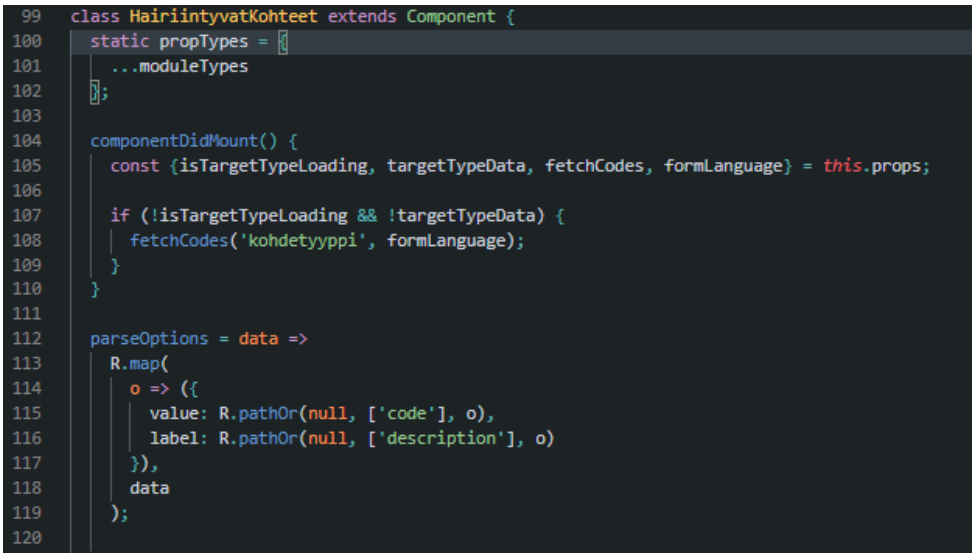
Kolmannessa otsikossa on käytetty HTML:n näköistä JSX-syntaksia, koska asiakas on toivonut, että aina kun käytetään metriyksiköitä, niin käytetään tiettyä tyylimääritystä. Vastaanottava komponentti ei osaa tyylimuutoksia tehdä ilman JSX-syntaksia. Tämän lisäksi propsina annetaan raw: true, jotta komponentti ei yritä tehdä käännöstä saadusta datasta. Tässä kohdassa käytetään React-Intl kirjaston FormattedMessage-komponenttia. FormattedMessagelle annetaan id, jonka perusteella se hakee käännöstekstin lokalisointitiedostosta.

Tämän jälkeen alustetaan cellRenderers-taulukko, jonka sisällä on jokaista saraketta kohden yksi apufunktiokutsu. Tässä kutsutaan tuotua cellRenderer-apufunktiota, joka auttaa muun muassa sarakkeita näyttämään tyylillisesti yhtenäiseltä. Ensimmäisessä cellRenderer-kutsussa annetaan toisena parametrina LabeledSelect-joka määrittää, että tässä solussa käytetään pudotusvalikko-komponenttia sisältönä. Kolmantena parametrina annetaan kohdetyyppivaihtoehdot.

```
72  const otherTargetsTableProps = moduleId => ({
73    headerData: [
74      {text: `${MODULE_ID}KuormittavatToiminnotNimi`, width: '60%'},
75      {
76        text: (
77          <div>
78            <span>
79              <FormattedMessage id={`${moduleId}KuormittavatToiminnotEtaisyysNumero`} />
80            </span>
81            <span className="ylva-table-header-units">
82              <FormattedMessage id={`${moduleId}MetriYksikko`} />
83            </span>
84          </div>
85        ),
86        width: '30%',
87        textAlign: 'center',
88        raw: true
89      }
90    ],
91    cellRenderers: [
92      cellRenderer('KuormittavatToiminnotNimi'),
93      cellRenderer('KuormittavatToiminnotEtaisyysNumero', 'number', NumberPicker)
94    ],
95    title: TABLE_NAME2,
96    moduleId
97  });
```

Kuva 24. Häiriintyvät kohteet -pääkomponentti kuva 3/6

Kuvassa 24 määritellään toisen Muut kuormittavat toiminnot -taulukon propsit vastaavaan tapaan kuin mitä Häiriintyvät kohteet -taulukossa tehtiin. Kyseessä on hyvin vastaavan kaltainen taulukko.



```
99  class HairiintyvatKohteet extends Component {
100   static propTypes = {
101     ...moduleTypes
102   };
103
104   componentDidMount() {
105     const {isTargetTypeLoading, targetTypeData, fetchCodes, formLanguage} = this.props;
106
107     if (!isTargetTypeLoading && !targetTypeData) {
108       fetchCodes('kohdetyyppi', formLanguage);
109     }
110   }
111
112   parseOptions = data =>
113     R.map(
114       o => ({
115         value: R.pathOr(null, ['code'], o),
116         label: R.pathOr(null, ['description'], o)
117       }),
118       data
119     );
120
```

Kuva 25. Häiriintyvät kohteet -pääkomponentti kuva 4/6

Kuvassa 25 nähdään varsinainen luokka HairiintyvatKohteet, joka pitää sisällään enemmän React-spesifistä koodia. Rivillä 99 alustetaan luokka, joka määritellään käyttämään Reactin Component-luokkaa ja sen sisältämiä funktioita ja toiminnallisuutta. Tämän jälkeen luokassa käytetään elinkaarimetodia nimeltään ComponentDidMount. ComponentDidMount-metodi ajetaan aina, kun komponentti kiinnitetään DOM-malliin. Useimmiten tämä tarkoittaa sitä, että metodi ajetaan silloin kuin komponentti tulee näkyville käyttöliittymään. Metodin sisällä alustetaan metodille annetut propsit johon viitataan this-sanalla. Propseja on tässä vaiheessa isTargetTypeLoading, targetTypeData, fetchCodes ja formLanguage. Osa näistä propseista on tuotu HairiintyvatKohteet-komponentin saataville käyttäen MapStateToProps-funktiota, joka on nähtävillä kuvassa 27 rivillä 211. MapStateToProps on osa React-Reduxin connect-funktiota, jolla React ja Redux saadaan toimimaan yhdessä. Tähän komponenttiin connect tuodaan osana formModule -moduulia, jossa se on määritelty valmiiksi. Eli sitä ei erikseen määritellä tässä komponentissa. MapStateToProps-funktiossa määritetään mitä tietoja globaalista tilasta halutaan komponentin ulottuville. Tässä tapauksessa kutsutaan getFormLanguage()-funktiota, joka tuo lomakkeella käytetyn kielen komponentin tietoon. Kutsutaan myös isTargetTypeDataLoading-funktiota, jotta saadaan tietoon, onko Häiriintyvät kohteet -koodistodata jo ladattu komponentille. Tämän lisäksi kutsutaan myös getCodeData()-funktiota, jonka tehtävä on hakea haluttu koodistodata. Nämä kaikki kutsutut funktiot ovat ”selektoreita”, jotka on tuotu komponentin saataville import-lausekkeella. MapStateToProps-funktiota kutsutaan joka kerta, kun Reduxin globaali tila päivittyy.

ComponentDidMountin sisällä tarkistetaan onko koodistohaku päällä sekä onko koodistodata haettu. Mikäli molemmat ehdot täyttyvät, niin

käynnistyy kutsutaan fetchCodes-actionia, jolle annetaan parametrina ”kohdetyyppi”-kirjainjono ja formLanguage-olio, joka sisältää lomakkeen kielen, jotta palautetut arvot ovat valmiiksi käännettynä oikealle kielelle. fetchCodes on YLVA-spesifinen action-tyyppinen funktio, jonka tarkoituksena on hakea tietty tietokannasta koodisto komponentin saataville. Tässä tapauksessa kyseessä on ”kohdetyyppi”-koodisto, jota hyödynnetään pudotusvalikossa Häiriintyvät kohteet -taulukossa.

Tämän jälkeen rivillä 126 alustetaan parseOptions-metodi. Tässä metodissa hyödynnetään RamdaJS-kirjaston kahta eri funktiota. Metodi ottaa parametrina vastaan taulukollisen olioita, jonka jokaisen olion se käy läpi käyttämällä Ramdan map-funktiota. Map-funktion sisällä hyödynnetään Ramdan pathOr-funktiota. Tarkoituksena on palauttaa uusi taulukko, joka on kohdekomponentin ymmärtämässä muodossa. Eli taulukon tulee sisältää olioita, joilla on ominaisuutena sekä value että label.

```
121   render() {
122     const {readOnly, targetTypeData} = this.props;
123     const tableProps1 = affectedTargetsTableProps(
124       MODULE_ID,
125       targetTypeData ? R.concat(defaultOption, this.parseOptions(targetTypeData)) : []
126     );
127     const tableProps2 = otherTargetsTableProps(MODULE_ID);
128
129     return (
130       <div className="form">
131         <Row>
132           <Col width="12">
133             <FieldArray
134               component={FormTable}
135               name={TABLE_NAME1}
136               props={{readOnly, ...tableProps1}}
137             />
138           </Col>
139         </Row>
140         <Row>
141           <Col width="12">
142             <FormattedMessage id={`${MODULE_ID}TargetMapText`} />
143             <hr className="mb-lg" />
144           </Col>
145         </Row>
146         <Row>
147           <Col width="12">
148             <FormInput
149               Component={LabeledCheckbox}
150               label={`${MODULE_ID}MuutKuormittavatToiminnotKytkin`}
151               name={`${MODULE_ID}MuutKuormittavatToiminnotKytkin`}
152             />
153           </Col>
154         </Row>
```

Kuva 26. Häiriintyvät kohteet -pääkomponentti kuva 5/6

Kuvassa 26 nähdään render()-metodin sisältöä. Kyseessä on ainoa Reactin Component-luokan pakollinen metodi. Riveillä 122-127 määritellään render-metodin propsit. Nostona tästä voidaan nostaa rivillä 123 alustettava tableProps1, johon tallennetaan affectedTargetsTableProps-funktion

palauttama arvo. Niin kuin aiemmin rivillä 41 nähdään, affectedTargets-TableProps ottaa vastaan kaksi parametria, moduleIdn ja targetTypeOptionsin. Rivillä 125 käytetätään JavaScriptin ehdollista operaattoria ja tarkistetaan, että onko targetTypeData olemassa. Jos se on olemassa, niin käytetään Ramdan concat-funktiota palauttamaan peräkkäin aseteltuna merkkijonona defaultOption-olio sekä parseOptionsin palauttama taulukko. Mikäli targetTypeData-muuttujaa ei ole olemassa, niin palautetaan tyhjä taulukko.

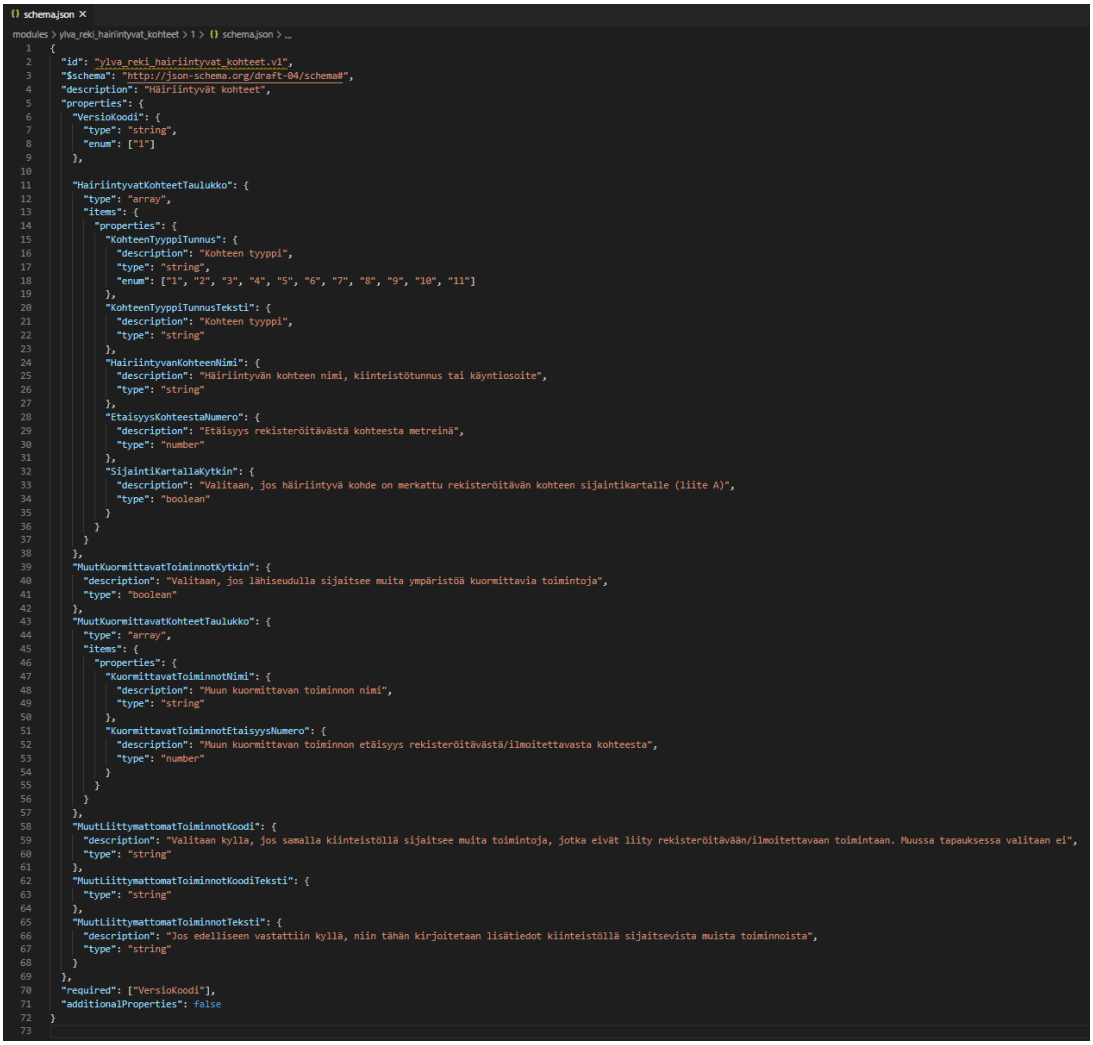
Return-lausekkeen sisällä käytetään JSX-syntaksia. Lausekkeen sisällä näytetään se mitä ruudulla näkyy, kun komponentti kiinnitetään DOM-malliin. Riville on kirjoitettu FieldArray-komponentti. Kyseessä on Redux-formin y komponentti, joka ottaa propsina sisäänsä toisen komponentin. FieldArray sisältää toiminnallisuutta mm taulukon sisältämien kenttien lisäämiseen reduxin tilaan ja tallentamiseen. FieldArrayn sisälle annetaan propsina FormTable-niminen komponentti. FormTable on SPAv2-projektille tietojen syöttöä varten luotu lomaketaulukko-komponentti. Rivillä 148 käytetään FormInput-komponenttia, joka ottaa vastaan propsina toisen komponentin. FormInput on SPAv2 ja YLVAssa hyvin usein käytetty ”wrapper”-komponentti. Sen käyttö helpottaa ja nopeuttaa moduulien koodaamista sekä muiden kehittäjien koodin lukemista, koska kyseessä on johdonmukaisesti luotua koodia. FormInput sisältää muun muassa komponentin kommunikoimisen Reduxin kanssa. Tässä esimerkissä propsina syötetty komponentti on LabeledCheckbox. Kyseinen komponentti näyttää käyttöliittymällä valintaruudun, jonka yläpuolella on teksti. Kyseinen teksti haetaan lokalisointitiedostoista sille annetulla id:llä. Esimerkin tuottama teksti on ”Lähiseudulla sijaitsee muita ympäristöä kuormittavia toimintoja”. Tähän valintaruutuun liittyy logiikkaa, josta lisää ConditionalRender-komponentin kuvauksessa.

```
155          <Row className="mt-sm">
156            <Col width="12">
157              <ConditionalRender
158                dependOn={{moduleId: MODULE_ID, field: `${MODULE_ID}MuutKuormittavatToiminnotKytkin`}}
159                renderIf={val => val === true}
160                moduleId={MODULE_ID}
161              >
162                <FieldArray
163                  component={FormTable}
164                  name={TABLE_NAME2}
165                  props={{readOnly, ...tableProps2}}
166                />
167              </ConditionalRender>
168            </Col>
169          </Row>
170          <br />
171          <Row>
172            <Col width="12">
173              <PureLabel label={`${MODULE_ID}MuutLiittymattomatToiminnotOhjeTeksti`} />
174              <FormInput
175                Component={RadioGroup}
176                name={`${MODULE_ID}MuutLiittymattomatToiminnotKoodi`}
177                options={muutLiittymattomatToiminnotOptions}
178              />
179            </Col>
180            <ConditionalRender
181              dependOn={{
182                moduleId: MODULE_ID,
183                field: `${MODULE_ID}MuutLiittymattomatToiminnotKoodi`
184              }}
185              renderIf={val => val === 'kylla'}
186              moduleId={MODULE_ID}
187              clearFields={`${MODULE_ID}MuutLiittymattomatToiminnotTeksti`}
188            >
189              <Col width="12">
190                <FormInput
191                  Component={LabeledTextarea}
192                  name={`${MODULE_ID}MuutLiittymattomatToiminnotTeksti`}
193                  label={`${MODULE_ID}MuutLiittymattomatToiminnotTeksti`}
194                />
195              </Col>
196            </ConditionalRender>
197          </Row>
198        </div>
199      );
200    }
201  }
202
203  HairiintyvatKohteet.propTypes = {
204    readOnly: PropTypes.bool,
205    fetchCodes: PropTypes.func.isRequired,
206    formLanguage: PropTypes.string,
207    isTargetTypeLoading: PropTypes.bool.isRequired,
208    targetTypeData: PropTypes.array
209  };
210
211  const mapStateToProps = state => ({
212    formLanguage: getFormLanguage(state),
213    isTargetTypeLoading: isCodeDataLoading('kohdetyyppi')(state),
214    targetTypeData: getCodeData('kohdetyyppi')(state)
215  });
216
217  const mapDispatchToProps = dispatch => bindActionCreators({fetchCodes}, dispatch);
218
219  export default formModule({
220    id: MODULE_ID,
221    versions: ['1'],
222    mapStateToProps,
223    mapDispatchToProps,
224    showInfo: true
225  })(HairiintyvatKohteet);
```

Kuva 27. Häiriintyvät kohteet -pääkomponentti kuva 6/6

Kuvassa 27 rivillä 157 käytetään ConditionalRender-apukomponenttia, joka ottaa vastaan dependOn-nimisen propsin. Tämä apukomponentti on riippuvainen MuutKuormittavaToiminnotKytkin-nimisestä lomakekentästä. Kyseessä on valintaruutu, johon viitattiin edellisessä kappaleessa. ConditionalRender-apukomponentti ottaa vastaan myös RenderIf-nimisen propsin. Siinä tarkisteaan onko viitattavan lomakekentän arvo tosi. Jos arvo on tosi, niin ConditionalRender-komponentin sisällä oleva JSX-koodi käännetään selaimelle näytettävään muotoon. Mikäli arvo on epätosi, niin JSXää ei näytetä. Conditional Render-komponenttia käytetään toistamiseen rivillä 180.

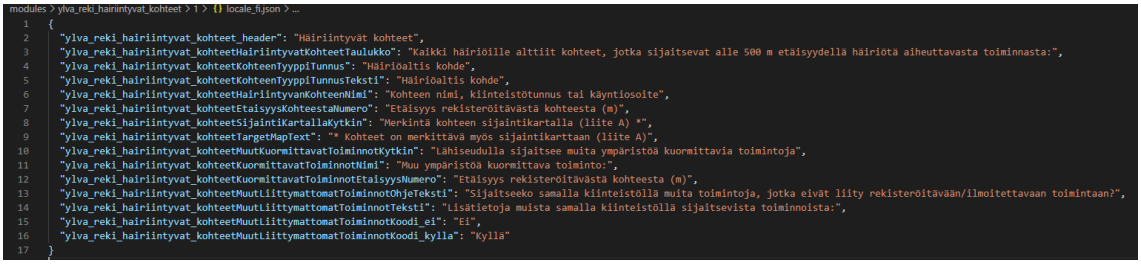
Komponentin tyyppien tarkistukseen käytetään Reactin omaa prop-types-kirjastoa. Mikäli jokin ominaisuus annetaan komponentille vääränä tyyppinä, niin kehittäjälle ilmoitetaan tästä selaimen konsolinäkymässä. Rivillä 219 on export default -lauseke, jossa komponentti/moduuli viedään muiden sitä tarvitsevien komponenttien saataville.



Kuva 28. Valmis schema.json -tiedoston sisältö

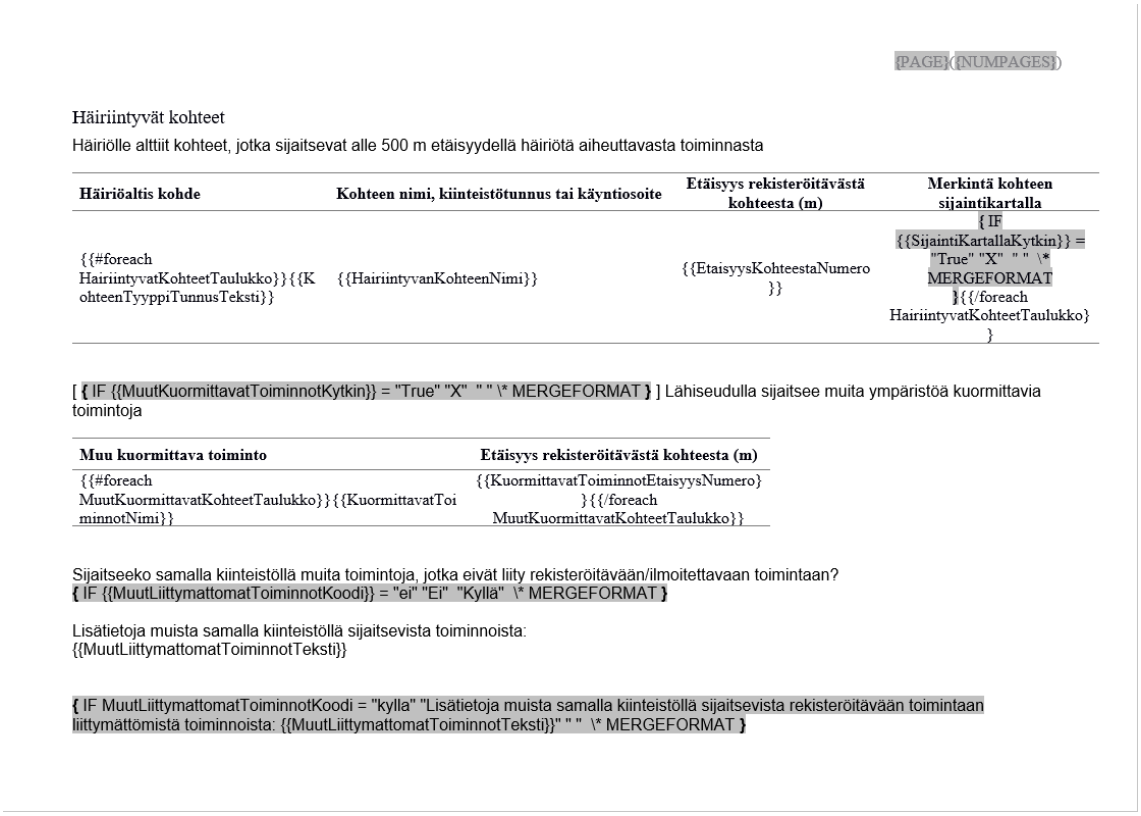
Kuvassa 28 nähdään valmis schema.json tiedoston sisältö, jota validoidaan JSON-scheman sisältöä vastaan. Tarkistetaan siis, että lähetettävän JSON-

datan formaatti on scheman mukainen ja data on oikeassa muodossa. Schema.json tiedostossa näkyvät nimet määritellään lomakekentissä name-ominaisuudella, jota verrataan schemaan. Esimerkiksi riveillä 28–30 määritellään EtaisyysKohteestaNumero-kentän tyypiksi numeron. Mikäli kyseessä olisi merkkijono, niin lomaketta lähetettäessä käyttäjä saisi validointivirheen. Description-kentässä on ohjeet sellaisille käyttäjille, jotka käyttävät lomakkeiden lähettämistä YLVAn rajapinnan kautta ilman SPAv2 alustaa. Tällaisia voivat olla esimerkiksi isot tehtaat, joille lomakkeiden täyttäminen olisi työlästä.



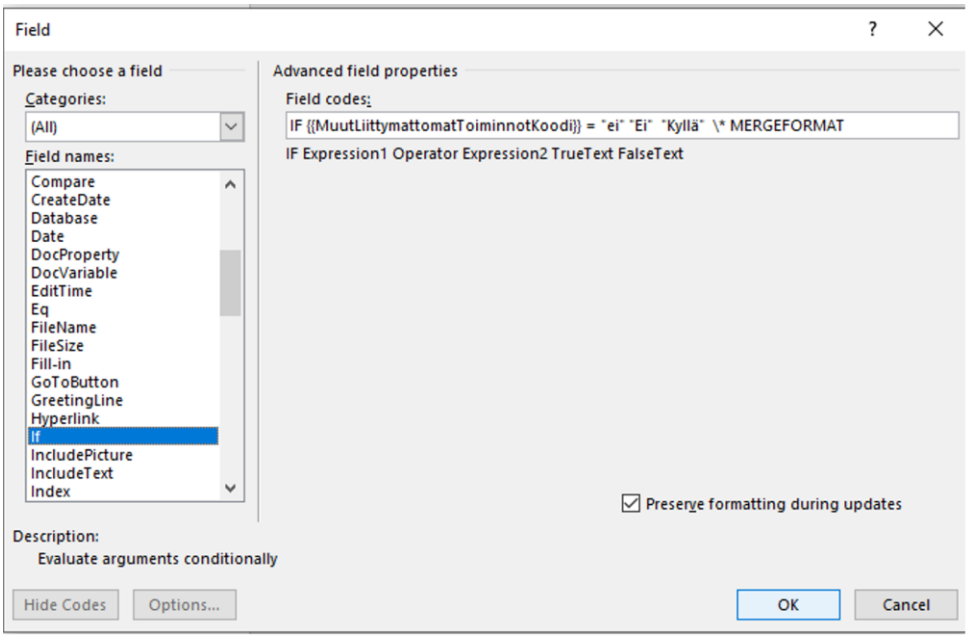
Kuva 29. Locale_fi.json -tiedoston sisältö

Kuvassa 29 nähdään suomenkielisen käännöstiedoston sisältö. Tästä tiedostosta muun muassa react-intl-kirjasto hakee käännöstekstit käyttämällä label-propsia tunnisteena. Sinisellä tekstillä näkyy id ja oranssilla varsinainen käännösteksti.



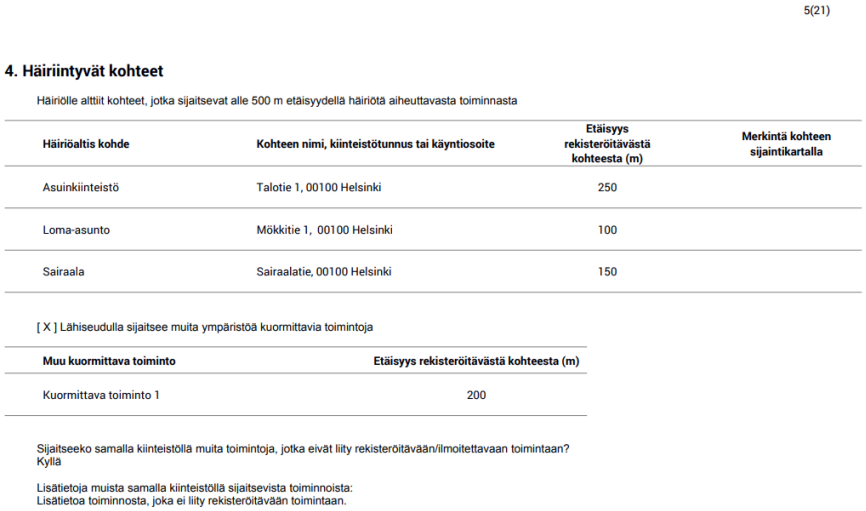
Kuva 30. Moduulin Word-sapluuna Asposen käsiteltäväksi

Kuvassa 30 nähdään Word-sapluuna, jossa käytetään Asposen määrittämää syntaksia. Tätä sapluunaa muokataan Word-editorilla. Word-sapluunassa näkyvät nimet kuten ’HairiintyvanKohteenNimi’ identifioivat haettavan lomakekentän tiedot Word-sapluunaan.



Kuva 31. Esimerkki Field-kentän sisältämästä syntaksista Wordissa

Kuvassa 31 näytetään myös Wordin Field codes -syntaksia. Koodi upotetaan Word-sapluunan sisälle. Tässä tapauksessa tarkistetaan ’MuutLiittymattomatToiminnotKoodi’ kentän arvo. Mikäli se on ’ei’ niin tulostetaan ”Ei”. Muussa tapauksessa tulostetaan ”Kyllä”.



5(21)

4. Häiriintyvät kohteet

Häiriölle alttiit kohteet, jotka sijaitsevat alle 500 m etäisyydellä häiriötä aiheuttavasta toiminnasta

Häiriöaltis kohde	Kohteen nimi, kiinteistötunnus tai käyntiosoite	Etäisyys rekisteröitävästä kohteesta (m)	Merkintä kohteen sijaintikartalla
Asuinkiinteistö	Talotie 1, 00100 Helsinki	250	
Loma-asunto	Mökkitie 1, 00100 Helsinki	100	
Sairaala	Sairaalatie, 00100 Helsinki	150	

[X] Lähiseudulla sijaitsee muita ympäristöä kuormittavia toimintoja

Muu kuormittava toiminto	Etäisyys rekisteröitävästä kohteesta (m)
Kuormittava toiminto 1	200

Sijaitseeko samalla kiinteistöllä muita toimintoja, jotka eivät liity rekisteröitävään/ilmoitettavaan toimintaan?
Kyllä

Lisätietoja muista samalla kiinteistöllä sijaitsevista toiminnoista:
Lisätietoa toiminnosta, joka ei liity rekisteröitävään toimintaan.

Kuva 32. Kuvakaappaus tulostetun PDF-lomakkeen Häiriintyvät kohteet -moduulin osasta

Kuvassa 32 nähdään kuvakaappaus tulostetun PDF-lomakkeen Häiriintyvät kohteet -moduulin osasta. Kuvakaappaus on osa PDF-raporttia, jonka käyttäjä voi ladata itselleen ennen lomakkeen lähettämistä. Raportti voidaan ladata myös lomakkeen lähettämisen jälkeen.

7 LOPPUTULOS

Lopputulokseksi saatiin asiakkaan tilaama moduuli valmiiksi. Vaikuttaa, että kyseessä on aikaisempaa käyttäjäystävällisempi versio lomakkeen täyttämisprosessista. Sprint demossa asiakas hyväksyi moduulin tämänhetkisen version. Häiriintyvät kohteet -moduuli on vain yksi osa REKI-asioinnin kokonaisuutta, joten kokonaisarvio saadaan vasta loppukäyttäjiltä, kun asiointi otetaan kokonaisuudessaan käyttöön syksyllä 2020. Ennen sitä REKI-asioinnin käyttöliittymä menee testikäyttäjille testikierrokselle ennen tuotantoversioon viemistä. Ja kuten ketterien menetelmien työkaluihin kuuluu, niin tämä moduuli voi vielä muuttaa muotoaan myöhemminkin.

Moduulin toteuttamisessa käytettiin jonkin verran jo vanhentunutta React-tekniikkaa, koska React-versio on suhteellisen vanha. Myös moni muu JavaScript-lisäkirjasto kuten Redux, React-intl ja Redux-form ovat omalta osaltaan vanhentuneita. Tämän myötä esimerkiksi Reactin osalta täytyy käyttää luokkapohjaisia komponentteja funktionaalisten komponenttien sijaan. Tällöin ei ole mahdollista kirjoittaa mukautettuja hooks-metodeja. Tämä saattaa vaikuttaa heikentävästi kehittäjäkokemukseen (DX/developer experience), kun ei voida käyttää uusinta versiota Reactista. Nämä asiat asettavat rajoituksia Reactin kirjoittamiseen. Myös React-Intl kirjastosta jää muutamia uusimpia ominaisuuksia hyödyntämättä, koska hookseja ei voida käyttää.

Projekteissa hyödynnetään Entity Frameworkia ja sen osalta Database-First lähestymistapaa, jossa ensin suunnitellaan ja luodaan tietokanta ja vasta sen jälkeen voidaan luoda tietokantaa mallintava luokkarakenne C#-kielellä. Entity Framework tukee nykyään myös Code-First lähestymistapaa, jolloin ensiksi kirjoitetaan luokkarakenne C#-kielellä. Tämän jälkeen tehdään migraatio, joka luo luokkarakenteen pohjalta automaattisesti tietokantataulut. Tämä helpottaa huomattavasti kehittäjän työtä. Code-First-lähestymistavan käyttöönotto saattaisi parantaa kehittämisen tehokkuutta ja mielekkyyttä.

Näiden kehitysideoiden osalta täytyy ottaa huomioon vanhan koodin asettamat rajoitukset, sekä uusien versioiden ja tekniikoiden käyttöönotto. Uusien versioiden käyttöönotto veisi todennäköisesti paljon aikaa ja taloudellisia resursseja, ettei koodikannan refaktorointia kokonaisuudessaan kannata tehdä. Uuden ja vanhan yhteensovittamisessa täytyy ottaa huomioon myös muut SPAv2-alustaa käyttävät asiakasorganisaatiot YLVAn lisäksi. Esimerkiksi uuden React-version päivittäminen vaikuttaisi myös näihin muihin projekteihin. Toki YLVA-projektissa on aiemminkin tehty olemassa olevien ominaisuuksien refaktorointia uudelle tekniikalle. Esimerkiksi aiemmin käyttöliittymän osalta on hyödynnetty Microsoftin MVC-tekniikkaa ennen siirtymistä Single-page-application arkkitehtuuriin ja Reactiin.

Toteutettu komponentti on asiakkaan vaatimusten ja määrittelyjen mukainen. Komponentti on toteutettu niin geneerisesti, että sitä pystytään hyödyntämään useissa eri asioinneissa. Häiriintyvät kohteet -moduuli on yksi REKI-asioinnin yhteisistä moduuleista, joten pelkästään tämä määritelmä täyttää uudelleenkäytettävyyden kriteerit. Rakennettua komponenttia voidaan käyttää joko osittain tai kokonaan SPAv2-alustalla käyttöliittymän rakennuspalikkana.

8 YHTEENVETO

Tämän työn tavoitteena oli suunnitella ja toteuttaa lomakemoduuli asiakkaan sähköiseen asiointijärjestelmään. Tavoite täyttyi ja asiakkaalle toimitettiin tilattu moduuli. Työn tarkoituksena oli tarjota tietoa modernin web-asiointipalvelun toteuttamisesta julkishallinnolle. Tämä tarkoitus täytettiin myös mielestäni hyvin.

Olen tyytyväinen työn lopputulokseen. Työn tekeminen oli haastavaa ja samalla opettavaa. Opin työn tekemisessä hyvin paljon lisää etenkin ReactJS:n ja JavaScriptin teoriasta. Myös Redux-kirjastoon liittyvä osaaminen syventyi osaltani entisestään. Teoriaosaamiseni eri projektinhallintamenetelmistä syveni. Etenkin ketterät menetelmät ja Scrum-prosessina avautui itselle entistä enemmän. Opin paljon myös kommentoimaan ja dokumentoimaan kirjoittamaani koodia. Tämä on hyvä ominaisuus modernille sovelluskehittäjälle.

Työssä otettiin kantaa siihen mitä asioita pitää ottaa huomioon osittain vanhentuneella tekniikalla toteutetun järjestelmän sekä uuden komponentin yhteensopivuudessa. Tätä käsiteltiin etenkin lopputulosta arvioitaessa. Työssä käytiin myös läpi myös uusia tekniikoita ja sitä, mitä lisäarvoa ne voisivat tuoda komponentin toteutukseen. Rakennetun komponentin osien uudelleenkäytettävyyttä YLVAssa arvioitiin myös useissa eri työn vaiheissa.

LÄHTEET

Aspose.com (n.d.) .NET Apis to Process Word Documents. Haettu 7.8.2020 osoitteesta: https://products.aspose.com/words/net

Auer, L., Heinäsmäki, M., Hölttä, J., Kalliala, E., Laanti Maarit, Laine, K., . . . Auer, A. (2013). Ketterää kehitystä. [Helsinki]: Finn Lectura.

Chiarelli, Andrea. 2018. Beginning React – Simplify your frontend development workflow and enhance the user experience of your applications with React. Packt Publishing.

Entityframeworktutorial.net (2020). What is Entity Framework. Haettu 27.8.2020 osoitteesta: https://www.entityframeworktutorial.net/what-is-entityframework.aspx

Finlex (2020). Ympäristönsuojelulaki. Haettu 9.7.2020 osoitteesta https://www.finlex.fi/fi/laki/ajantasa/2014/20140527

FreeCodeCamp (2017). Yes, React is taking over front-end development. The question is why. Haettu 12.7.2020 osoitteesta: https://www.freecodecamp.org/news/yes-react-is-taking-over-front-end-development-the-question-is-why-40837af8ab76/

Haikala, I. & Mikkonen, T. (2011). Ohjelmistotuotannon käytännöt (12. uud. p.). Helsinki: Talentum.

Hoque, Shama. 2018. Full-Stack React Projects: Modern Web Development Using React 16, Node, Express, and MongoDB. Packt Publishing.

Honkanen, Joni. 2017. ReactJS. Opinnäytetyö. Seinäjoen ammattikorkeakoulu. Haettu 12.7.2020 osoitteesta: https://www.theseus.fi/bitstream/handle/10024/138247/Honkanen_Joni.pdf

JSON Schema (2020). Understanding JSON Schema. Haettu 7.8.2020 osoitteesta: http://json-schema.org/understanding-json-schema/

MDN web docs (2020). JavaScript. Haettu 4.8.2020 osoitteesta: https://developer.mozilla.org/fi/docs/Web/JavaScript

Medium (2020). Top 25 Companies/Brands Using ReactJS Development. Haettu 12.7.2020 osoitteesta: https://medium.com/front-end-weekly/top-25-companies-brands-using-reactjs-development-8be87b32cec2

Medium, Federico Knüssel (2017). A look at the inner workings of Redux. Haettu 4.8.2020 osoitteesta: https://medium.com/@fknussel/redux-3cb5aac94a66

Myllymäki, R., Hinkka, T., Hirvensalo, J. & Hämäläinen, J. (2015). Onnistunut tietojärjestelmäprojekti: Osa 1, Neuvoja tietojärjestelmää hankkivalle (2. p.). [Vantaa]: Ketterät kirjat.

Pierce, Benjamin C. 2002. Types and Programming Languages. Cambridge, Massaschusetts: The MIT Press

RabbitMQ (n.d.). RabbitMQ is the most widely deployed open source message broker. Haettu 7.8.2020 osoitteesta: https://www.rabbitmq.com/

RamdaJS.com (n.d.). Ramda. A practical functional library for JavaScript Programmers. Haettu 31.8.2020 osoitteesta: https://ramdajs.com/

Redux (2020). What is Redux? Haettu 4.8.2020 osoitteesta: https://redux.js.org/tutorials/essentials/part-1-overview-concepts#what-is-redux

Reactjs.org. (n.d.). Virtual DOM and Internals. Haettu 30.6.2020 osoitteesta: https://reactjs.org/docs/design-principles.html

Reactjs.org. (n.d.). Virtual DOM and Internals. Haettu 2.7.2020 osoitteesta: https://reactjs.org/docs/faq-internals.html

Reactjs.org. (n.d.). Introducing JSX. Haettu 15.7.2020 osoitteesta: https://reactjs.org/docs/introducing-jsx.html

Reactjs.org. (n.d.). Components and props. Haettu 16.7.2020 osoitteesta: https://reactjs.org/docs/components-and-props.html

Reactjs.org. (n.d.). React component. Haettu 20.7.2020 osoitteesta: https://reactjs.org/docs/react-component.html

Sqlservertutorial.net (2020). What is SQL Server. Haettu 27.8.2020 osoitteesta: https://www.sqlservertutorial.net/getting-started/what-is-sql-server/

Ympäristöhallinnon yhteinen verkkopalvelu (2013). Rekisteröintimenettely. Haettu 6.7.2020 osoitteesta https://www.ymparisto.fi/fi-FI/Asiointi_luvat_ja_ymparistovaikutusten_arviointi/Luvat_ilmoitukset_ja_rekisterointi/Ymparistonsuojelulain_mukainen_rekisterointi

Ympäristöhallinnon yhteinen verkkopalvelu (2013). Ympäristölupa. Haettu 6.7.2020 osoitteesta https://www.ymparisto.fi/fi-

FI/Asiointi_luvat_ja_ymparistovaikutusten_arviointi/Luvat_ilmoitukset_ja_rekisterointi/Ymparistolupa

Ympäristöhallinnon yhteinen verkkopalvelu (2013). Ympäristönsuojelun valvonnan sähköinen asiointijärjestelmä YLVA. Haettu 8.7.2020 osoitteesta https://www.ymparisto.fi/fi-FI/Kartat_ja_tilastot/Tietojarjestelmat/Ymparistonsuojelun_valvonnan_sahkoinen_asiointijarjestelma_YLVA

W3Schools (2020). What is React? Haettu 12.7.2020 osoitteesta: https://www.w3schools.com/whatis/whatis_react.asp

W3Schools.com (2020). C# Tutorial. Haettu 4.8.2020 osoitteesta: https://www.w3schools.com/cs/

W3Schools.com (2020). Bootstrap Grid System. Haettu 17.7.2020 osoitteesta: https://www.w3schools.com/bootstrap/bootstrap_grid_system.asp

HAASTATTELUT

Laurila, P. (2020). Projektipäällikkö, Hämeen ELY-keskus. Haastattelu 27.5.2020.