SÄHKÖISTEN LOMAKKEIDEN KÄSITTELYPROSESSI

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

HAMK Riihimäki, Tieto- ja Viestintätekniikka

Kevät, 2020

Panu Siik

Tieto- ja viestintätekniikka
HAMK Riihimäki

Tekijä Panu Siik **Vuosi** 2020

Työn nimi Sähköisten lomakkeiden käsittelyprosessi

Työn ohjaaja/t Toni Laitinen

TIIVISTELMÄ

Digitalisoimalla lomakkeita sähköiseen muotoon voidaan tehostaa ja nopeuttaa käsittelyprosesseja saaden samalla myös taloudellisia hyötyjä.

Tämän opinnäytetyön tarkoituksena on luoda käsittelyprosessit uusille lomaketyypeille, jotka toimivat joiltain osin automaattisesti. Työssä käydään läpi projektissa käytettyjä teknologioita, sekä pieniä esimerkkejä niiden käytöstä. Työn tarkoituksena ei ole antaa valmista ratkaisua, vaan tarjota ideoita liittyen hyödyllisiin teknologioihin.

Käytännön osuus toteutettiin Alfame Systems Oy:n toimesta alihankintana KEHA-keskukselle osana YLVA-projektia, johon luotiin prosessiputki joka helpottaa Elinkeino-, liikenne- ja ympäristökeskuksen vastaanottamien ympäristöraporttien ja ilmoitusten käsittelyä.

Avainsanat C#, JSON, React, Redux, .Net

Sivut 29 sivua, joista liitteitä 4 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information & Communications Technology
HAMK Riihimäki

Author Panu Siik **Year** 2020

Subject Processing of Digital Forms

Supervisors Toni Laitinen

ABSTRACT

By converting paper forms into a digital from, it is possible to streamline handling processes making the processing times shorter and expenses smaller.

The goal of this thesis is to create a processing pipeline, which to some extent works automatically. Technologies used in the project are also walked through with some examples and links to more documentation. The goal of this thesis is not to give a ready-made solution but provide ideas about useful technologies.

The practical part of this thesis was carried out by Alfame Systems Oy as subcontractor to the KEHA-Center by creating a process pipeline for handling environmental reports and notifications which The Centre for Economic Development, Transport and the Environment (ELY Centre) receives.

Keywords React, Redux, JSON, C#, .Net

Pages 29 pages including appendices 4 pages

SISÄLLYS

SANASTOA

API	Ohjelmointirajapinta
Back-end	Sovelluksen taustalla toimiva logiikka ja palvelut.
Front-end	Sovelluksen visuaalinen käyttöliittymä
Framework	Runkorakenne, esim. AngularJS
JHR	Hakemus jätteen ammattimaisen kuljettamisen tai välittämisen hyväksymiseksi jätehuoltorekisteriin JL 94 §. Sisältää yhden hakemuslomakkeen, josta palautetaan asiakkaalle rekisteriote ja päätös.
JSON	JSON (JavaScript Object Notation) on kevyt tiedon välittämisformaatti, joka on kehittäjälle helposti luettava ja kirjoitettava. Tämä formaatti koostuu avain (*key*) / arvo (*value*) pareista.
JSON Schema	JSON Schema on yleisesti käytetty työkalu JSON-datan sisällön kuvailuun ja validointiin.
Kohde	Jokaiselle yritykselle käsittelyjärjestelmässä luodaan kohde, joka on päätaso yrityksen osioille ja pisteille.
Library	Kirjasto, esim. ReactJS
MARA	Ilmoitus jätteiden hyödyntämisestä maarakentamisessa VNa 843/2017. Sisältää ilmoitus ja loppuraportti -lomakkeet.
Osio	Kohteella voi olla monia osioita, mutta vain yksi pääosio. (Esim. tehtaita ympäri Suomea, joista yksi on pääosio)
PIMA	Pilaantuneen maaperän ja pohjaveden puhdistaminen sekä selvittäminen (YSL 527/2014, 135-136§). Lomakekokonaisuus, joka sisältää kolme erillistä lomaketta: Maaperän ja pohjaveden pilaantuneisuuden ja

	puhdistamistarpeen tutkimusraportti, ilmoitus, sekä loppuraportti
Piste	Osiolla voi olla lukuisia pisteitä, joiden toimintaa raportoidaan. (Esim. Tehtaiden piiput tai kattilat)
SPAv2 (Asiointialusta)	Aluehallinnon sähköinen asiointipalvelu, jossa asiakas täyttää lomakkeet ja pystyy seuraamaan lomakkeiden käsittelyn vaihetta. Täältä asiakas myös saa ladattua käsittelyn jälkeen tarvitsemansa päätökset tai rekisteriotteet.
USPA	Palvelu, jossa viimeistellään ja arkistoidaan toimenpiteiden sisältämät asiakkaan lähettämät, sekä asiakkaalle toimitettavat dokumentit
Vastuuvalvoja	Valtion työntekijä, jonka vastuulla on valvoa yrityksen raportointia.
Worker	Luokka, joka on automatisoitu osa prosessointikoodia, joka esimerkiksi hakee tietokannasta arkistorivejä lomaketyypin ja tilan perusteella. Näitä voidaan ketjuttaa kattamaan prosessin vaiheita.
YLVA (Käsittelyjärjestelmä)	Ympäristönsuojelun tietojärjestelmän valvontaosa. Toimii työkaluna kohteiden tietojen ylläpidolle, lomakkeiden käsittelylle ja virallisten dokumenttien laatimiselle asiakkaalle

1 JOHDANTO

Asiakasyritys vastaanottaa vuosittain tuhansia ilmoituksia ja hakemuksia liittyen ympäristöasioihin. Näitä ilmoituksia, raportteja ja hakemuksia on jo vuosien ajan muunnettu digitaaliseen muotoon, jotta vastaanotettuja tietoja olisi mahdollista hallita paremmin ja hyödyntää tehokkaammin esimerkiksi tilastoinnissa. Käytännön osuus toteutettiin KEHA-keskukselle alihankintana Alfame Systems Oy:n toimesta osaksi jo toimivaa YLVA-käsittelyjärjestelmää.

Opinnäytetyö on perusversio uusien JHR, MARA ja PIMA -ilmoitusten, -hakemusten ja -raporttien käsittelyprosesseista asiakasyritykselle. Prosessit alkavat erilliseltä SPAv2-asiointialustalta, jossa sähköiset lomakkeet täytetään ja lähetetään YLVA-käsittelyjärjestelmään mahdollisten liitteiden kera.

Tämän jälkeen suurin osa prosessista tapahtuu YLVA-käsittelyjärjestelmässä, jonka avulla ilmoitukset tai hakemukset käsitellään ja laaditaan tarvittavat asiakirjat asiakkaalle. Edellä mainitut asiakirjat arkistoidaan USPA-palveluun, joka on asiakirjojen viimeistely- ja säilytyspaikka.

Kun asiakirjat on käsitelty USPAssa, käsittelyjärjestelmän automatisoitu prosessi hakee viimeistellyt asiakirjat, joihin on liitetty mukaan merkintä sähköisestä hyväksymisestä ja palauttaa asiakirjan/asiakirjat asiakkaalle asiointipalveluun. Edellä mainittuun prosessiin liittyen liite 1 sisältää prosessikaavion, joka selventää siirtymiä näkymästä toiseen ja prosessin kulkua.

Työn tavoitteena on edistää julkishallinnon digitalisaatiotavoitteita korvaamalla paperisia lomakkeita sähköisillä versioilla ja automatisoida prosessin vaiheita siinä missä mahdollista. Tämän myötä vastuuvalvojien käsittelyyn käyttämä aika lyhenee merkittävästi, josta seuraamuksena on taloudelliset ja muut hyödyt.

Valtiovarainministeriön sivuilla aiheesta kirjoitetaan seuraavanlaisesti julkishallinnon sähköisten palvelujen osalta:

”Sähköiset palvelut lisäävät kansalaisten, yritysten ja yhteisöjen mahdollisuuksia käyttää julkisia palveluja ajasta ja paikasta riippumatta. Sähköinen asiointi on yleensä helpoin ja nopein tapa hoitaa viranomaisasioita. Kun sähköisten palvelujen käyttö lisääntyy, julkinen palvelutuotanto tehostuu ja yhteisiä verovaroja säästyy. Lähtökohtana on,

että julkisen hallinnon sähköiset palvelut ovat toimivia, helppokäyttöisiä ja turvallisia.

Julkisen hallinnon asiakkuusstrategian mukaan viranomaisten tulee huolehtia siitä, että sähköinen kanava on asiakkaalle houkuttelevin vaihtoehto. Sähköisten palvelujen rakentamisessa keskeistä on käyttäjäkeskeinen suunnittelu, palveluprosessien uudistaminen, palvelujen yhteen toimivuus sekä tietoturva- ja tietosuoja." (Valtiovarainministeriö, n.d.)

Valtioneuvoston kanslian julkaisemasta selvityksestä löytyy tämän kappaleen loppuun liitetty taulukko (Taulukko 1), joka kuvaa hyvin julkishallinnon saavuttamia hyötyjä digitalisaatiosta. Taulukossa mitattavat asiat jakautuvat viiteen osaan: Suorat rahalliset hyödyt, tehokkuuden parantuminen, parantunut palvelun tuottaminen, parantunut työtyytyväisyys (virkailijat ym.) ja parantunut päätöksenteko ja demokratia.

Suoria rahallisia hyötyjä verrattuna aiempaan, tutkimuksessa havaittiin säästöt tukiresursseissa, tietojärjestelmäkuluissa, tukikuluissa, matkakuluissa, julkaisu- ja jakelukuluissa, maksuissa kansainvälisille organisaatioille.

Verojen keräyksen osalta huomattiin aiempaan verrattuna suurempi/tarkempi verojen keräys, joka myös näkyi vähentyneenä harmaana taloutena ja petoksina.

Tehokkuus parantui työajan säästöjen, palvelun tarpeen vähentymisen, tulevien projektien kustannuksien ja resurssien käytön osalta. Lisätuloja saatiin kaupallisten palvelujen ja datan käytöstä.

Palvelujen laatu myös parani, joka ilmeni parantuneena asiakaspalveluna, yhdenmukaisuutena ja tasa-arvoisuutena, käyttäjätyytyväisyytenä, kommunikointina julkishallinnon toimijoiden välillä, etujen hyödyntämisenä, maineena ja käyttäjien luottamuksena, oppimisen ja kokemusten jakamisena eri toimijoiden välillä, sekä digitaalisten palvelujen käyttökynnyksen alenemisena. Työtyytyväisyys parani mielekkäämpien työtehtävien, parantuneen riskienhallinnan ja turvallisuuden myötä.

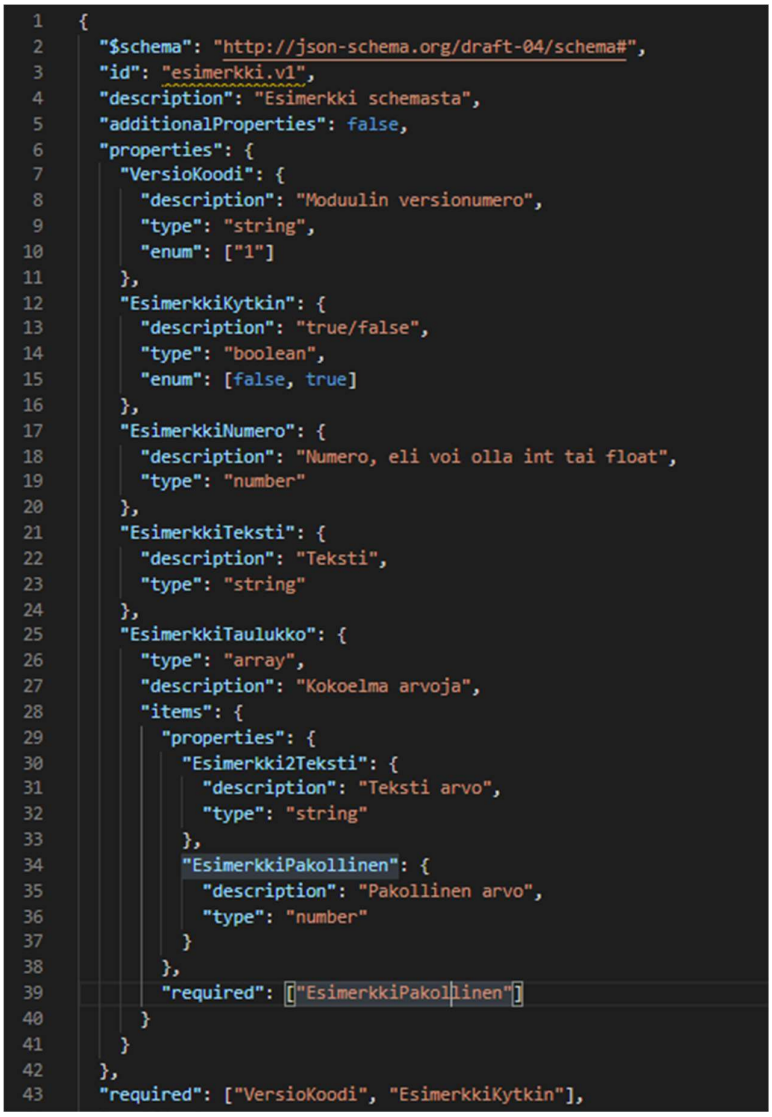
Päätöksenteko ja demokratia parani paremman sääntelyn ja vaikutusten yhteyden sekä parempana kansalaisten osallistumisen ja läpinäkyvyyden myötä.

Taulukko 1. Julkishallinnon saamat kustannushyödyt: mitattavat asiat, osa-alueet ja mittarit (Parviainen, Kääriäinen, Honkatukia & Federley, 2017, s.54)

Mitattava asia	**Osa-alueet**	**Mittarit**
Suorat rahalliset hyödyt	Säästöt tukiresursseissa	Muutos (esim. ed. vuoteen verrattuna): - tietojärjestelmäkuluissa (lisensiointikustannukset, ylläpitokustannukset) - muissa resurssikuluissa (rakennuksiin liittyvät kulut, palvelinkustannukset ym.) - tukikuluissa - matkakuluissa - julkaisu ja jakelukuluissa - makuissa kansainvälisille organisaatioille
	Suurempi/tarkempi verojen keräys (vähentynyt harmaa talous ja petokset)	Arvio lisääntyneestä verotulosta
	Lisääntyneet ulkoiset tulot	Lisätulot kaupallisten palvelujen ja datan käytöstä
Tehokkuuden parantuminen	Työajan säästöt	Muutos palvelujen tuottamiseen käytetyssä työmäärässä Muutokset valitusten määrissä ja uudelleen käsittelyssä
	Palvelun tarpeen vähentyminen	Palvelun kokonaiskäyttömäärän muutos Palvelun tuottamiseen käytetyt resurssit
	Aiemmat kustannukset tulevissa projekteissa	Arvio kustannussäästöissä tulevaisuudessa
	Resurssien käytön tehokkuus	Arvio päällekkäisyyksien määrän muutoksesta Kapasiteettien käyttömäärät
Parantunut palvelun tuottaminen	Parantunut asiakaspalvelu Parantunut palvelun yhdenmukaisuus ja tasa-arvoisuus Parantunut käyttäjätyytyväisyys Parantunut kommunikointi julkishallinnon toimijoiden välillä Parempi etujen hyödyntäminen Parempi maine ja käyttäjien luottamus Oppiminen ja kokemusten jakaminen eri toimijoiden välillä Digitaalisten palvelujen käyttökynnyksen aleneminen	Muutokset asiakastyytyväisyydessä (kysely) Muutokset valitusten määrissä ja uudelleen käsittelyssä Muutokset julkishallinnon maineessa (kysely) Muutokset palvelujen käyttäjien määrissä Muutokset palvelun läpimenoajassa
Parantunut työtyytyväisyys (virkailijat ym.)	Mielekkäämmät työtehtävät Parantunut riskien hallinta Parantunut turvallisuus	Muutokset työtyytyväisyydessä (kysely)
Parantunut päätöksenteko ja demokratia	Parempi sääntelyn ja vaikutusten yhteys Parempi kansalaisten osallistuminen ja läpinäkyvyys	Kansalaisosallistumisen määrä Muutokset julkishallinnon maineessa (kysely)

2 DATAN MUOTOVAATIMUKSET

JSON Schema on yleisesti käytetty työkalu JSON-datan muodon validointiin. Schema.json tiedostossa määritellään kaikki lomakkeen kenttien nimet ja minkä tyyppisiä arvoja voidaan odottaa. Voidaan myös määritellä yksittäisten arvojen tai kokoelmien pakollisuuksia, joita voidaan hyödyntää lomakkeen back-end validoinnissa, jotta keskeneräistä lomaketta ei olisi mahdollista lähettää. Esimerkki scheman sisällöstä löytyy kuvasta 1, jossa näkyy ensimmäiset 43 riviä.



```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "id": "esimerkki.v1",
  "description": "Esimerkki schemasta",
  "additionalProperties": false,
  "properties": {
    "VersioKoodi": {
      "description": "Moduulin versionumero",
      "type": "string",
      "enum": ["1"]
    },
    "EsimerkkiKytkin": {
      "description": "true/false",
      "type": "boolean",
      "enum": [false, true]
    },
    "EsimerkkiNumero": {
      "description": "Numero, eli voi olla int tai float",
      "type": "number"
    },
    "EsimerkkiTeksti": {
      "description": "Teksti",
      "type": "string"
    },
    "EsimerkkiTaulukko": {
      "type": "array",
      "description": "Kokoelma arvoja",
      "items": {
        "properties": {
          "Esimerkki2Teksti": {
            "description": "Teksti arvo",
            "type": "string"
          },
          "EsimerkkiPakollinen": {
            "description": "Pakollinen arvo",
            "type": "number"
          }
        },
        "required": ["EsimerkkiPakollinen"]
      }
    }
  },
  "required": ["VersioKoodi", "EsimerkkiKytkin"],
```

Kuva 1. JSON Schema -esimerkki.

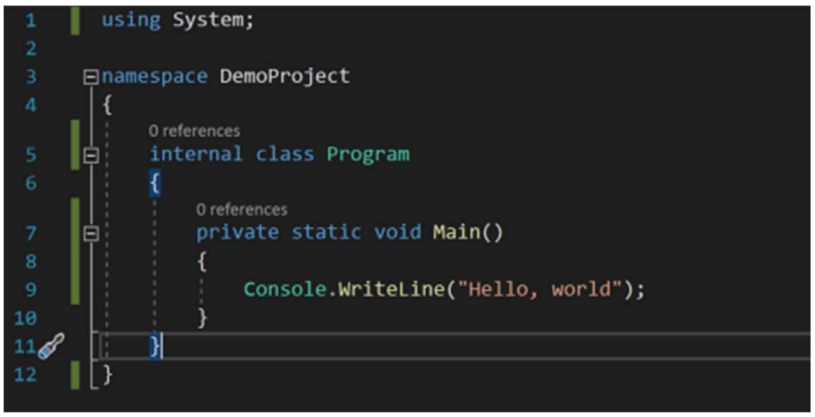
Lisätietoa ja esimerkkejä JSON Scheman käytöstä löytyy osoitteesta https://json-schema.org

3 KÄYTETTYJÄ TEKNOLOGIOITA

Käsittelyprosessi vaatii usein monenlaisia eri teknologioita, jotta saadaan toimiva kokonaisuus. Teknologiat kehittyvät huimaa vauhtia, josta syystä jokaisen eri teknologian kohdalla on myös linkki lisätietoihin, jotta ajantasainen tieto on helposti löydettävissä myös tämän raportin tekohetken jälkeen.

3.1 C# -ohjelmointikieli

Microsoftin kehittämä ja ylläpitämä objekti-orientoitunut ohjelmointikieli, jolla voidaan kehittää ohjelmia jotka suoritetaan .NET Frameworkin (/.NET Core) ympäristössä. Tällä kielellä voidaan tehdä laaja-alaisesti monenlaisia sovelluksia ajettavaksi joko lokaalisti käyttäjän tietokoneella tai serverillä web-sovelluksen back-endinä.



Kuva 2. C#-Koodi -esimerkki.

Kuvan koodiesimerkki alkaa using-direktiivillä, joka viittaa System nimiavaruuteen (namespace). Tämä System-nimiavaruus sisältää monia eri tyyppejä, kuten esimerkissä käytetty Console-luokka. Mikäli System nimiavaruuteen ei tehdä viittausta tällä tavalla, pitäisi Console.Writeline() kirjoittaa pidemmin System.Console.WriteLine().

Lisätietoa C#-ohjelmointikielestä ja sen ominaisuuksista löytyy osoitteesta https://docs.microsoft.com/en-us/dotnet/csharp/

3.1.1 Aspose.Words - API

API (Application Programming Interface), jonka avulla voidaan manipuloida Word-tiedostoja, sekä generoida PDF-tiedostoja. Tämä mahdollistaa Word-sapluunojen laatimisen käyttäen syntaksia, jota Aspose ymmärtää. Asposen toimintaan on monipuoliset mahdollisuudet vaikuttaa koodin kautta.

Yksittäinen arvo saadaan JSON:sta injektoitua Word -tiedostoon käyttämällä {{ Avain }} -syntaksia. Taulukon osalta esimerkkinä käytämme kuvan 3 JSON-dataa. Se sisältää kokoelman, jossa on kaksi objektia.

Kuva 3. Taulukon JSON-data.

Kun kyseessä on kuvan 3 kaltainen taulukko, jonka nimi on Taulukko, voidaan käyttää seuraavanlaista syntaksia:

Taulukko 2. Aspose-syntaksi esimerkki

{{#foreach Taulukko}}{{Avain1}}	{{Avain2}}	{{Avain3}}{{/foreach Taulukko}}

Yllä olevan taulukko 2 mukaisilla ohjeilla Aspose käy läpi kaikki Taulukko-kokoelman objektit ja asettaa arvot avainten paikoille. Lopputuloksena saataisiin seuraavanlainen taulukko:

Taulukko 3. Esimerkki taulukkoon asetetusta JSON-datasta.

1. objektin Avain1 selite	1. objektin Avain2 selite	1. objektin Avain3 selite
1. objektin Avain1 selite	2. objektin Avain2 selite	2. objektin Avain3 selite

Lisätietoa Aspose:n ominaisuuksista löytyy osoitteesta https://products.aspose.com/words

3.1.2 Quartz.NET -kirjasto

Quartz.Net on C#-ohjelmointikielellä kirjoitettu .NET kirjasto, joka perustuu Javalla toteutettuun Quartz-aikataulutus frameworkiin.

Tällä kirjastolla voidaan aikatauluttaa toimintoja siten, että jokin toiminto laukaistaan esimerkiksi minuutin välein.

Lisätietoa ja esimerkkejä löytyy osoitteesta https://www.quartz-scheduler.net/

3.2 JavaScript -ohjelmointikieli

JavaScript (JS) on kevyt, ajossa koottu (Just-In-Time, JIT) ohjelmointikieli. Vaikka se tunnetaan parhaiten verkkosivujen skriptikielenä, myös muut kuin selainympäristöt, kuten Node.js. JavaScript tukee oliokeskeisiä, pakollisia ja deklaratiivisia (esim. Funktionaalinen ohjelmointi) tyylejä. (Mozilla, 2020)

JavaScriptin päälle on rakennettu monenlaisia JS-kirjastoja ja frameworkeja helpottamaan kehitystyötä, joista tässä projektissa on käytössä ReactJS.

Selainten ymmärtämä JavaScript tunnetaan nimellä ES5, joka tunnetaan myös nimillä ECMAScript 5 ja ECMAScript 2009. Se on ECMA-organisaation laatima standardi JavaScriptille. Käytännössä kaikki eniten käytetyt selaimet tänä päivänä osaavat kääntää ES5. (W3Schools, 2020)

3.2.1 ReactJS JS-kirjasto

Facebookin kehittämä ja ylläpitämä JavaScript-kirjasto, joka on suunniteltu käyttöliittymien toteuttamiseen. Se on sekoitus JavaScriptiä ja HTML:n (Hypertext Markup Language) näköistä syntaksia. Todellisuudessa nämä HTML:n näköiset asiat ovat täysin JavaScriptiä. Esimerkiksi kuvassa 4 renderöidään ruudulle yksi div, jonka sisällä on h1-otsikko. Tässä kirjoitetussa muodossa koodi on helposti luettavaa. Kyseistä syntaksia kutsutaan nimellä JSX. ReactJS tunnetaan yleisestä nimellä React.

```
import React, { Component } from "react";

class App extends Component {
  render() {
    return (
      <div>
        <h1>Hey, I'm Panu</h1>
      </div>
    );
  }
}
```

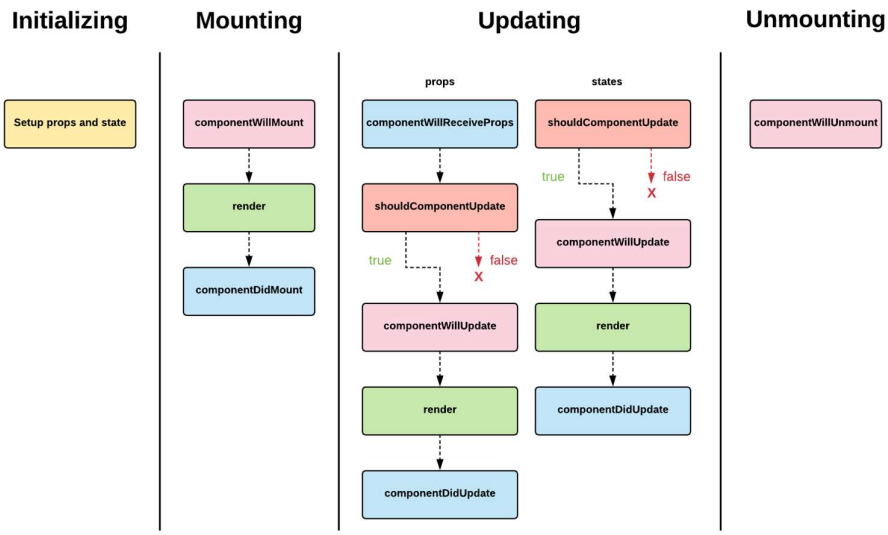
Kuva 4. ReactJS-esimerkki 1/2.

Aiemman esimerkin (Kuva 4) JSX-koodi käännetään taustalla kuvan 5 esittämään muotoon, jonka selain aiemmin mainitusti ymmärtää.

```
import React, { Component } from "react";

class App extends Component {
  render() {
    return React.createElement(
      "div",
      null,
      React.createElement("h1", null, "Hey, I'm Panu")
    );
  }
}
```

Kuva 5. ReactJS-esimerkki 2/2.

Tätä työtä tehdessä käytössä oli 15.6.2 versio ReactJS-kirjastosta, jonka osalta elinkaaret (lifecycle) toimivat kuvan 6 esittämällä tavalla.



Kuva 6. React elinkaari.

Reactin kanssa tutuksi tulee niin sanottu ”state”, joka on React-komponenttiin sisään rakennettu objekti, johon säilötään arvoja, joita komponentti käyttää. Kun tämä objekti muuttuu, komponentti renderöi itsensä uudelleen selaimen näkymässä.

Initializing: Hetki, jolloin sovelluksen tila alustetaan

Mounting: componentWillMount() ajetaan juuri ennen kuin näkymä renderöidään selaimeen näkyviin. Usein tämän metodin sisällä ajetaan esimerkiksi hakuja tietokannasta webbisivulla näytettävää sisältöä varten.

Updating: componentWillReceiveProps() käytetään silloin kun tiedetään että sovelluksen näkymään tulee uutta dataa silloin kun näkymä on jo ladattu. Tieto eri komponenttien välillä välitetään niin sanottuina propseina. Nämä propsit voivat sisältää käytännössä mitä tahansa yksinkertaisista numero- tai tekstiarvoista funktioihin. Voidaan myös ajaa shouldComponentUpdate() kun näkymä päivittyy.

Unmounting: componentWillUnmount() ajetaan ennen kun komponentti poistuu selaimen näkymästä.

Ajantasalla olevaa tietoa ReactJS:n uusista ominaisuuksista löytyy osoitteesta https://reactjs.org

3.2.2 ReduxJS JS-kirjasto

ReduxJS on JavaScript-sovelluksen web-selaimen tilanhallintaan suunniteltu kirjasto, jonka avulla on mahdollista hallita näitä tiloja tehokkaammin. Tämä eroaa käytetystä ReactJS-version sisään rakennetusta tilanhallinasta siten, että ReduxJS vie halutut arvot globaalille tasolle, josta niitä voidaan käyttää missä tahansa komponentissa, joka on liitetty Reduxiin. Tämän kirjaston käyttö ei ole missään nimessä rajoitettu vain Reactin kanssa käytettäväksi. Myös ReduxJS sisältää omanlaisensa elinkaaren liittyen tilaan, johon kuuluu seuraavia käsitteitä:

State (Tila): tarkoittaa Redux API:ssa yksittäistä tilan arvoa, joka on varaston hallinnoima ja getState():n palauttama. Se sisältää Redux sovelluksen koko tilan, joka usein sisältää paljon sisäkkäisiä objekteja.

UI: käyttöliittymä, josta muutokset tilaan laukaistaan joko käyttäjän tai front-end koodin toimesta.

Actions (Toimenpiteet): Action on yksinkertainen objekti, joka edustaa aikomusta tehdä muutos tilaan. Actionilla täytyy olla tyyppi, jonka suositellaan olevan string-tyyppisiä, koska ne ovat serialisoitavissa

Reducer (Supistaja): Funktio, joka hyväksyy akkumuloidun arvon, joka Reduxin tapauksessa on tila-objekti, sekä akkumuloitavat arvot jotka ovat actioneita. Reducerit laskevat uuden tilan aiemman tilan ja actionin pohjalta.

Store (Varasto): Objekti, joka sisältää koko tilapuun

Lisätietoa Reduxista ja sen ominaisuuksista löytyy osoitteesta https://redux.js.org

3.2.3 RamdaJS JS-Kirjasto

JavaScript-kirjasto, jonka olemassaolon tarkoituksena on helpottaa JavaScript objektien ja kokoelmien käsittelyä. Tässä projektissa tämä teknologia on käytössä luonnollisesti käyttöliittymän puolella, kun halutaan datan vastaanoton yhteydessä prosessoida JSON-data sellaiseen muotoon mitä on helpompi käyttää esimerkiksi käyttöliittymän valintalista-komponenteissa, jotka usein vaativat eri valinnat tietynmuotoisena kokoelmana. Tästä esimerkkinä yksinkertainen objektikokoelman käsittely. Kuvitellaan että vastaanotettu data on kuvan 7 esittämässä muodossa, eli kokoelma objekteja, jotka sisältävät kaksi avain-arvo paria.

```
const data = [
  { code: '1', description: 'selite 1' },
  { code: '2', description: 'selite 2' }
];
```

Kuva 7. Esimerkki objektikokoelmasta.

Kuvan 7 dataa käyttäen, lopputulokseksi halutaan kokoelma, jossa arvot säilyvät, mutta avaimet ovat eri. Esimerkin kokoelma voidaan prosessoida kuvan 8 esittämällä tavalla:

```
const parseOptions = data => {
  R.map(o => ({
    value: R.pathOr(null, ['code'], o),
    label: R.pathOr(null, ['description'], o)
  }),
    data
  )
};
```

Kuva 8. RamdaJS-esimerkki.

R.map(): funktio, joka palauttaa kokoelman.

value & **label** määrittelevät objektin uudet avaimet.

R.pathOr(): funktio, joka palauttaa avaimen arvon jos sillaista on. Muussa tapauksessa se palauttaa tyhjäarvon (null).

Lopputuloksena saadaan kuvan 9 mallinen kokoelma, joka sisältää alkuperäiset arvot eri avaimilla.

```
[
  { value: '1', label: 'selite 1' },
  { value: '2', label: 'selite 2' }
]
```

Kuva 9. Kokoelma prosessoinnin jälkeen.

RamdaJS:n käyttömahdollisuudet ovat todella laajat. Tämä on vain pintaraapaisu siitä, mihin sitä voidaan käyttää.

Mikäli RamdaJS ei ole vielä tuttu, suosittelen tutustumaan siihen osoitteessa https://ramdajs.com.

3.3 RabbitMQ -viestinvälittäjä

RabbitMQ on avoimen lähdekoodin viestinvälitysohjelma. Tämä mahdollistaa eri järjestelmien välillä kommunikaation ja esimerkiksi JSON datan lähettämisen ja vastaanottamisen järjestelmästä toiseen.

Toiminta perustuu kuvan 10 esittämällä tavalla siihen, että on viestin lähettävä taho (producer), broker (itse RabbitMQ, joka välittää viestejä), sekä vastaanottava taho (consumer).

Kuva 10. RabbitMQ-esimerkki

RabbitMQ:ta ajetaan serverillä ja se asettaa vastaanottamansa viestit jonoon jolle on määritelty nimi, esimerkiksi ”testi_jono”. Vastaanottava taho, eli consumer, kuuntelee tätä jonoa. Kun viesti on vastaanotettu, brokerille lähetetään kuittaus siitä, että viesti on saapunut perille, mikäli näin on määritelty.

Tarkempaa lisätietoa RabbitMQ:sta löytyy osoitteesta https://www.rabbitmq.com

3.4 **JSON -formaatti**

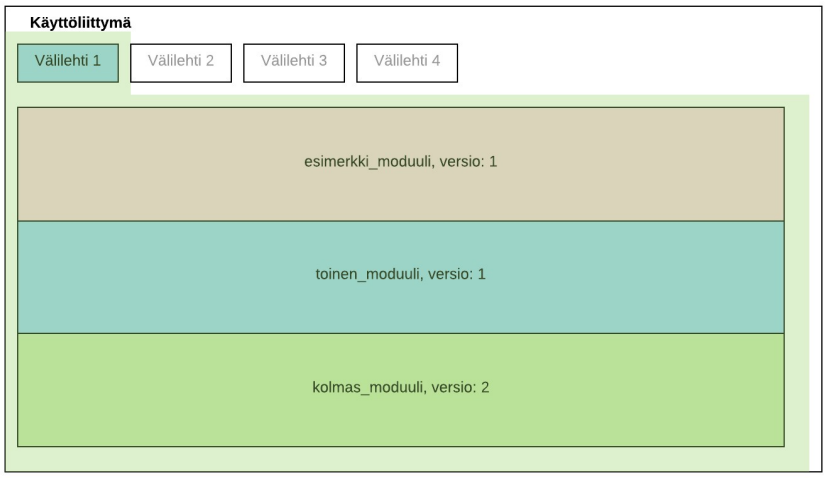
JSON (JavaScript Object Notation) on kevyt tiedon välittämisen formaatti, joka on kehittäjälle helposti luettava ja kirjoitettava. Tämä formaatti koostuu avain (key) / arvo (value) pareista. Kuvassa 11 on esimerkki rakenteesta ja muutamasta eri data-tyypistä.

Kuva 11. Esimerkki JSON:sta

Lisätietoa JSON:sta löytyy osoitteesta https://www.json.org/

4 **PROSESSIN NÄKYMÄT**

Iso osa tätä projektia oli käyttöliittymien laatiminen käsiteltäville lomakkeille asiointialustalle, sekä samojen lomakemoduulien siirtäminen YLVAan. Toteutus on tehty siten, että taustapalvelussa määritellään jokaista sivun välilehteä kohden moduulit, mitä halutaan näyttää ottaen huomioon myös moduulin version. Asioinnin aloitushetkellä käytössä olleet moduulien versiot säilyvät asioinnissa muuttumattomina asioinnin loppuun asti. Iso osa välilehtien näkymistä koostuu useammasta moduulista kuten on esitetty kuvassa 12.



Kuva 12. Useampi eri moduuli samassa näkymässä

Näkymien toteutus moduuleina mahdollistaa samojen moduulien käytön useammassa eri lomakkeessa, jos on tarve kerätä samoja tietoja.

4.1 **Asiointialusta**

Prosessi alkaa asiointialustalta, jossa asiakas aloittaa uuden asioinnin ja täyttää ja lähettää lomakkeen. Lomakkeilla on käytössä monelta osin reaaliaikainen validointi, jolla varmistetaan, että tiedot täytetään oikeassa muodossa. Jos pakollisia kenttiä jää täyttämättä, niistä tulee virheilmoitus yritettäessä lähettää lomaketta. Mikäli kaikki tarvittavat tiedot on täytetty, lähetys onnistuu ja selaimen näkymä palaa asiointinäkymään, josta ilmenee lomakkeen käsittelyn tila. Lähetys käsittelyjärjestelmään tehdään käyttäen RabbitMQ:ta.

4.2 **Käsittely**

Käsittely kattaa tässä tapauksessa kokonaisuudessaan Käsittelyjärjestelmässä tapahtuvat toiminnot, mukaan lukien itse käsittelijän tekemät toimenpiteet käyttöliittymällä. Käsittelijä on se joka tarkastaa vastaanotetut tiedot ja päättää niiden pohjalta jatkotoimenpiteet. Tällä päätöksellä on suuri vaikutus päätöksen viimeistelyn jälkeen tapahtuviin toimintoihin.

4.2.1 Vastaanotto

Käsittelyjärjestelmä kuuntelee RabbitMQ-jonoa ajastetusti Quartz.NETia käyttäen. Kun käsittelyjärjestelmä saa viestin, että uusi lomake on lähetetty asiointialustalta, alkaa ensimmäinen automatisoitu prosessi, joka muuttaa asiointialustan asioinnin tilan tilaan ”Käsittelyssä” hyödyntäen RabbitMQ:ta, parsii lomakkeen datan tietokantaan uudelle arkistoriville ja päivittää metatietoja, jos niitä on saatavilla (esimerkiksi diaarinumero).

4.2.2 Metadatan parsinta ja päivitys

Metadatan päivityksessä haetaan lomakkeen kuvan 13 esittämällä tavalla JSON:sta tietoja, jotka päivitetään tietokantatauluun työjonoon lisäämisen yhteydessä. Näitä voivat olla esimerkiksi profiloinnissa määritelty kohdetunnus, yrityksen nimi ja sijaintikunta. Näiden tietojen kaivamiseen hyödynnetään Newtonsoft.Json.Linq-kirjastoa JSON-datan deserialisoimiseen ja haluttujen arvojen etsimiseen.

```
1 reference | Panu Siik, 50 days ago | 1 author, 3 changes | 11 work items
private ArkistoSPAv2 UpdateMetadataPimaTutkimusraportti(ArkistoSPAv2 arkisto)
{
    var tutkimusraportti = JObject.Parse(arkisto.FormData);

    // Poimitaan mahdollinen kohde mihin asiointi kohdistuu
    var profilingTargetIdToken = tutkimusraportti.SelectToken("$.ylva_pima_ilmoituksentekija.Profilointi.KohdeTunnus");
    var targetIdToken = tutkimusraportti.SelectToken(".ylva_pima_alueentiedot_tunnistetiedot.KohdeIdTunnus");

    if (profilingTargetIdToken != null)
    {
        arkisto.Target_ID = (int)profilingTargetIdToken;
    }
    else if (targetIdToken != null)
    {
        arkisto.Target_ID = (int)targetIdToken;
    }

    // Toiminnan harjoittajan yritys
    var customerToken = tutkimusraportti.SelectToken("$.ylva_pima_ilmoituksentekija.YritysNimi");
    if (customerToken != null)
    {
        arkisto.Customer = (string)customerToken;
    }

    // Formconfig
    arkisto.FormConfig = YlvaLomakeApuri.CreatePollutedSoilResearchReportFormConfig(arkisto.FormData).ToString();

    // Vastaava ELY-keskus organisaatio tieto
    using (var koodistoService = _uow.KoodistoService())
    {
        var kotiKuntaNumero = (int)tutkimusraportti.SelectToken("$.ylva_pima_alueentiedot_osoitetiedot.SijaintiKuntaKoodi");
        var kunta = _uow.Cache().Kunnat.Single(k => k.Kunta_Id == kotiKuntaNumero);
        // Kunnan ympäristöasioista vastaavan elyn tunniste
        arkisto.Organisaatio_ID = kunta.YmpVastuuEly_Id;
    }

    return ArkistoSPAv2_Update(arkisto);
}
```

Kuva 13. Esimerkki metadatan parsinnasta

Kuten esimerkistä käy ilmi, tietoja yritetään etsiä lomakkeen datasta hyödyntäen SelectTokenia, joka on JObjectin metodi.

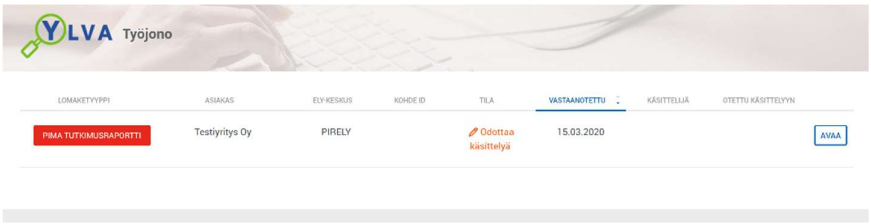
Käytetystä syntaksista löytyy lisätietoa esimerkiksi osoitteesta https://jsonpath.com/

Tässä vaiheessa myös luodaan lomakkeelle ”FormConfig” jota käytetään lomakkeen näyttämiseen selaimessa. Tässä konfiguraatiossa määritellään muun muassa se mitkä välilehdet ovat aktiivisia missäkin tilassa ja mitä moduuliversioita käytetään.

Parsinnan lopuksi arkistorivi päivitetään uusilla tiedoilla.

4.2.3 Työjono

Työjono on kuvan 14 mukainen näkymä, johon listataan saapuneet lomakkeet. Jokainen rivi sisältää lomakkeeseen liittyviä perustietoja, kuten lomakkeen tyyppi, asiakas, ELY-keskus, kohdetunniste, tila ja saapumispäivämäärä. Riville päivittyy myös käsittelijän nimi ja käsittelyyn oton päivämäärä käsittelyyn oton yhteydessä.

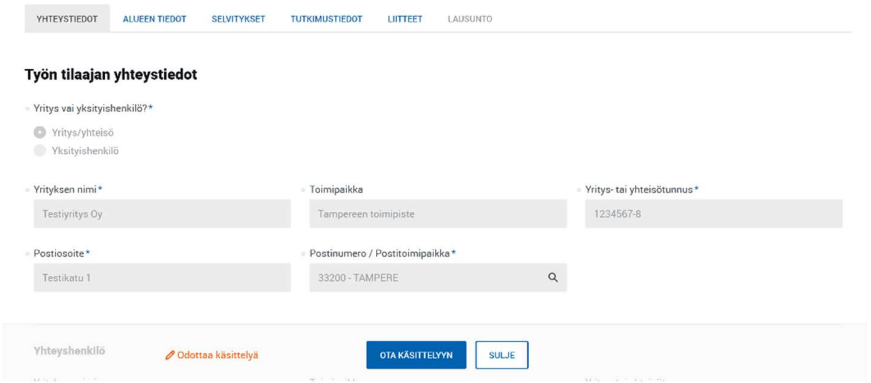


Kuva 14. Työjono näkymä

Lomake ilmestyy käsittelyjärjestelmän työjonoon, joka näkyy kaikkien ELY-keskuksien kaikille vastuuvalvojille. Tästä näkymästä valvojalla on mahdollisuus avata lomake ja tarkastella tietoja ennen käsittelyyn ottoa.

4.2.4 Käsittelynäkymä

Lomakkeen ulkoasu on sama kuin asiointialustalla, mutta lisäksi on lomakkeen mukaan esimerkiksi hyväksyntään ja/tai hylkäämiseen liittyvät välilehdet, jotka ovat estettynä ennen kuin päätös on tehty. Tässä tapauksessa kuvassa 15 näkyvä ”Lausunto”-välilehti on deaktivoitu, kunnes on tehty jokin ratkaisu. Tässä tapauksessa hylkäys ei ole optiona, vaan ilmoitusta täydennetään asiointialustalle lähetettävillä täydennyspyynnöillä niin pitkään, että tiedot ovat riittävät hyväksyntään.



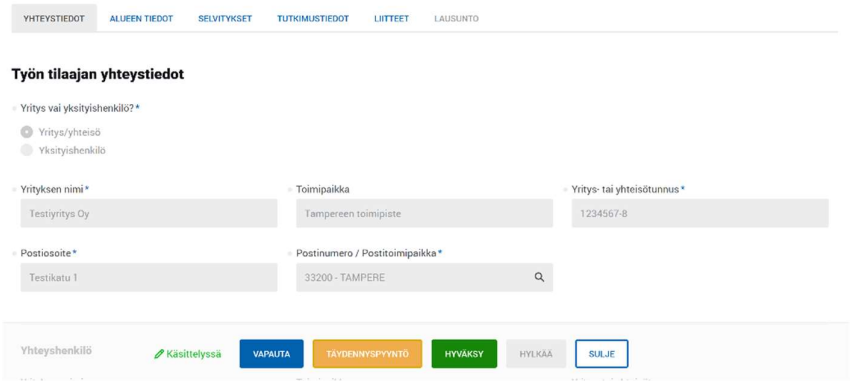
Kuva 15. Avattu lomake

Kuvan 15 esittämän verkkosivun alareunassa on käsittelyyn ottoa varten ”Ota käsittelyyn” painike, jota napsauttamalla lomakkeen käsittely aloitetaan. Samaan aikaan tapahtuu seuraavat asiat:

1. Lomakkeen käsittelijäksi asetetaan sillä hetkellä kirjautuneena oleva vastuuvalvoja.
2. Lomakkeelta saatujen tietojen pohjalta perustetaan uusi kohde, mikäli kyseessä on uusi ilmoitus tai hakemus, joka ei liity mihinkään olemassa olevaan kohteeseen. Samalla päivitetään kohteen ja kohteen osioiden osalta

koordinaattitiedot, osoitetiedot sekä mm. yhteyshenkilöiden tiedot.

3. USPAan luodaan tarvittaessa uusi asia ja toimenpide ja alkuperäinen ilmoitus- tai hakemusdokumentti lisätään toimenpiteeksi.
4. Arkistorivi päivitetään tilaan ”Käsittelyssä”
5. Sivun alareunan painikkeet päivittyvät siten, että tilalle tulee itse päätöksen tekoon liittyvät näppäimet: Vapauta, Täydennyspyyntö, Hyväksy, Hylkää, sekä Sulje. Nämä ovat esitetty kuvassa 16.



Kuva 16. Päätösnäkymä

Näillä kuvan 16 näkymän näppäimillä on kaikilla omat toimintonsa, jotka mahdollistavat käsittelyn. Selkeyden vuoksi nämä toiminnot ovat kuvattuna seuraavissa kappaleissa järjestyksessä vasemmalta oikealle.

Vapauta

Jos tulee tilanne, että alkuperäinen käsittelijä haluaa vapauttaa lomakkeen muiden käsiteltäväksi, se voidaan vapauttaa. Tällöin arkistorivin tila päivitetään takaisin ”Odottaa käsittelyä”-tilaan.

Täydennyspyyntö

Jos lomakkeessa havaitaan puutteita, vastuuvalvoja voi tehdä täydennyspyynnön, joka välitetään asiointipalveluun. Tämä muuttaa arkistorivin tilaan ”Odottaa täydennystä”, jolloin lomakkeen käsittely on lukossa siihen asti, että täydennetty lomake saadaan asiakkaalta. Täydennyspyynnön mukana viestissä voidaan tarkentaa mitä tietoja asiakkaan täytyy täydentää.

Hyväksy

Mikäli lomakkeen tiedot ovat kunnossa, eikä ole tarvetta täydennykselle, käsittelijä voi hyväksyä lomakkeen viimeisteltäväksi. Arkistorivin tila muutetaan tilaan ”Ilmoitus Hyväksytty” jonka myötä ”Hyväksyntä”

välilehti aktivoituu. Tällä avautuneella välilehdellä käsittelyprosessi saatetaan loppuun viimeistelemällä se.

Hylkää

Hylkäys on harvinaista, mutta mikäli täydennyspyynnönkään jälkeen ilmoituksen tai hakemuksen tiedot eivät riitä hyväksyntään, se voidaan hylätä. Tällöin ”Hylkäys”-välilehti aktivoituu, missä vastuuvalvoja voi kirjoittaa perustelun hylkäävälle päätökselle. Tässä esimerkissä hylkäys ei ole vaihtoehtona, vaan lomaketta täydennetään tarvittaessa täydennyspyynnöillä siihen pisteeseen asti, että lomake voidaan käsitellä.

Sulje

Mikäli käsittelyä ei jostain syystä haluta tehdä heti loppuu, se voidaan sulkea, jonka jälkeen käsittelyä voidaan jatkaa myöhemmin. Tämä valinta ei muuta käsittelijää.

4.2.5 Päätöksen viimeistely

Päätöksen viimeistelyyn PIMA-tutkimusraportin osalta on yksinkertainen käyttöliittymä, joka sisältää valinnan sille, onko puhdistamistarvetta vai ei. Mikäli katsotaan että puhdistustarvetta on, asioinnin seuraava vaihe on ilmoituksen tekeminen. Mikäli puhdistustarvetta ei ole, asiointi päättyy, kun raportti on käsitelty.

Lausunto-näkymässä on lisäksi myös tekstikenttä, johon käsittelijä voi kirjoittaa vapaamuotoisesti lisätietoja liittyen ratkaisuunsa. Tällä välilehdellä voidaan myös valita USPAa varten käsittelijät ja roolit.

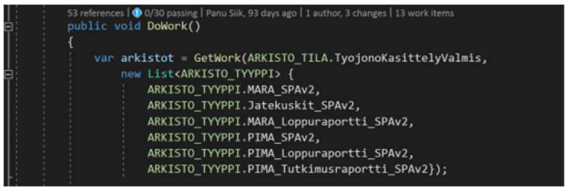
Välilehden alareunassa on ”esikatselu”-valinta, jolla on mahdollisuus esikatsella dokumentti, sekä ”viimeistele”-valinta, joka viimeistelee prosessin käyttöliittymän osalta ja automaattiset taustaprosessit tekevät käsittelyprosessin USPAssa tehtäviä toimenpiteitä lukuun ottamatta loppuun.

5 AUTOMATISOIDUT PROSESSIT

Työjonossa käsitellyt hakemukset, ilmoitukset tai raportit kulkevat saman prosessiputken läpi. Haasteena tässä oli ottaa huomioon jokaisen eri lomakekokonaisuuden tarpeet käsittelyn osalta. Seuraavissa kappaleissa keskitytään PIMA-arkistotyyppiin ja sen reittiin prosessin läpi.

5.1 **Vaihe 1 (Dokumenttien luonti)**

Kun päätös on viimeistelty käsittelynäkymässä, arkistorivin tila siirtyy tilaan ”Käsittely valmis”. Työjonon ensimmäinen worker löytää tämän suodattamalla arkistotaulusta kaikki ne rivit joiden tila on haluttu.



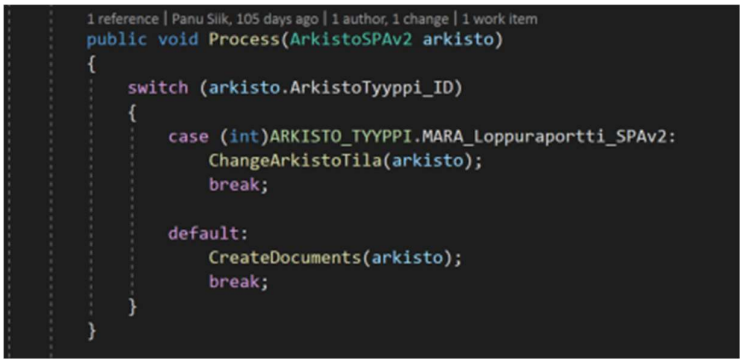
Kuva 17. Arkistojen haku.

Työläisluokkien jakama GetWork-metodi ottaa vastaan haettavan tilan, sekä listan eri arkistotyypeistä, joita hakutuloksen halutaan sisältävän. Samankaltainen haku tehdään kaikissa myöhemmissä vaiheissa, erona haettu tila.

Kuva 18. Try/Catch.

Hakutuloksen jokainen arkisto yritetään prosessoida. Mikäli tapahtuu virhe, kirjataan virhe arkistoriville, sekä virheviesti prosessointipalvelun konsoliin arkistorivin tunnisteen kera.

Arkistot voidaan prosessoida kahdella eri tavalla, joka vaatii joko if/else-, tai kuvan 19 kaltaisen switch-case -rakenteen.



Kuva 19. Switch-Case.

Kuvan 19 ensimmäinen case viittaa arkistotyyppiin, josta ei luoda mitään palautettavaa dokumenttia, vaan myöhemmässä vaiheessa palautetaan

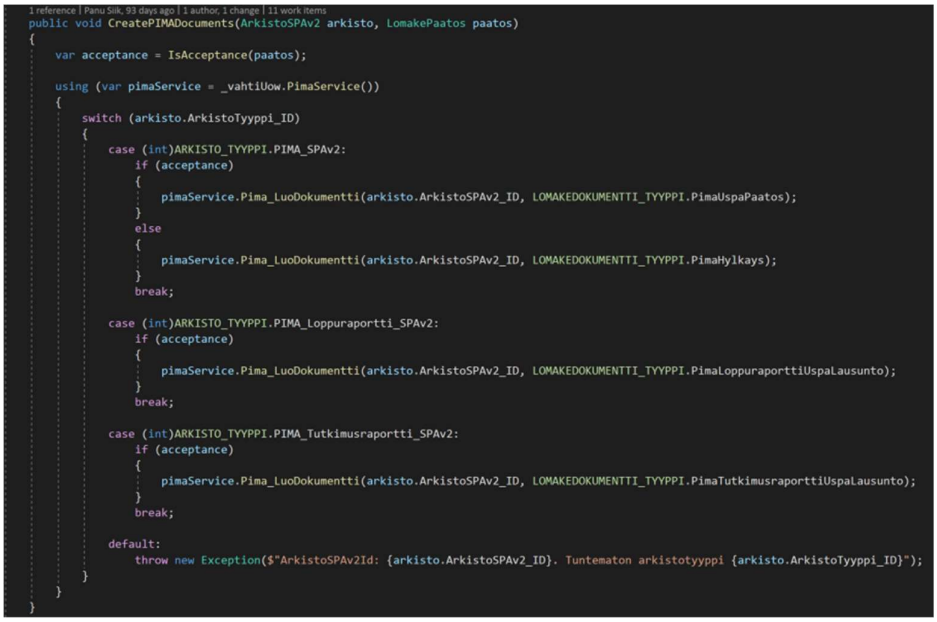
sama loppuraportti dokumentti, johon on lisätty sähköiseen hyväksyntään liittyvät tekstit.

1. Jokaisella arkistotyypillä on kuvan 20 esittämällä tavalla omat metodinsa dokumentinluonnille.

```
1 reference | Panu Silk, 93 days ago | 1 author, 4 changes | 12 work items
public void CreateDocuments(ArkistoSPAv2 arkisto)
{
    using (var lomakeService = _vahtiUow.LomakeService())
    using (var raportointiDataService = _vahtiUow.RaportointiDataService())
    using (var raportointiDataServiceTransaction = raportointiDataService.BeginTransaction())
    using (var lomakeServiceTransaction = lomakeService.BeginTransaction())
    {
        try
        {
            // Haetaan päätös
            var paatos = lomakeService.LomakePaatos_Search(
                new Data.Filters.LomakePaatos_Filter
                {
                    ArkistoSPAv2_ID = new List<int> { arkisto.ArkistoSPAv2_ID },
                    Includes = Data.Filters.LomakePaatosIncludes.ArkistoSPAv2
                }).Single();

            switch (paatos.ArkistoSPAv2.ArkistoTyyppi_ID)
            {
                case (int)ARKISTO_TYYPPI.Jatekuskit_SPAv2:
                    CreateJHRDocuments(arkisto, paatos);
                    break;

                case (int)ARKISTO_TYYPPI.MARA_SPAv2:
                    CreateMARADocuments(arkisto, paatos);
                    break;

                case (int)ARKISTO_TYYPPI.PIMA_SPAv2:
                case (int)ARKISTO_TYYPPI.PIMA_Loppuraportti_SPAv2:
                case (int)ARKISTO_TYYPPI.PIMA_Tutkimusraportti_SPAv2:
                    CreatePIMADocuments(arkisto, paatos);
                    break;

                default:
                    throw new Exception($"Tuntematon ArkistoTyyppi: {paatos.ArkistoSPAv2.ArkistoTyyppi_ID}");
            }

            // Päivitetään arkistorivi
            arkisto.Set_ArkistoTila(ARKISTO_TILA.TyojonoDokumentitLuotu);
            raportointiDataService.ArkistoSPAv2_Update(arkisto);

            // Kommitoidaan
            lomakeServiceTransaction.Commit();
            raportointiDataServiceTransaction.Commit();

            _logger.Info($"Dokumentit luotu: 'KasittelyValmis' -> 'DokumentitLuotu' ArkistoSPAv2_ID: {arkisto.ArkistoSPAv2_ID}");
        }
        catch (Exception)
        {
            lomakeServiceTransaction.Rollback();
            raportointiDataServiceTransaction.Rollback();
            throw;
        }
    }
}
```

Kuva 20. Dokumenttien luonti.

PIMA:n tapauksessa arkistotyyppejä on kolme, joista jokainen käyttää samaa CreatePIMADocuments-metodia, johon täytyy välittää arkisto, sekä arkistoon liittyvä päätös, jotta saadaan haettua dokumenttia varten kaikki tarvittavat tiedot.



```
1 reference | Panu Siik, 93 days ago | 1 author, 1 change | 11 work items
public void CreatePIMADocuments(ArkistoSPAv2 arkisto, LomakePaatos paatos)
{
    var acceptance = IsAcceptance(paatos);

    using (var pimaService = _vahtiUow.PimaService())
    {
        switch (arkisto.ArkistoTyyppi_ID)
        {
            case (int)ARKISTO_TYYPPI.PIMA_SPAv2:
                if (acceptance)
                {
                    pimaService.Pima_LuoDokumentti(arkisto.ArkistoSPAv2_ID, LOMAKEDOKUMENTTI_TYYPPI.PimaUspaPaatos);
                }
                else
                {
                    pimaService.Pima_LuoDokumentti(arkisto.ArkistoSPAv2_ID, LOMAKEDOKUMENTTI_TYYPPI.PimaHylkays);
                }
                break;

            case (int)ARKISTO_TYYPPI.PIMA_Loppuraportti_SPAv2:
                if (acceptance)
                {
                    pimaService.Pima_LuoDokumentti(arkisto.ArkistoSPAv2_ID, LOMAKEDOKUMENTTI_TYYPPI.PimaLoppuraporttiUspaLausunto);
                }
                break;

            case (int)ARKISTO_TYYPPI.PIMA_Tutkimusraportti_SPAv2:
                if (acceptance)
                {
                    pimaService.Pima_LuoDokumentti(arkisto.ArkistoSPAv2_ID, LOMAKEDOKUMENTTI_TYYPPI.PimaTutkimusraporttiUspaLausunto);
                }
                break;

            default:
                throw new Exception($"ArkistoSPAv2Id: {arkisto.ArkistoSPAv2_ID}. Tuntematon arkistotyyppi {arkisto.ArkistoTyyppi_ID}");
        }
    }
}
```

Kuva 21. PIMA-dokumenttien luonti.

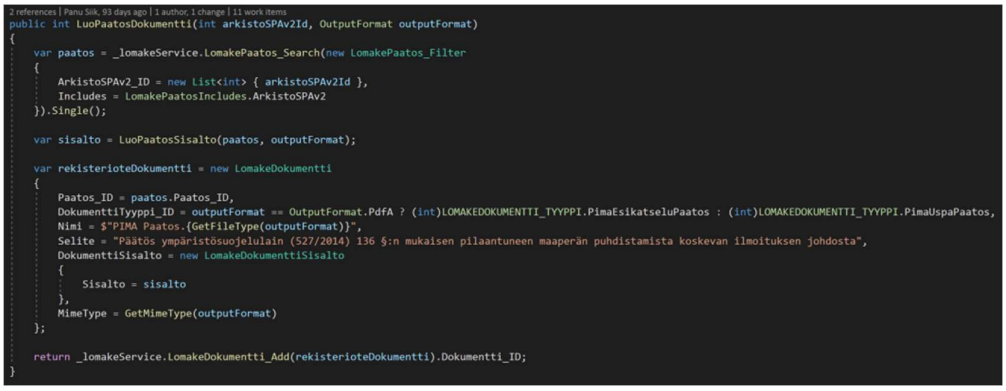
Metodin alussa oleva IsAcceptance palauttaa arvon true tai false, riippuen siitä sisältääkö päätöksen JSON-data hyväksyntään liittyvän lomakemoduulin. Mikäli ei sisällä, kyseessä on hylkäys. Tämä on luotettava tapa, koska päätöksen JSON-data ei sisällä hyväksyntään liittyvää moduulia jos kyseessä on hylkäys.

Pima_LuoDokumentti ottaa vastaan arkistorivin tunnisteen, sekä Lomakedokumentti tyypin. Jokaisesta dokumentista on mahdollista luoda tarpeen mukaan joko Word- tai PDFa -dokumentti.

```
9 references | Panu Siik, 93 days ago | 1 author, 1 change | 11 work items
public int Pima_LuoDokumentti(int arkistoSPAv2Id, LOMAKEDOKUMENTTI_TYYPPI tyyppi)
{
    switch (tyyppi)
    {
        case LOMAKEDOKUMENTTI_TYYPPI.PimaEsikatseluPaatos:
            return LuoPaatosDokumentti(arkistoSPAv2Id, OutputFormat.PdfA);

        case LOMAKEDOKUMENTTI_TYYPPI.PimaUspaPaatos:
            return LuoPaatosDokumentti(arkistoSPAv2Id, OutputFormat.Preserve);

        case LOMAKEDOKUMENTTI_TYYPPI.PimaLoppuraporttiEsikatseluLausunto:
            return LuoLoppuraporttiLausuntoDokumentti(arkistoSPAv2Id, OutputFormat.PdfA);

        case LOMAKEDOKUMENTTI_TYYPPI.PimaLoppuraporttiUspaLausunto:
            return LuoLoppuraporttiLausuntoDokumentti(arkistoSPAv2Id, OutputFormat.Preserve);

        case LOMAKEDOKUMENTTI_TYYPPI.PimaTutkimusraporttiEsikatseluLausunto:
            return LuoTutkimusraporttiLausuntoDokumentti(arkistoSPAv2Id, OutputFormat.PdfA);

        case LOMAKEDOKUMENTTI_TYYPPI.PimaTutkimusraporttiUspaLausunto:
            return LuoTutkimusraporttiLausuntoDokumentti(arkistoSPAv2Id, OutputFormat.Preserve);

        case LOMAKEDOKUMENTTI_TYYPPI.PimaHylkays:
            return LuoHylkaysDokumentti(arkistoSPAv2Id, OutputFormat.PdfA);

        default:
            throw new ArgumentOutOfRangeException(nameof(tyyppi), tyyppi, null);
    }
}
```

Kuva 22. Erilaisten dokumenttien luonti.

LuoPaatosDokumentti hakee yksittäisen päätöksen käyttäen sille välitettyä arkistorivin tunnistetta. Mikäli päätöksiä löytyy arkistoriville useita, tapahtuu poikkeus, joka estää prosessin etenemisen. (.Single())



```
2 references | Panu Sik, 93 days ago | 1 author, 1 change | 11 work items
public int LuoPaatosDokumentti(int arkistoSPAv2Id, OutputFormat outputFormat)
{
    var paatos = _lomakeService.LomakePaatos_Search(new LomakePaatos_Filter
    {
        ArkistoSPAv2_ID = new List<int> { arkistoSPAv2Id },
        Includes = LomakePaatosIncludes.ArkistoSPAv2
    }).Single();

    var sisalto = LuoPaatosSisalto(paatos, outputFormat);

    var rekisterioteDokumentti = new LomakeDokumentti
    {
        Paatos_ID = paatos.Paatos_ID,
        DokumenttiTyyppi_ID = outputFormat == OutputFormat.PdfA ? (int)LOMAKEDOKUMENTTI_TYYPPI.PimaEsikatseluPaatos : (int)LOMAKEDOKUMENTTI_TYYPPI.PimaUspaPaatos,
        Nimi = $"PIMA Paatos.{GetFileType(outputFormat)}",
        Selite = "Päätös ympäristösuojelulain (527/2014) 136 §:n mukaisen pilaantuneen maaperän puhdistamista koskevan ilmoituksen johdosta",
        DokumenttiSisalto = new LomakeDokumenttiSisalto
        {
            Sisalto = sisalto
        },
        MimeType = GetMimeType(outputFormat)
    };

    return _lomakeService.LomakeDokumentti_Add(rekisterioteDokumentti).Dokumentti_ID;
}
```

Kuva 23. Dokumentin luonti.

Aiemmassa esimerkkikuvassa 23 asetetaan Lomake dokumentille tarvittavat tiedot ja sisältö:

- Paatos_ID: Kyseisen dokumentin tunniste tietokannassa,
- DokumenttiTyyppi_ID: Tiedoston formaatti,
- Nimi: Tiedoston nimi,
- Sisältö: tiedoston sisältö muodossa byte[]
- MimeType: Standardin mukainen tyyppi määrittely muotoa ”tyyppi/alatyyppi” esimerkiksi ”application/pdf”

Dokumentin sisältö rakennetaan erillisessä metodissa, josta on esimerkki seuraavassa kuvassa.

```
1 reference | Panu Sik, 81 days ago | 1 author, 2 changes | 11 work items
private byte[] LuoPaatosSisalto(LomakePaatos paatos, OutputFormat asposOutputFormat)
{
    var data = new JObject();
    var template = Properties.Resources.pima_paatos_template_fi;

    var formData = JObject.Parse(paatos.FormData);

    foreach (var child in formData.Children())
    {
        var subItem = formData[child.Path] as JObject;
        data.Merge(subItem);
    }

    var kohde = _vahtiService.Kohde_Get(paatos.ArkistoSPAv2.Target_ID.Value, Data.Filters.KohdeIncludes.Viranomainen | Data.Filters.KohdeIncludes.Organisaatio | Data.Filters.KohdeIncludes.KohdeAsia);

    AddSupervisingElyCenterToJson(kohde, data);
    AddKasittelijaTieto(paatos.ArkistoSPAv2, formData);
    AddCustomerInfos(paatos.ArkistoSPAv2, kohde, data);
    AddDates(paatos, data);
    AddHallintoOikeus(paatos.ArkistoSPAv2, data);

    return _asposeClient.LuoDokumentti(data.ToString(), template, asposOutputFormat);
}
```

Kuva 24. Esimerkki metodista, joka asettaa dokumenttia varten tiedot.

Koodi asettaa foreach-silmukassa data -nimiseen JObject:iin automaattisesti päätös dokumentin JSON-datan mukaiset avaimet ja arvot. Koska tämän objektin mukaan tarvitaan arvoja, joita alkuperäisessä päätös lomakkeen datassa ei ollut, laadin apumetodeja täyttämään nämä tarpeet. Erittelin lisätietojen lisäämisen aiheryhmittäin eri metodeihin selkeyden vuoksi.

Tästä esimerkkinä kuvan 25 AddDates-metodi, joka lisää data-objektiin kaikki päivämääriin liittyvät tiedot.

```
4 references | Panu Siik, 93 days ago | 1 author, 1 change | 11 work items
private void AddDates(LomakePaatos paatos, JObject data)
{
    var hakemusPvm = paatos.ArkistoSPAv2.Luotu.ToString("dd.MM.yyyy");
    var kasittelyynottoPvm = paatos.ArkistoSPAv2.OtettuKasittelyyn?.ToString("dd.MM.yyyy");
    var muokkausPvm = paatos.ArkistoSPAv2.Muokattu.ToString("dd.MM.yyyy");
    var paatosPvm = paatos.Luotu.ToString("dd.MM.yyyy");
    var voimassaoloAikaArray = paatosPvm.Split('.');
    var voimassaoloAika = $"{voimassaoloAikaArray[0]}.{voimassaoloAikaArray[1]}.{DateTime.Now.Year + 3}";

    data.Add("dates", new JObject
    {
        {"ilmoitusPvm", hakemusPvm },
        {"paatosPvm", paatosPvm },
        {"voimassaoloAika", voimassaoloAika },
        {"kasittelyynottoPvm", kasittelyynottoPvm },
        {"muokkausPvm", muokkausPvm }
    });
}
```

Kuva 25. Esimerkki lisätietojen asettamisesta.

Lisäksi data-objektiin lisätään paljon muitakin tietoja samalla menetelmällä, mutta niissä toistuu sama perusajatus.

Kun olemme asettaneet kaikki dokumentin luontiin tarvittavat arvot koodissa, ajettaessa sovellusta voidaan generoida dokumentti käyttäen Asposea, jolla saamme luotua Word tai PDF -tiedoston, joka sisältää haluamamme arvot seuraavan esimerkki kuvan tavalla määritetyissä paikoissa.

Elinkeino-, liikenne- ja ympäristökeskus

Päätös {{Diaarinumero}}

{{dates.paatosPvm}}

ASIA Päätös ympäristösuojelulain (527/2014) 136 §:n mukaisen pilaantuneen maaperän puhdistamista koskevan ilmoituksen johdosta.

ILMOITUKSEN TEKIJÄN NIMI JA OSOITE

{{IlmoituksenTekijaNimi}}

{{Postinumero}} {{Postitoimipaikka}}

KUNNOSTETTAVAN ALUEEN SIJAINTI

{{Kyla}} {{Kaupunki}}
Kiinteistöt:
{{#foreach Kiinteistotiedot}}{{KiinteistonNimi}}, {{KiinteistoKuntaTunnus}}-
{{KiinteistoKylaTunnus}}-{{KiinteistoTaloKortteliTunnus}}-
{{KiinteistoTilaTonttiTunnus}},

{{/foreach Kiinteistotiedot }}

Kuva 26. Word-sapluuna esimerkki.

Pääsääntöisesti käytössä oli kaksi eri syntaksia Word-sapluunoissa, joista molemmista on esimerkki yläpuolen esimerkkikuvassa. Lisäksi Word-sapluunaan on mahdollista luoda totuusarvomuuttujan mukaan toimivia kenttiä, mikäli käyttöliittymällä on valintaruutuja, tai kyllä/ei -valintoja.

Ensimmäinen vaihe on tässä työjonon prosessiketjussa monimutkaisin. Kun dokumentin luonti onnistuu, on se valmis lähetettäväksi USPAan. Worker muuttaa arkistorivin tilan ”Dokumentit luotu”-tilaan, jonka prosessiketjun seuraava worker löytää ja ottaa prosessoitavaksi. Lisähuomiona se, että luotu dokumentti on tässä kohtaa Word-muodossa, koska halutaan säilyttää mahdollisuus muokkauksille USPA-vaiheen aikana. Mikäli halutaan luoda esikatselu versio lomakkeesta käsittelyn käyttöliittymältä, tiedostomuotona on PDFa.

5.2 **Vaihe 2 (Dokumenttien lähetys USPAan)**

Kuten aiemmassa vaiheessa, seuraava worker hakee arkistorivit, joiden tila on ”Dokumentit luotu”.

Tähän vaiheeseen sisältyy toimenpiteen luonti USPAan sekä asiakirjojen lisääminen toimenpiteelle. Tämä raportti ei tule kattamaan tätä vaihetta, koska kyseessä on ulkopuolinen järjestelmä. Kiteytettynä USPAssa käsittelijät tarkastavat dokumenttien sisällöt ja kun kaikki on kunnossa, ne arkistoidaan, sekä toimenpiteet linkitetään ja päätetään. Arkistoimisen yhteydessä USPA luo dokumentista PDF-tiedoston, johon on lisätty teksti sähköisestä hyväksynnästä, jossa on päivämäärä, käsittelijöiden nimet sekä roolit käsittelyssä.

Kun dokumentit on onnistuneesti välitetty USPAan, päivitetään arkistorivi tilaan ”USPA Luotu”.

5.3 **Vaihe 3 (Dokumenttien vastaanotto USPAsta)**

Tämä worker etsii arkistoriveistä kaikki ne rivit, joiden tila on ”USPA Luotu” ja tarkistaa säännöllisin väliajoin dokumenttien tilan. Kun dokumentit on arkistoitu ja asia päätetty, muutetaan arkistorivi tilaan ”USPA valmis”.

5.4 **Vaihe 4 (Asiakirjojen välitys asiointialustalle)**

Tämä worker on työjonoon liittyvän prosessiketjun viimeinen vaihe. Se etsii kaikki arkistorivit, joiden tila on ”USPA valmis” ja hakee USPAsta sen luomat uudet PDF-tiedostot, joissa on mukana aiemmin jo mainittu sähköinen hyväksyntä-teksti. Jokaisen dokumentin kohdalla tehdään tarkistus siitä, minkä tyyppinen lomake on kyseessä. Jos kyseessä on arkistotyyppi PIMA, tarkistetaan, onko lomake päätös vai hylkäys. Ero näillä käy ilmi asiointialustalla, sillä tässä tapauksessa päätös ei sulje asiointia, vaan avaa seuraavan lomakkeen täytettäväksi. Hylkäys sen sijaan sulkee asioinnin joka tapauksessa.

6 YHTEENVETO

Uudet käsittelyprosessit ovat olleet tuotannossa pääosin tämänlaisena käytössä alkuvuodesta 2020 lähtien. Lomakkeiden ulkoasua ja tietosisältöä, sekä käsittelyn näkymiä, prosesseja jatkokehitetään jatkuvasti vastaamaan uusia asiakkaan ja lainsäädännön asettamia tarpeita. Julkishallinnon digitalisaation tavoitteiden näkökulmasta päästiin askeleen lähemmäs päämäärää.

Kaiken kaikkiaan käytännön osuuden toteutus onnistui hyvin, vaikka työmääräarviot eivät ihan olleet tarkkoja ja tuli ylityksiä, koska kaikkia haasteita ei osattu ennalta tietää. Yksi näistä oli se, että esimerkiksi PIMAssa täytyi olla mahdollista aloittaa mistä tahansa kolmesta lomakkeesta siten, että prosessi etenee määritellyllä tavalla loppuun asti. Lisähaasteita toi myös asiointialustalla tämän ominaisuuden huomioon ottaminen sellaisella tavalla, että tämä muista asioinneista poikkeava toiminto toimii halutulla tavalla.

Olen oppinut huiman määrän uutta liittyen projektin käyttämiin teknologioihin sekä kokonaisuuden arkkitehtuuriin, mikä helpottaa tulevaisuudessa uusien ominaisuuksien kehitystyötä. Olen myös oppinut, kuinka tärkeä riittävän määrittelyn merkitys on kehitystyön kannalta. Kun ominaisuudet on määritelty riittävän hyvin, säästyy paljon aikaa myöhemmissä vaiheissa, kun todennäköisemmin toteutetut asiat ovat kerralla tehty oikein.

LÄHTEET

Dan Abramov & the Redux documentation authors. (2019). *Redux – A predictable state container for JS Apps*. Haettu 24.06.2019 osoitteesta https://redux.js.org

Facebook Inc. (2019). *React – A JavaScript Library for building user interfaces*. Haettu 24.06.2019 osoitteesta https://reactjs.org

JSON Schema. (2020). *JSON Schema is a vocabulary that allows you to annotate and validate JSON documents*. Haettu 04.01.2020 osoitteesta https://json-schema.org

Microsoft. (2020). *C# Documentation*. Haettu 03.01.2020 osoitteesta https://docs.microsoft.com/en-us/dotnet/csharp/

Mozilla. (2019). *JavaScript*. Haettu 20.12.2019 osoitteesta https://developer.mozilla.org/en-US/docs/Web/JavaScript

Parviainen P., Kääriäinen J., Honkatukia J., Federley M. (2017). *Julkishallinnon digitalisaatio – tuottavuus ja hyötyjen mittaaminen*. Haettu 13.01.2020 osoitteesta https://tietokayttoon.fi/documents/10616/3866814/3_Julkishallinnon+digitalisaatio+–+tuottavuus+ja+hyötyjen+mittaaminen/49e6b987-6d37-44dd-a86e-cc548fc66760?version=1.0

Quartz.NET. (n.d.). *Quartz.NET 3.0 API Documentation*. Haettu 05.01.2020 osoitteesta https://quartznet.sourceforge.io/apidoc/3.0/html/

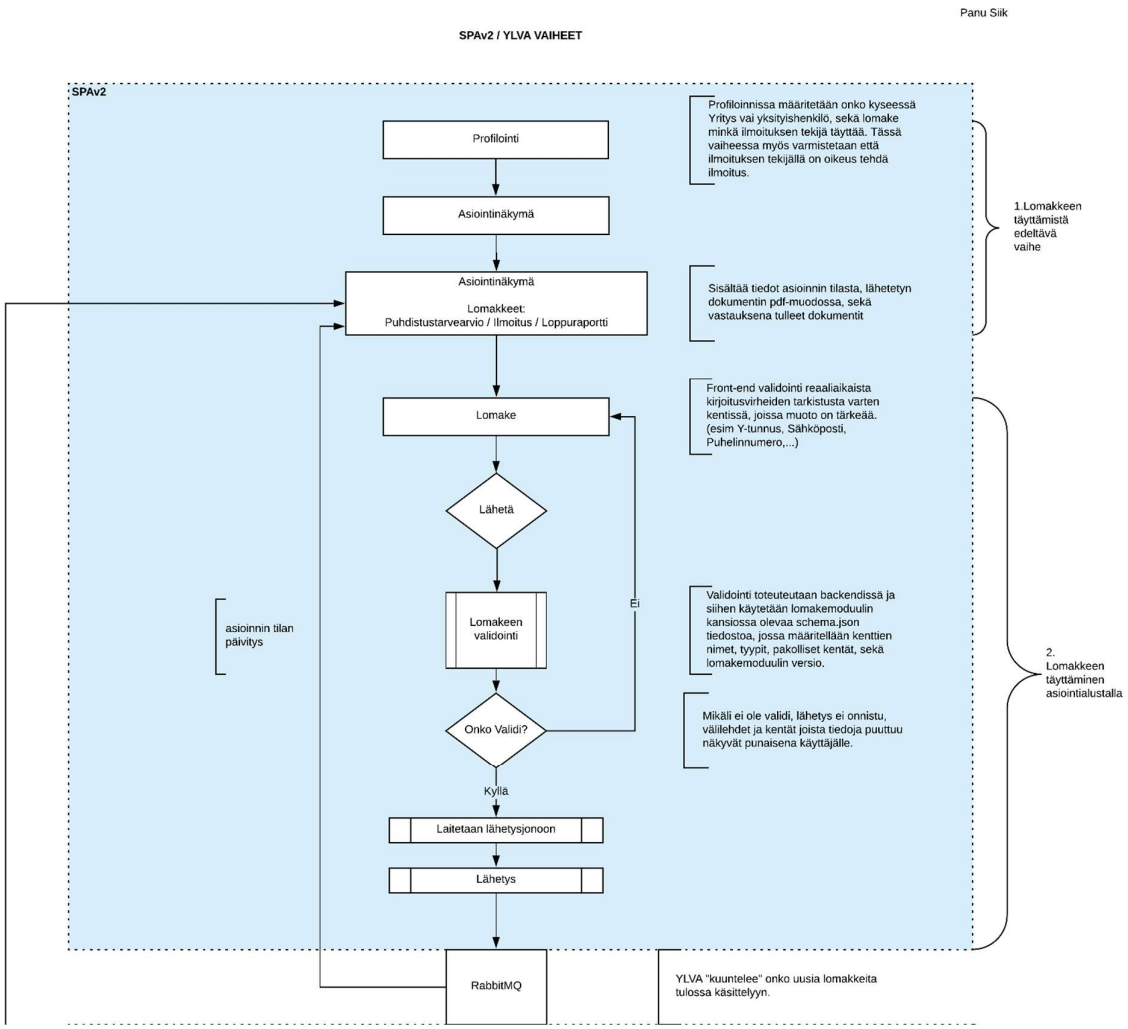
RamdaJS. (2019). *Ramda – A practical functional library for JavaScript programmers*. Haettu 25.06.2019 osoitteesta https://ramdajs.com

Valtiovarainministeriö. (2020). *Julkishallinnon digitalisaatio*. Haettu 13.01.2020 osoitteesta https://vm.fi/digitalisaatio

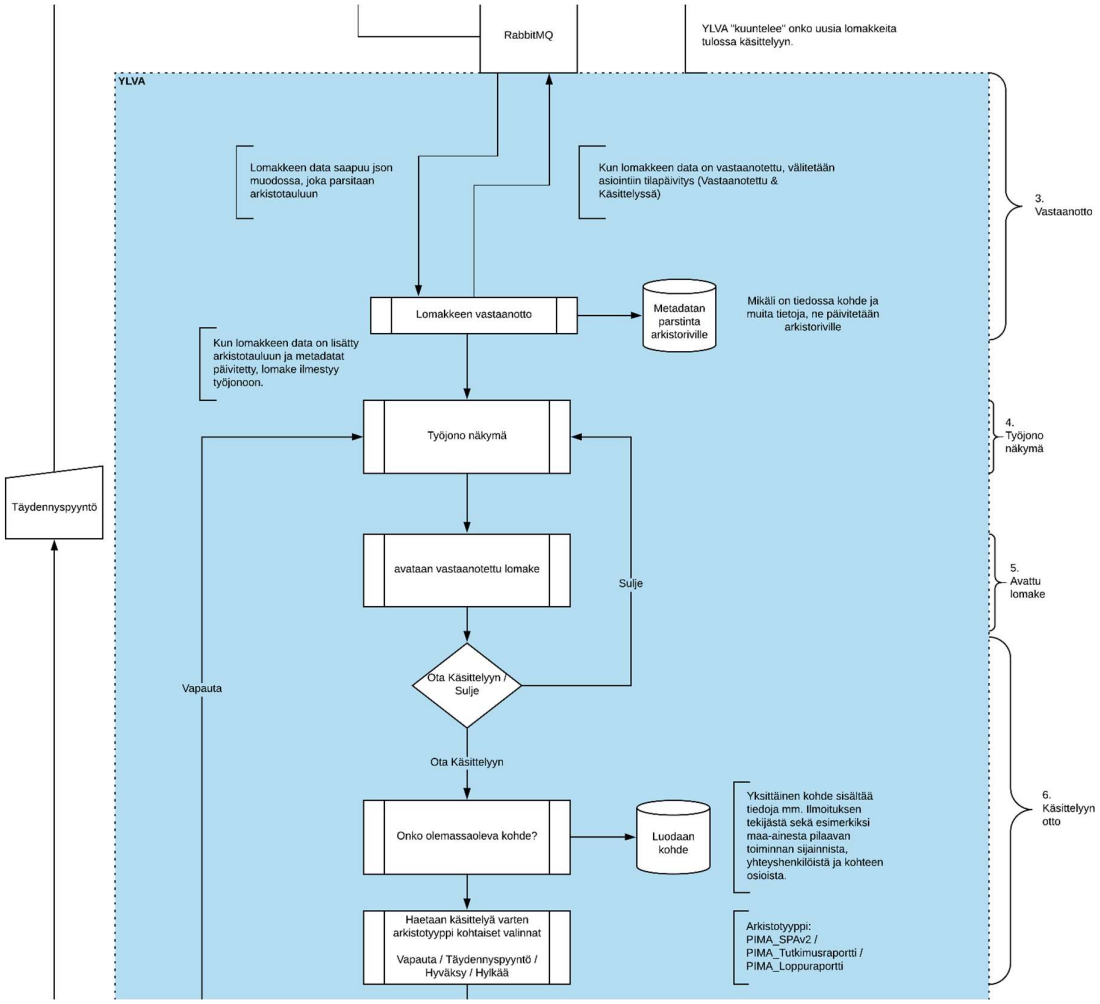
W3Schools. (2020). *JavaScript Versions*. Haettu 07.05.2020 osoitteesta https://www.w3schools.com/js/js_versions.asp

Liite 1

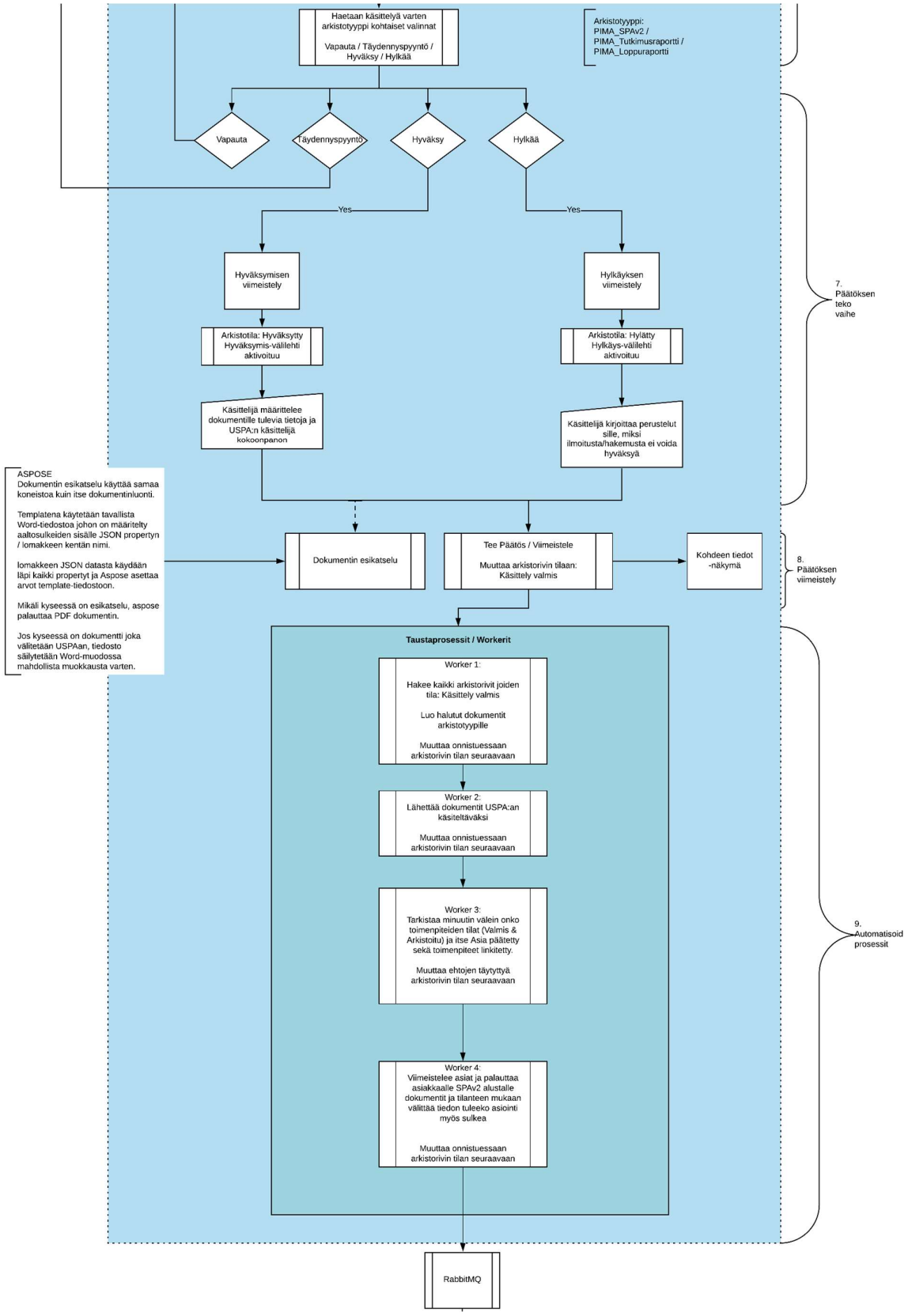
PROSESSIKAAVIO



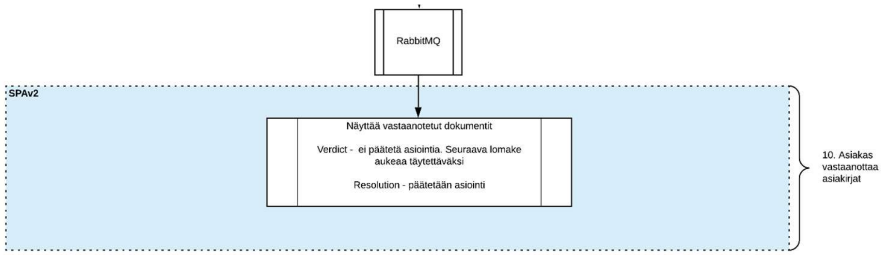
Jatkuu seuraavalla sivulla →



Jatkuu seuraavalla sivulla →



Jatkuu seuraavalla sivulla →


RabbitMQ
SPAv2
Näyttää vastaanotetut dokumentit
Verdict - ei päätetä asiointia. Seuraava lomake aukeaa täytettäväksi
Resolution - päätetään asiointi
10. Asiakas vastaanottaa asiakirjat