TAMPEREEN
AMMATTIKORKEAKOULU

Full-stack-ohjelmiston uudistaminen

Case Study: Visma Consulting Oy, TOJ 2.0

Väinö Niemi

Opinnäytetyö
Marraskuu 2018
Tietojenkäsittely
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Ohjelmistotuotanto

NIEMI, VÄINÖ:
Full-stack-ohjelmiston uudistaminen
Case Study: Visma Consulting Oy, TOJ 2.0

Opinnäytetyö 35 sivua, joista liitteitä 2 sivua
Marraskuu 2018

Tämän opinnäytetyön tavoitteena oli kehittää toimeksiantajan nykyisestä ohjelmistosta paremmin tarpeita täyttävä versio ja tutkia uudistamisprosessia. Rakentamisprosessissa käytettiin uusia teknologioita ja työkaluja mahdollisimman paljon. Tarkoituksena oli rakentaa itsenäisesti toimiva ohjelmisto, joka voidaan myöhemmin yhdistää toimeksiantajan järjestelmiin korvaamaan vanha ohjelmistoversio.

Ohjelmiston uudistamisprosessi on yllättävän suuri työ verrattuna normaaliin kehittämistyöhön. Tällaisessa prosessissa täytyy ottaa huomioon kohderyhmän tarpeiden ja käytettävien teknologioiden lisäksi myös vanhan ohjelmistoversion tuomat piirteet: mitkä asiat olivat toimivia kokonaisuuksia, pystytäänkö niitä hyödyntämään jatkossa ja mitkä asiat rajoittavat uudistamista.

Projekti antoi myös mahdollisuuden pilotoida uusia teknologioita käytännössä. Mahdollisten teknologioiden kirjo verkkosovelluksia varten on suuri. Lisäksi erilaisia kirjastoja ja ohjelmistokehyksiä tulee tasaisesti markkinoille. Projekti oli hyvä mahdollisuus kokeilla, olisiko joku näistä uusista teknologioista käyttökelpoinen myös muissa projekteissa jatkoa ajatellen.

Ohjelmiston uudistaminen saatiin projektin puitteissa toteutettua. Tämän jälkeen uusia teknologioita alettiin käyttämään muissakin toimeksiantajan projekteissa.

Asiasanat: angular, full-stack, javascript, java, spring

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Software Production

NIEMI, VÄINÖ:
Rebuilding a Full-Stack software
Case Study: Visma Consulting Oy, TOJ 2.0

Bachelor's thesis 35 pages, appendices 2 pages
November 2018

The objective of this thesis was to develop a new version of an old, insufficient software and to research the rebuilding process. This study was carried out as a project. New technology and tools were used as much as possible during this project. The purpose of this thesis was to design and implement an independent software that could be integrated into the existing applications used by the client.

There is a surprising amount of work related to upgrading existing software compared to developing a new one from the beginning. In this kind of process, multiple things must be considered in addition to customer needs and existing software features, as well as new technologies: what parts of the existing applications work, can these parts be used in the new version and what kind of limitations arise from the old technology?

The project also allowed the testing of new technologies in practice. The amount of possible technologies related to web-based applications is large. In addition, more libraries and frameworks keep steadily flowing to the market. The project was a good opportunity to try out if any of these fresh technologies were usable in the client's other projects.

Upgrading and implementing the software was finished in the project. The client has now applied what was learned into new projects as well. The research done also helps to ease the process of rebuilding software in the future.

Key words: angular, full-stack, javascript, java, spring

SISÄLLYS

LYHENTEET JA TERMIT

Angular, Angular.js	verkkosovelluskehitykseen tarkoitettu JavaScript-pohjainen ohjelmistokehys
Apache Maven	pääasiassa Java-ohjelmistojen paketoinnin automatisointiin tarkoitettu työkalu
Apache Tomcat	palvelinympäristö, jossa voi ajaa Java-pohjaisia sovelluksia
Client	Asiakas- tai pääteohjelma. Ottaa yhteyden palvelimeen ja välittää viestejä käyttöliittymän ja palvelimen välillä.
CSS	tyylikieli, jota käytetään verkkosivujen ulkoasun tyylittämiseen
Docker	Sovellusten ajamisen ja kehittämisen helpottamiseksi tehty työkalu. Dockerilla voidan luoda kontteja, jotka sisältävät kaikki sovelluksen ajamiseen vaadittavat asiat käyttöjärjestelmää myöten. (Opensource.com)
Full-stack-ohjelmisto	ohjelmistokokonaisuus, joka sisältää esimerkiksi käyttöliittymä-, palvelin- sekä tietokantasovelluksen. Useimmiten puhutaan full-stack-kehittäjistä, jotka osaavat tekniikoita liittyen kaikkiin ohjelmistokokonaisuuden osiin liittyen.
HTML	merkintäkieli, jota käytettään verkkosivujen rakentamiseen
”IoC”, Inversion of Control	Sovelluskehitysmenetelmä, jossa ohjelmistokehys tai -kirjasto kattaa yleiset ohjelmistoon liittyvät toiminnot, kuten tietokantakutsut tai verkkorajapintojen hallinnan. Se kutsuu rajapintojen kautta tiettyyn tarpeeseen tehtyjä erikoistuneita funktioita.
Java	olio-ohjelmointikieli
JavaScript, JS	ohjelmointikieli
JSON	tiedostomuoto, jota käytetään pääasiassa tiedonvälitykseen
MVC, Model-View-Controller	MVC on suunnittelumalli, jota käytetään pääasiassa verkkosovellusten tekemiseen. Malli(Model) sisältää bisneslogiikan esim. datan manipulointia varten. Näkymä(View) tarkoittaa

	käyttäjälle näkyvää käyttöliittymää. Nämä erotetaan toisistaan riippumattomiksi kokonaisuuksiksi. Ohjain (Controller) vastaanottaa käyttäjältä saatuja käskyjä sekä muuttaa näkymää sekä mallia saatujen ohjeiden mukaan (Holzner, Steven. 2006).
MySQL	relaatiotietokantaohjelmisto
NoSQL-database	dokumenttipohjaisia tietokantoja tarkoittava yläkäsite
Refaktorointi	olemassa olevan ohjelmistokoodin muokkausta siten, että ohjelmiston toiminnallisuus säilyy samana, vaikka ohjelmiston rakenteet tai funktiot muuttuvat.
Spring Framework	Java-pohjainen sovelluskehys
SQL, Structured Query Language	ohjelmointikieli, jolla rakennetaan relaatiotietokantoja

1 JOHDANTO

Ohjelmiston refaktorointi tai uudistaminen ovat monelle ohjelmoijalle tuttuja käsitteitä. Moni ohjelmoija pitää uudelleenrakentamista hyvänä asiana: puhtaalta pöydältä aloittaminen vaikuttaa usein parhaalta ratkaisulta. Sen vaatima työmäärä kuitenkin monesti yllättää.

Uudelleenrakentamista tehdään monista eri syistä: Vanha ohjelmisto ei toimi enää halutulla tavalla tai ohjelmisto alkaa olemaan liian ikääntynyt tai vaikeaselkoinen ylläpidettäväksi. Jälkimmäinen vaihtoehto tulee monesti eteen tilanteessa, jossa vanhan ohjelmiston alkuperäiset kehittäjät eivät enää ole ylläpitämässä ohjelmistoa syystä tai toisesta, eikä kukaan uudemmista kehittäjistä ole perehtynyt ohjelmistoon syvemmin.

Vanhan ohjelmiston toiminnallisuus voi olla riittämätöntä useista syistä. Ohjelmiston koodi voi olla rakennettu tavalla, jonka jatkokehittäminen on haastavaa. Käytetty ohjelmistokehys voi myös olla jo niin ikääntynyt, että sitä ei enää ylläpidetä. Tällaisessa tapauksessa mahdolliset kehyksen ongelmat jäävät joko kehittäjien korjattavaksi (jos kyseessä on avoimen lähdekoodin projekti) tai se voi olla mahdotonta. Muita mahdollisia kehitystä estäviä ongelmia voi olla tietokantarakenteen hidas toiminta tai vaikea ylläpidettävyys.

Tätä aihetta lähestyttiin toimeksiantajan antaman työtehtävän ohella. Vanha ohjelmistoversio tuli päivittää nykypäivään uudella tekniikalla, jonka pohjalle sovellus uudelleenrakennettiin. Tämän uuden ohjelmistoversion tulisi pystyä tekemään samoja asioita kuin vanhan version, joskin paremmin tai nopeammin. Lisäksi uudelleenrakentamisen tavoitteena oli saada tehtyä paremmin ylläpidettävä ja jatkokehitettävä pohja ohjelmistolle.

Opinnäytetyön tavoitteena oli vertailla vanhan ohjelmiston refaktorointia ohjelmiston kokonaisvaltaiseen uudistamiseen. Tähän kuuluu muun muassa erilaisten ongelmien tuominen esille. Lisäksi pohditaan, milloin kannattaa refaktoroida uudistamisen sijaan. Opinnäytetyön tarkoituksena oli määritellä, toteuttaa ja ottaa käyttöön vanhan ohjelmistoversion pohjalta rakennettu uusi kokonaisuus. Ohjelmistoversiota myös vertaillaan vanhan version kanssa sekä tutkitaan mahdollisia jatkokehityssuuntia.

2 PROJEKTIN ESITTELY

2.1 Toimeksiantajan esittely, lähtökohdat ja tarvevaatimus

Projektin toimeksiantajana toimii Visma Consulting Oy. Visma on eräs Pohjois-Euroopan suurimpia ohjelmistotaloja. Visma Consulting on Visman IT-konsultointiin erikoistunut alajaosto, joka keskittyy muun muassa räätälöityihin asiakasratkaisuihin, palvelumuotoiluun sekä digitaalisiin palveluihin. Lisäksi Visma Consultingin Oy:llä (tästä eteenpäin Visma) on muutamia omia tuotteita.

Opinnäytetyön tekijä on toiminut Vismalla tämän sekä muiden projektien parissa nyt pari vuotta. Työhön liittyvän projektin toteuttamisessa toimi hänen lisäkseen yksi palvelinasiantuntija, yksi testaaja projektin loppuvaiheessa sekä muutamia muita työntekijöitä konsulttiapuna projektin eri vaiheissa.

Opinnäytetyön projektin tarve syntyi edellisen version toiminnallisten ongelmien kasvaessa tasolle, jossa niiden korjaus alkoi olemaan välttämätöntä. Lisäksi haluttiin tutkia uusia teknologioita, joita voitaisiin hyödyntää Visman muissa projekteissa.

2.2 Projektin tuotoksen kuvaus

Projektin lopputuotteena oli tiedonohjausjärjestelmä, jolla ohjataan asiakirjallisen tiedon muodostamista, käsittelyä ja hallintaa. Se perustuu sähköiseen arkistonmuodostussuunnitelmaan, jossa kuvataan tehtävien käsittelyvaiheet ja asiakirjatyypit sekä niiden oletusmetatietoarvot, esimerkiksi säilytysaika (Kansallisarkisto 2012).

Käytännössä järjestelmässä kuvataan säilöttävien asiakirjojen perustietoja, asiayhteyksiä, säilytysaikaa sekä salausluokituksia, muiden tietojen ohella. Järjestelmällä voidaan tehdä uusia suunnitelmia ja päivittää niitä eri aikajaksoille sopivaksi, esimerkiksi vuosittaisiksi suunnitelmiksi. Lisäksi järjestelmään tallennettavaa tietorakennetta tulisi pystyä muuttamaan asiakaskohtaisesti sopivaksi. Tämä edellyttää tietorakenteen suunnittelemista siten, että sen muuttaminen onnistuu mahdollisimman vaivattomasti.

Lopputuote asennetaan asiakaskohtaisesti niin, että itse käyttäjä ei tarvitse muuta kuin tietokoneen ja ajantasaisen verkkoselaimen sovelluksen käyttämiseen. Tuote on suunniteltu käytettäväksi kannettavalla tai pöytätietokoneella. Mobiilikäyttöisen version tekeminen ei ollut tarkoituksenmukaista: muut tätä ohjelmistoa hyödyntävät järjestelmät ovat myös pääasiassa tietokoneella käytettäviä.

2.3 Projektin työkalujen valinta

Projektin työkalut valittiin useampien muuttujien vaikutuksesta. Opinnäytetyön toteuttajan osaamisalat vaikuttivat työkalujen valintaan. Lisäksi Vismalla jo käytössä olleet työkalut vaikuttivat vaihtoehtoihin. Jos aletaan testaamaan uutta työkalua, tulee sen myös olla käytettävissä muiden jo käytössä olevien kanssa.

Projektin alussa tekijän osaaminen sijoittui lähinnä Java-ohjelmointikieleen, JavaScript-ohjelmointikieleen sekä HTML-kuvauskieleen ja CSS-tyylikieleen. Tietokantojen osalta tuttuja tekniikoita olivat SQL-pohjaiset tietokannat sekä mm. MongoDB -dokumenttitietokanta. Nämä osaamisalueet rajoittivat myös mahdollisia työkaluja.

Lopputuote rakentuu käyttöliittymäsovelluksesta, siihen liitetystä palvelinsovelluksesta sekä tietokannasta. Valmis tuote tullaan ajamaan yhtenä pakettina valitussa palvelinympäristössä.

2.3.1 Palvelinsovellus

Palvelinsovelluksen ohjelmointikielenä käytettiin Java-ohjelmointikieltä. Mahdollista olisi myös käyttää JavaScript-pohjaisia kirjastoja tai ohjelmistokehyksiä, mutta useamman uuden asian yhtäaikainen kokeilu veisi enemmän aikaa kuin projektille oli varattu. Lisäksi toimeksiantajalla käytetään pääasiassa Java-pohjaisia palvelinkehyksiä, joten toteuttamisvaiheessa löytyi apua tarpeen vaatiessa.

Projektissa päädyttiin käyttämään Spring-ohjelmistokehystä palvelimen rakentamisessa. Suurin osa kehitysajasta meni käyttöliittymäsovelluksen rakentamiseen uudella, toimeksiantajallekin tuntemattomalla ohjelmistokehyksellä. Ei haluttu, että palvelinsovelluksen

rakentamiseen menisi aikaa enemmän kuin olisi tarpeen. Spring oli toimeksiantajalla laajalti käytössä, ja näin ollen ongelmatilanteissa apua löytyi helposti.

Spring, tässä tapauksessa tarkemmin Spring Web MVC (Model-View-Controller) on ohjelmistokehys, joka erottaa sovelluksen toiminnalliset osat erilleen ja hoitaa niiden välisen kommunikaation (Tutorialspoint). Näin pystytään vaikuttamaan eri osien toiminnallisuuteen ilman, että hajotetaan toista.

Malli (Model) sisältää käytössä olevat datamallit ja datan manipulointiin liittyvät toiminnot. Näkymä (View) tarkoittaa tässä tapauksessa kehittämäämme käyttöliittymäsovellusta. Näkymä on vastuussa datan näyttämisestä käyttäjälle verkkoselaimessa. Ohjain (Controller) on näiden kahden välillä oleva palikka, joka välittää näkymästä saadut ohjeet oikealle mallille toteutettavaksi, ja päivittää näkymää näiden toimintojen tuloksena. (Tutorialspoint)

Sovelluksessa näkymän rooli annetaan käyttöliittymäsovellukselle, ja Springin avulla rakennetaan sekä malli että ohjain. Ohjaimena tässä tapauksessa tulevat toimimaan erilaiset rajapinnat, joiden kautta käyttöliittymä voi kommunikoida mallien kanssa.

2.3.2 Tietokanta

Datan tallentamiseen löytyy muutamia vaihtoehtoja. Pääasiassa projektiin sopivat vaihtoehdot ovat joko relaatio- tai dokumenttitietokantoja. Suurimmat erot näiden kahden välillä ovat niiden tietorakenteessa sekä tiedon käsittelytavassa.

Relaatiotietokannat perustuvat taulukkoihin ja niiden välisiin yhteyksiin. Jokainen taulukko sisältää pystysarakkeita, ja jokaiseen sarakkeeseen voidaan tallentaa yhdentyyppistä dataa, esimerkiksi kokonaislukuja tai tekstiä. Yksi rivi taulukossa vastaa esimerkiksi yhden tilauksen tietoja, jos puhuttaisiin esimerkiksi verkkokaupasta. Jokaisessa taulukossa täytyy olla vähintään yksi pystysarake, johon tallennetaan jokaista riviä kohden uniikki tunnistetieto. Se voisi tässä tapauksessa olla esimerkiksi tilausnumero tilausta kohden. (Ambysoft Inc.)

Näitä taulukoita voidaan yhdistää toisiinsa edellä mainittujen uniikkien tunnisteiden perusteella. Kuvitellaan esimerkiksi, että jokaisella tilauksella on tilauksen tehnyt asiakas.

Tietokantaan on tallennettu toiseen tauluun monia asiakkaita, joiden uniikkina tunnisteena toimii asiakasnumero. Tilauksen yhteyteen on tallennettu tilauksen tehneen asiakkaan asiakasnumero. Nyt, jos halutaan saada selville, että mitä tilauksia jollain asiakkaalla on tehtynä, voidaan käyttää hyödyksi tätä asiakasnumeroa tilausten hakemisessa.

Dokumenttitietokannat toimivat täysin toisenlaisella periaatteella. Relaatietokannassa jokaiselle tietotaululle määritellään rakenne sitä luotaessa. Tämä rakenne, ”schema”, määrittää esimerkiksi millaisia datatyyppejä mihinkin pystysarakkeeseen tallennetaan. Näin ollen relaatiotietokannan yhdestä taulusta saatava tieto on aina saman mallista.

Dokumenttitietokannassa dataa tallennetaan kantaan tyypillisesti JSON-muotoisena datana. Erilaisia dokumentteja voidaan koota kokoelmiksi, jotka vastaavat relaatiotietokannan tauluja. Kokoelman sisällä olevilla dokumenteilla ei kuitenkaan tarvitse olla samanlainen datamalli, vaan kehittäjät voivat itse päättää millainen malli kutakin dokumenttia kohden toteutuu. Jokainen dokumentti on itsenäinen eikä niiden tarvitse liittyä toisiin tallennettuihin dokumentteihin. (Amazon.com Inc.)

Projektissa valittiin käyttöön SQL-pohjaisen MySQL-relaatiotietokannan. Suurin syy valintaan oli vanhan tiedonohjausjärjestelmän jo käytössä oleva tietokantaratkaisu. Tuotetta kehitettäessä tuli ottaa huomioon siirtyvät asiakkaat, joilla oli käytössä vanha ohjelmistoversio. Nämä asiakkaat olivat ilmaisseet toiveen siirtää vanhasta versiosta datan käytettäväksi uuteen sovellusversioon. Dataa oli jo tallennettuna valtavat määrät, ja haluttiin, että aikanaan tapahtuvasta datan siirtämisestä ei tulisi liian iso ongelma tuotteelle. Siirto on kuitenkin tehtävä, sillä vanha datarakenne on liian kankea toiminnallisuuden kannalta.

2.3.3 Käyttöliittymäsovellus

Sovelluksen uudistamisen tarve on lähtöisin pääasiassa vanhasta käyttöliittymästä, muun muassa sen kankeudesta käytössä. Käyttöliittymän rakentamiseen on lukuisia erilaisia vaihtoehtoja ja ne rajattiin kolmeen erilaiseen kirjastoon tai ohjelmistokehykseen: Angular.js, Angular sekä React.js.

React.js on käyttöliittymän tekemiseen tarkoitettu kirjasto. React itsessään hoitaa lähinnä käyttäjälle näkyvän käyttöliittymän piirtämistä. Reactin hyviä puolia ovat kirjaston pieni

tiedostokoko sekä kehittämisen yksinkertaisuus. React ei kuitenkaan sovellu koko käyttöliittymäsovelluksen tekemiseen itsessään, jos tarkoituksena on tehdä paljon datan käsittelyyn ja validointiin liittyviä toimintoja. React tekee kuitenkin helpoksi muiden kirjastojen liittämisen sovellukseen, joka mahdollistaa tarpeellisten toiminnallisuuksien lisäämisen kokonaisuuteen.

Angular.js sekä Angular ovat käyttöliittymäsovelluksien tekemiseen tarkoitettuja ohjelmistokehyksiä. Google on kehittänyt molemmat versiot kehyksestä. Angular.js tarkoittaa kaikkia 1.x versioita, ja Angular on 2.x tai uudempia versioita koskeva nimi. Yleisesti nämä erotetaan kahdeksi eri kehykseksi sen takia, että 2.x versio on käytännössä kokonaan uudelleenkirjoitettu verrattuna aikaisempaan, eikä se ole suoraan yhteensopiva 1.x -versioiden kanssa. Tämä vaikuttaa isoimpana asiana siihen, millä tavalla ohjelmakoodia kirjoitetaan. Molemmat versiot pystyisivät kuitenkin tekemään samanlaisia asioita tämän projektin puitteissa.

Näkyvän käyttöliittymän piirtämisen lisäksi ne hoitavat datan sitomisen käyttöliittymän ja ohjelmakoodin välille. Angular itsessään toteuttaa myös aikaisemmin mainitsemaani MVC-mallia. Molemmat ovat melko iso kokoelma erilaisia ominaisuuksia, joista tärkeimpiä tässä projektissa olivat DOM-manipuloinnin hoitaminen sekä lomakevalidointi.

DOM-manipulointi onnistuu sekä Reactilta että Angularilta. Käytännössä tällä tarkoitetaan näkyvän HTML-sivun sisällön dynaamista muuttamista. React/Angular hoitavat tämän automaattisesti esimerkiksi silloin, kun jonkin muuttujan tietosisältö muuttuu. Ilman näitä työkaluja tämä toimenpide vaatisi useimmiten sivuston päivittämisen verkkoselaimessa, että muutokset näkyisivät käyttäjälle. Lisäksi muutokset täytyisi erikseen ohjelmoida tapahtumaan.

Lomakevalidointi on nimensä mukaisesti lomakkeisiin syötetyn sisällön validointiin tarkoitettu työkalu. Angularissa tulee mukana tällainen työkalu. Työkalulla voidaan esimerkiksi varmistaa, että syötetty sähköpostiosoite on oikean muotoinen sekä estää lomakkeessa eteneminen, jos validointi ei onnistu. Projektin vaatimuksiin kuului muun muassa dynaamisen lomakesisällön luominen, johon tällainen työkalu soveltuu mainiosti.

Koska tarkoituksena oli testata uusia työkaluja sovelluskehitystä varten, päädyttiin käyttämään uudempaa versiota Angularista. Angular.js oli jo toimeksiantajalla laajalti käytössä. Lisäksi Angular.js on siirretty LTS-kehityksen pariin. LTS tulee sanoista ”Long Term Support”, jolla tarkoitetaan käytännössä sitä, että vakaa sovellusversio siirretään ylläpidettäväksi. Tätä versiota ei enää rikasteta uusilla ominaisuuksilla, vaan lähinnä korjataan mahdollisia virheitä tai tietoturvariskejä, jos niitä esiintyy.

3 PROJEKTIN MÄÄRITTELY JA RATKAISTAVAT YDINONGELMAT

3.1 Vanhan sovellusversion esittely

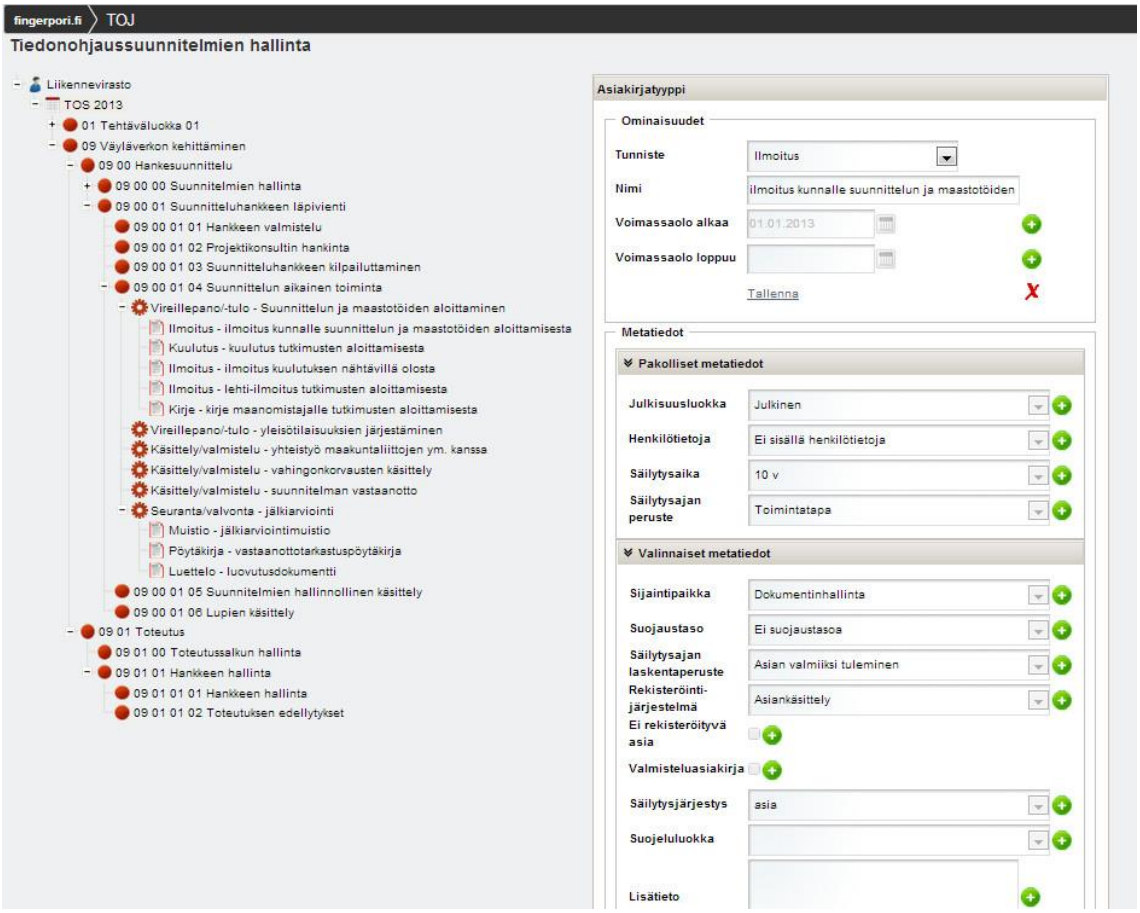
Vanha sovellus oli toteutettu noudattamaan SÄHKE2-määräyksen antamia ohjeita. Tämä järjestelmä pohjautui sähköiseen arkistonmuodostussuunnitelmaan, jossa kuvataan erilaisten toimintaan liittyvien tehtävien käsittelyvaiheet, asiakirjalliset tiedot ja asiakirjatyypit sekä niiden oletusmetatietoarvot (SÄHKE2-sertifiointi). Tällaista arkistonmuodostussuunnitelmaa voivat käyttää esimerkiksi kunnat tai muut organisaatiot. Suunnitelmasta voidaan käyttää myös nimitystä TOS, tiedonohjaussuunnitelma. Järjestelmän rooli organisaation tietojärjestelmäarkkitehtuurissa on toimia keskitettynä asiakirjallisten tietojen hallinnan välineenä sekä metatietojen ja niiden oletusarvojen lähteenä.

Vanhan sovelluksen käyttöliittymä (kuva 1) koostui pääasiassa navigointipuusta, jossa kuvataan julkaistu tiedonohjaussuunnitelma kuvassa vasemmalla, sekä valitun rakenteen tietojen esitysnäyttö kuvassa oikealla.

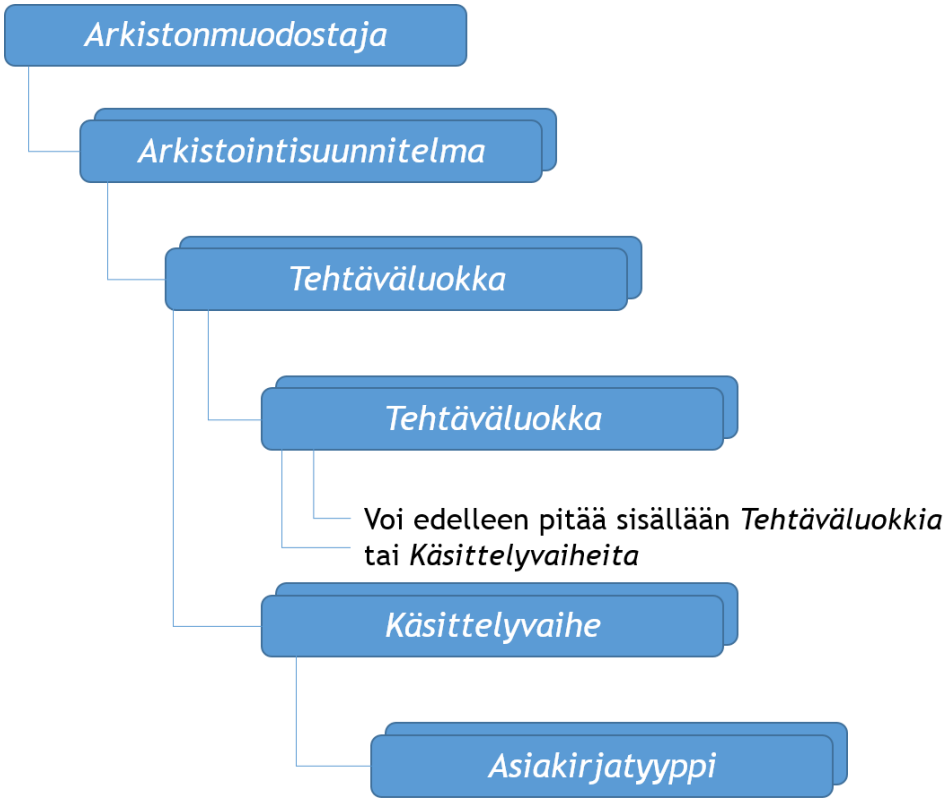
Navigointipuuhun on kuvattu erilaisilla ikoneilla eri osia. Nämä osat ovat hierarkkiseen rakenteeseen sijoitettavia osia, joiden tyyppi voi olla Arkistonmuodostaja, Arkistointisuunnitelma, Tehtäväluokka, Käsittelyvaihe tai Asiakirjatyyppi. Jokaiselle osalle voidaan kuvata SÄHKE-2 määräyksen mukaisia metatietoja ja niiden arvoja. Näiden osien keskinäinen mahdollinen järjestys on kuvattu kuvassa 2.

Jokaisella suunnitelman osalla on oma voimassaoloaika, jonka avulla useiden arkistointisuunnitelmien ja niiden osien versiointi sekä samanaikainen käsittely on mahdollista.

Suunnitelman osia voidaan kopioida ja liittää rakenteesta toiseen, lisätä sekä poistaa. Jokaiselle suunnitelman osalle voidaan määritellä omat oletusmetatietoarvot. Lisäksi rakenteessa ylemmällä tasolla olevan osan oletusarvot periytyvät alemmilla tasoille oleville osille, ellei niille erikseen määritetä omia arvoja. Poistettaessa rakenteen osa myös sen osan alla olevat osat poistetaan suunnitelmasta.



KUVA 1: Sovelluksen ulkoasu



KUVA 2: Mahdollisten rakenteiden järjestys suunnitelmassa

Osan tietojen esitysnäytössä olevat kentät ovat suurimmaksi osaksi SÄHKE2-määräyksen määräämiä vakiometatietoja varten. Asiakkaan on mahdollista lisätä ja poistaa omia metatietokenttiä vaadittujen vakiokenttien lisäksi. Tämä ei kuitenkaan ole määräyksen vaatima ominaisuus.

3.2 Ydinongelmat

Tässä osiossa esitellään isoimmat ongelmat vanhaan sovellusversioon liittyen. Näiden ongelmien ratkaiseminen on uuden sovellusversion ensisijainen tehtävä. Ongelmat olivat suurin syy siihen, että sovellusta alettiin uudistamaan refaktoroinnin sijaan.

3.2.1 Tietokantatoimintojen tehokkuuden lisääminen

Vanhan sovellusversion tietokanta oli rakennettu niin, että jokainen rakenteen osa on omassa taulussaan. Nämä rakenteet tietävät rakenteessa ylemmällä olevan osan, vanhempansa, tunnisteen. Tällä tavalla osista voidaan rakentaa tietokantahauilla rakenne, jossa osat ovat oikeassa järjestyksessä.

Osien omien metatietojen arvot säilytetään omassa tietokantataulussaan niin, että niitä voidaan hakea osan tunnisteella. Metatietoarvot on yhdistetty metatietoavaimiin näiden tunnisteella: metatietoavaimet sekä metatietokuvaukset ovat myös omissa erillisissä tauluissaan. Kuvaukset ovat arvoja, jotka vastaavat käyttöliittymässä näkyviä kenttien otsakkeita.

Tietokantatauluja löytyy näiden lisäksi muitakin, mutta ne eivät ole aivan niin relevantteja ongelman esittämisen kannalta. Kuvitellaan tilanne, jossa avataan rakenteesta osa, jolla on lapsiosia. Koska ylempänä oleva osa ei tiedä, mitkä osat ovat sen lapsia, joudutaan tekemään tietokantakutsu, jolla haetaan nämä osat tietoineen. Tämä tarkoittaa sitä, että aina rakenteen osia avatessa tehdään uusi tietokantakutsu. Lisäksi kutsun mukana haetaan useista muista tietokantatauluista esimerkiksi metatiedot ja metatietokuvaukset.

Kokonaisuudessaan tietokantakutsuja ja -toimintoja kertyy yhden osan avaamisesta useita. Kun rakennetaan uutta suunnitelmaa, näitä kutsuja kertyy satoja: suunnitelmat ovat laajuudeltaan melko isoja ja yhdessä pääasiallisessa tehtäväluokassa on kymmeniä

alaosia. Jokainen kutsu lisää aikaa minkä käyttäjä joutuu odottamaan ennen kuin hän pääsee jatkamaan suunnitelman luomista. Vaikka puhutaankin pienistä aikamääristä käyttäjän kannalta, palvelussa on kuitenkin huomattavissa tauko.

Edellä mainittu esimerkki liittyy vain osien tietojen hakemiseen. Sovelluksessa on kuitenkin mahdollista myös poistaa ja siirtää osia sekä muokata niiden metatietoja. Koska yhden osan käyttöliittymään tarvittavat tiedot ovat sidottu useaan tietokantatauluun, nämä kaikki toiminnot hidastuvat huomattavasti.

3.2.2 Useamman käyttäjän yhtäaikainen toimiminen reaaliaikaisesti samojen rakenteiden parissa

Vanhassa sovellusversiossa yhtäaikainen muokkaaminen oli periaatteessa mahdollista, koska tiedot päivittyivät jokaisella tietokantakutsulla käyttöliittymään. Ne eivät kuitenkaan automaattisesti muuttuneet tietojen muuttuessa kannassa, vaan luottivat lähinnä siihen, että ennen muokkausta on saatu uusimmat metatiedot ja rakenteen osat.

Tällaisessa toimintatavassa käyttäjien täytyy kommunikoida keskenään sen puolesta, mitä osaa he rakenteesta ovat muokkaamassa. Jos ajatellaan että kaksi käyttäjää avasi muokattavaksi rakenteesta saman osan metatiedot ja muokkasivat niitä, syntyi ongelma: kumman tallentamat tiedot tallennetaan kantaan? Jos osaa oli jo muokattu silloin, kun toinen käyttäjä on muokkaamassa osan tietoja, mitä tehdään, kun uudemmat muutokset lähetetään palvelimelle ja tiedot olivat muuttuneet välissä? Nämä ongelmat haluttiin ratkaista uudessa ohjelmistoversiossa niin, että vastaavia tilanteita ei syntyisi ja että saman suunnitelman yhtäaikainen muokkaus useamman käyttäjän osalta olisi kuitenkin mahdollista.

4 UUDELLEENRAKENNUS (REFAKTOROINTI) VAI SOVELLUKSEN UUDISTAMINEN

4.1 Yleistä ohjelmistojen ylläpidosta

Ennen kuin lähdetään pohtimaan näiden kahden eroja, on hyvä tietää, millaisesta työstä puhutaan. Ohjelmistokehityksessä projektien vaihejaolle on paljon erilaisia malleja, joista nykypäivänä ketterät menetelmät ovat melko käytettyjä. Näissä kaikissa malleissa on kuitenkin yhteisiä osia, joista itse ohjelmistojen rakentaminen muodostuu. Pääasiassa nämä osat ovat seuraavanlaisia: määrittely, suunnittelu, toteutus, testaus, käyttöönotto. (Harsu, 2003. s.16)

Nykyisellään paljon käytetyissä ketterissä projektinhallintamalleissa neljää ensimmäistä vaihetta pyöritetään järjestyksessä uudestaan ja uudestaan koko prosessin ajan, kunnes projekti on valmis lopputestaukseen ja käyttöönottoon. Aikarajoitetuissa projekteissa, mitä suurin osa projekteista on jollain tasolla, määrittely saatetaan jättää pois jossain iteraatiossa ja lisätä toteutuksen määrää. Jossain iteraatiossa taas testataan enemmän. Pääasia kuitenkin on, että nämä vaiheet toistuvat jokaisessa sovellusprojektissa.

Käyttöönoton jälkeen päästään yleensä tilanteeseen, jossa sovellus on sen hetken tarpeisiin valmis. Mitä tapahtuu, jos jokin ympäristöön liittyvä asia muuttuu tai sovellus ei toimikaan halutulla tavalla? Sovellukseen liittyvät rajoitteet, esimerkiksi koko- tai muistirajoitukset, tai sovelluksen käyttöympäristö, ovat voineet käyttöönottovaiheen jälkeen muuttua (Harsu, 2003, s.18). Tässä vaiheessa projekti ei lopu, vaan päästään käsiksi seuraavaan melko yleiseen osaan ohjelmistokehitystä: ylläpitoon. Harsun (2003, s.18) mukaan on arvioitu, että ohjelmistoyrityksissä noin 70% työajasta kuluu vanhojen ohjelmistojen ylläpitämiseen ja vain noin 30% uusien ohjelmien kirjoittamiseen.

Erilaisia ylläpitovaiheen tehtäviä voidaan jakaa seuraavaksi mainittuihin tyyppeihin: korjaavaksi, mukauttavaksi, täydellistäväksi tai ehkäiseväksi ylläpidoksi. Korjaavaa ylläpitoa tehdään silloin, kun löydetään sovelluksesta uusia virheitä tai virhetiloja, joita ei testausvaiheessa ole löydetty. Mukauttavaksi ylläpidoksi kutsutaan tilannetta, jos sovellusta käyttävää laitteistoa joudutaan syystä tai toisesta uusimaan, ja sovellus täytyy korjata toimimaan myös uudemmilla laitteilla. Täydellistävällä ylläpidolla yleensä lisätään ohjel-

mistoon uusia ominaisuuksia tai muita parannuksia, yleensä asiakkaiden toiveesta. Ehkäisevällä ylläpidolla parannetaan ohjelmakoodia niin, että sen ylläpito ja muuttaminen on jatkossa helpompaa. (Harsu, 2003, s.19)

4.2 Uudistaminen ja refaktorointi

Yleensä kun puhutaan refaktoroinnista, puhutaan edellisessä kappaleessa mainituista erilaisista ylläpitovaiheen tehtävistä. Olemassa olevaa sovellusta siis muutetaan tavalla, joka parantaa sen käytettävyyttä, korjaa virheitä, helpottaa ylläpidettävyyttä tai lisää ominaisuuksia tai tukea uusille alustoille. Refaktoroinnilla voidaan myös tarkoittaa joidenkin rakenteiden uudelleenrakentamista niin, että sovelluksen toiminnallisuus säilyy samana. Tällaisesta uudelleenrakentamista käytetään yleensä olio-ohjelmiin liittyen, joissa muutetaan niihin liittyvää luokkarakennetta tai rajapintoja järkevämmäksi (Harsu, 2003, s.22).

Ohjelmistojen uudistaminen eroaa ylläpidosta. Sillä tarkoitetaan yleensä laaja-alaisempaa toimintaa, jolla parannetaan sovelluksen rakennetta tai toiminnallisuutta suuremmalta alueelta kuin ylläpitotoimissa. Uudistamisen tavoitteena on saada vanhan sovellusversion pohjalta rakennettua uusi sovellus, joka on suunniteltu ja toteutettu käyttäen ajankohtaisia tekniikoita. Sovelluksen toiminnallisuuden, eli sen tarkoituksen, mitä varten sovellus on alun perin tehty, on yleensä tarkoitus pysyä samana. (Harsu, 2003, s.178)

Ohjelmistojen uudistamisen tarve voi johtua useista erilaisista syistä. Taustalla saattaa olla esimerkiksi halu tukea uudempia tai useampia laitealustoja, eikä vanha sovellus ylläpitotoimillakaan siihen enää taitu. Toisaalta sovelluksen käyttämä datarakenne tai ohjelmakoodi voi olla liian monimutkainen ylläpidettäväksi. Muita syitä voi olla esimerkiksi vanhentunut teknologia, jonka taitajia projektin ylläpitäjissä ei enää ole.

Uudistaminen itsessään on iso työ. Sovelluksen uudelleenrakentaminen voi viedä huomattavasti enemmän aikaa kuin ylläpitotoimilla saatavat muutokset, ja projektin vastuuhenkilöiden täytyy punnita tätä suunnitellessaan parannuksia sovellukseen. Sen lisäksi että sovellus rakennetaan kokonaan uudestaan, ehkä jopa uudella tekniikalla, täytyy ottaa huomioon myös toiminnallisuuden säilyttäminen vanhasta sovellusversiosta. Tämä voi jossain tapauksessa vaatia lisätyötä takaisinmallinnuksen muodossa.

Takaisinmallinnus (reverse-engineering) on termi, jolla kuvataan olemassa olevan sovelluksen purkamista osiin ja sen toiminnallisuuden tutkimista sekä analysointia (Harsu, 2003, s.22). Tätä voidaan tehdä erikseen tai liitettynä esimerkiksi sovelluksen uudistamisprosessiin. Jos esimerkiksi uudistettavan sovelluksen lähdekoodi ei ole saatavilla tai se on liian vanhaa tai vaikeaselkoista kehittäjille, voidaan sovellusta käyttämällä ja tietoa sen käytöstä keräämällä selvittää, miten sovellus oikeastaan toimii.

5 PROJEKTIN TOTEUTUS, CASE: VISMA CONSULTING TOJ 2.0

5.1 Tietokantarakenteen suunnittelu ja toteutus

Vanha tietokantaratkaisu ei tukenut versiointia. Lisäksi rakenne oli sekava ylläpidettäväksi ja melko monimutkainen kokonaisuutena. Projektissa haluttiin yksinkertaistaa kantarakennetta niin, että sen käyttäminen olisi nopeampaa ja lukeminen yksinkertaisempaa. Toimeksiantaja halusi kuitenkin pitää tietokantasovelluksen teknologian samana. Tästä johtuen päädyttiin käyttämään MySQL-tietokantasovellusta. Tämä helpottaisi tulevaisuudessa tapahtuvaa tietojen siirtämistä vanhasta tietokannasta uuteen.

Tietokantarakenne suunniteltiin uudella tavalla, jossa koko puurakenteen sisältö tallennetaan yhden tietokantataulun rivin alle JSON-muotoisena rakenteena. Tässä puussa eri rakenteen osat ovat kaikki samassa pinossa. Jokaiselle puun osalle luotiin oma tunniste, jonka perusteella ne osataan järjestää hierarkkiseen järjestykseen. Esimerkiksi kaikki Tehtäväluokka-tyyppiset osat ovat rakenteessa saman JSON-kokoelman alla. Liitteessä 1 on kuvattu esimerkkihakutulos toteutetusta rakenteesta.

JSON perustuu avain-arvopareihin, jossa avain kuvaa tietoa ja arvo on sitä vastaava tallennettu tieto. Arvo voi olla jokin yksittäinen arvo, esimerkiksi tässä tapauksessa koko toiminnanohjaussuunnitelmalle annettava tunnistetieto tai viimeisin muokkauspäivämäärä. Se voi olla myös kokoelma JSON-muotoisia objekteja tai arvoja. JSON-muotoisen datan hyvänä puolena on sen käsittelemisen helppous. Moni ohjelmointikieli omaa valmiita työkaluja JSON-datan käsittelyyn ja datan käsittely on melko nopeaa.

Palataan esimerkkihakutulokseen (Liite 1): ”root”-objekti on yhden suunnitelman rakenteen ylin osa. Jokainen rakenteen osa tietää, mitkä osat tulisivat olla niiden lapsiosia, eli puurakenteessa osan alla olevia osia. Tällä tavalla näistä irtonaisen näköisistä objekteista voidaan rakentaa kokonainen puu ohjelmallisesti sen sijaan, että siihen käytettäisiin tietokantatoimintoja.

Versiointi tämän tietorakenteen pohjalta on helppoa. Tietokantataulujen riveihin voidaan tallentaa rivin luontihetki todella tarkasti. Tämän avulla luotiin toinen tietokantataulu, jo-

hon voidaan tallentaa operaatioita; pieniä JSON-muotoisia olioita, joita voidaan palvelinsovelluksessa soveltaa yksitellen alkuperäiseen suunnitelmaan. Alkuperäistä suunnitelmaa ei muuteta siis koskaan.

Esimerkki: kuvitellaan, että olemme luoneet kantaan uuden suunnitelman viikko sitten. Suunnitelma on tällöin ollut tyhjä lukuun ottamatta suunnitelman ”root”-objektia. Olemme tehneet viikon aikana suunnitelmaan paljon uusia osia sekä siirrelleet niitä ympäriinsä ja poistaneet tarpeettomia osioita. Jokaisesta toiminnosta, jota puuhun tehdään, on tapahtunut operaatio, joka on tallennettu tietokantaan.

Kun tietokannasta nyt haetaan tämä sama suunnitelma, haetaan ensin pohjasuunnitelma sille generoidulla tunnisteella. Tämän jälkeen haetaan kaikki tähän suunnitelmaan liittyvät operaatiot tietokannasta yhdellä haulla. Ne järjestetään aikajärjestykseen vanhimmasta uusimpaan. Tämän jälkeen jokainen operaatio sovelletaan pohjasuunnitelmaan yksitellen. Lopputuloksena tästä saamme valmiin, ajantasaisen rakenteen.

Versiointiin tällainen rakenne soveltuu mainiosti. Nyt voidaan tarkastella puun tilannetta esimerkiksi kaksi päivää luomisen jälkeen aikamääreellä rajaten. Kannasta voidaan hakea kaikki operaatiot tähän aikamääreeseen asti, ja soveltaa vain ne palautettavaan rakenteeseen. Lisäksi operaatioihin voidaan esimerkiksi liittää tiedot käyttäjästä, joka on muokannut rakennetta.

Esimerkkirakenteen (Liite 1) viimeisenä objektina on ”locks” niminen kokoelma. Tämä liittyy yhteen vanhan sovelluksen ydinongelmista: miten usean käyttäjän yhtäaikainen saman suunnitelman muokkaaminen onnistuu niin, että tieto pysyy kaikilla käyttäjillä päivitettynä uusimpaan versioon?

Tämä ongelma ratkaistiin seuraavanlaisesti. Käyttöliittymässä osan muokkaaminen tai luominen suoraan ei onnistu. Sen sijaan henkilön täytyy lukita osa itselleen muokattavaksi. Tämä koskee myös tilannetta, jossa halutaan lisätä tai poistaa osioita tai siirtää niitä. Tämä lukkotieto kulkee puurakenteen datan mukana koko ajan, jolloin pysytään käyttöliittymässäkin kartalla siitä, mitä rakenteen osioita käyttäjällä on mahdollisuus muokata. Lukittua rakenteen osaa ei voi muokata kukaan muu kuin henkilö, joka on rakenteen lukinnut. Jos rakenteen osa tietää olevansa lukittu, voidaan ohjelmallisesti käyttöliittymässä sanoa myös sen lapsiosien olevan lukittuna tietylle henkilölle.

Tällainen ratkaisu vaatii ainoastaan sen, että kun tehdään tietokantaan kutsuja käyttöliittymästä, palautetaan vastaussanomassa aina puun sen hetkinen rakenne käyttöliittymälle. Näin pysytään melko tarkasti synkronoidussa tilassa käyttäjien kesken. Lisäksi osien lukitseminen estää päällekkäisten muutosten tekemisen.

5.2 Palvelinsovelluksen toteutus

Projektin palvelinsovelluksen rakentamiseen valittiin Spring Boot -ohjelmistokehys. Tämä sovellus yhdistää käyttöliittymäsovelluksen sekä tietokannan erilaisilla rajapinnoilla. Lisäksi palvelinsovellus on se osa koko projektista, jolla sovelluskokonaisuus ajetaan eri ympäristöissä.

Spring Boot on Springin päälle rakennettu ohjelmistokehys, jonka tarkoituksena on helpottaa Spring-sovelluksen konfigurointia sekä sen toteuttamista. Se sisältää useita helppokäyttötoimintoja, jotka vähentävät esimerkiksi käsin tehtävän konfiguroinnin määrää. Konfiguraatiot ovat ohjelmalle ajon aikana annettavia ohjeita, joilla voi muuttaa ohjelman toimintaa. Esimerkkinä tästä voisi olla esimerkiksi tietokantapalvelimen osoitteen konfigurointi: tällaisella konfiguroinnilla ohjelma tietää, mihin se ottaa yhteyttä, kun halutaan käyttää tietokantaan liittyviä toimintoja.

Sovelluksesta haluttiin tehdä itsenäisesti ajettava sekä asiakaskohtaisesti konfiguroitava. Vanha ohjelmistoversio oli käytännössä yksi sovellus, johon eri organisaatioiden käyttäjät voivat kirjautua muokkaamaan omia toiminnanohjaussuunnitelmiaan. Tämä tarkoittaa sitä, että asiakaskohtaisia muutoksia ei voida tehdä muuttamatta koko sovelluskokonaisuutta kaikille sovellusta käyttäville organisaatioille.

Vanhaa sovellusta käyttävät muut sovellukset kommunikoivat sen kanssa erilaisten rajapintojen kautta. Tämän vuoksi uuden sovelluksen integrointi näiden muiden sovellusten kanssa ei ole ongelma. Edellä mainittu ei kuitenkaan ollut tämän projektin puitteissa tehtävä työ, vaikkakin se oli hyvä ottaa huomioon jo suunnitteluvaiheessa.

Vanhassa versiossa jokaista organisaatiota kohden luotiin oma Arkistonmuodostaja-tyyppinen rakenteen juuriosa, joka näytettiin aina kirjautuneelle organisaatiolle. Uudessa rat-

kaisussa tällaista juuriosaa ei tarvitse erikseen näyttää, vaan se voitiin määritellä aikaisemmin mainittujen konfiguraatioiden avulla asiakaskohtaisesti. Tätä tietoa ei itse toiminnanohjaussuunnitelman luonnissa tarvita. Sen sijaan tieto täytyy olla saatavilla muille sovelluksille, jotka hyödyntävät luotuja suunnitelmia.

Palvelinsovelluksessa datan käsittelyä varten on luotu erilaisia malleja toimintoineen. Nämä mallit hyödyntävät Spring Data JDBC -ohjelmistokehystä, joka on saatavilla osana Spring -kehystä. Kirjasto helpottaa tietokantatoimintojen käyttöä huomattavasti: se tarjoaa erilaisia rajapintoja tietokannan käyttämisen perustoimintoihin, jolloin tietokantaa voidaan käyttää kirjoittamatta perinteisiä SQL-hakulauseita.

Toinen tärkeä ominaisuus joka palvelinsovellukseen tarvitaan, on datan validointi. Tällä tarkoitetaan syötteen eheyden ja oikean muodon tarkistamista. Jos sovelluksessa on esimerkiksi kenttä, johon kautta halutaan tallentaa vain numeroita, täytyy se ohjelmallisesti varmistaa. Suurin osa validoinnista tulee tapahtumaan käyttöliittymän puolella käyttäjän syötteen tarkastamisessa. Tämä ei kuitenkaan pelkästään riitä, sillä data, jota käyttöliittymästä lähetetään, on manipuloitavissa. Tällaisen tapahtuman riski pienenee käyttäjäkunnan myötä, sillä sovellusta ei voi käyttää muuta kuin palvelun käyttöön ottaneet organisaatiot.

Datan validointia varten palvelinsovellukseen lähetetty JSON-muotoinen data puretaan erilaisilla työkaluilla Java-olioiksi. Mainittavana esimerkkinä tällaisesta työkalusta on Jackson -niminen Java-kirjasto, jolla JSON-datan käyttäminen Java-ohjelmointikielessä helpottuu. Purkamisen jälkeen nämä oliot vastaavat erilaisia rakenteen osia ja niille tallennettuja arvoja. Kun data on purettu olioiksi, on niiden validoiminen helpompaa. Validoinnin jälkeen oliot kasataan takaisin JSON-muotoiseksi dataksi ja tallennetaan tietokantaan.

Springin avulla voidaan tehdä käyttäjänhallintaa melko yksinkertaisesti. Springiin on saatavilla Spring Security -ohjelmistokehys, jonka avulla voidaan esimerkiksi toteuttaa verkkopalvelun kirjautumisikkuna muutamalla koodirivillä. Tässä projektissa sovelluskokonaisuus ei tule olemaan oman kirjautumisensa takana, vaan kirjautumispalvelu rakennetaan erillisenä järjestelmänä. Sovellus vaatii tiettyjä tietoja tältä järjestelmältä, jotka tarvitaan, kun yritetään yhdistää sovellukseen. Näitä tietoja voivat olla vaikkapa kirjautumiseen käytetty nimi tai roolitieto. Kun sovellus saa tarvittavat tiedot, se päästä käyttäjän

sisään varsinaiseen sovellukseen. Tällaisessa mallissa kehittäjien ei tarvitse ottaa kantaa kirjautumisjärjestelmän toteutukseen.

Edellä selitettyä mallia kutsutaan termillä ”Pre-Authentication”. Tätä käytetään tilanteessa, jossa käyttäjä on jo tunnistautunut järjestelmään jossain toisessa, luotetussa lähteessä. Spring Securityn tarvitsee tällaisessa tilanteessa ensin tunnistaa käyttäjä, jonka jälkeen tunnistetulle käyttäjälle annetaan erilaisia vapaasti määriteltäviä oikeuksia esimerkiksi saadun roolitiedon perusteella. (Alex, B. & Taylor, L)

Sovelluksessa on käytetty kolmea eri tasoista oikeutta. Nämä oikeudet jaetaan kirjautumisjärjestelmältä vaaditun roolitiedon perusteella. Käyttäjät, jotka voivat tarkastella julkaistuja suunnitelmia, saavat ”read” eli lukuoikeuden. Suunnitelmia tekevät käyttäjät saavat ”write”-oikeuden, joka muokkaamisen lisäksi antaa mahdollisuuden luonnostella uusia suunnitelmia. Viimeisenä mahdollisena oikeutena on ”admin” -oikeutus, jonka avulla tietyille käyttäjille annetaan järjestelmän hallintaan liittyvät työkalut käyttöön. ”Admin”-oikeutetut käyttäjät voivat esimerkiksi poistaa muiden käyttäjien tekemiä lukituksia, kun normaali ”write”-oikeutettu käyttäjä ei pysty hallitsemaan muuta kuin omia lukituksiaan.

5.3 Käyttöliittymäsovelluksen toteutus

Käyttöliittymäsovellus on rakennettu käyttäen Angular 4.4.6 -versiota. Angular on verkkosovellusten rakentamiseen tarkoitettu ohjelmistokehys, joka on täysin uudelleenkirjoitettu versio Angular 1.x -versioista. Projektin alussa (joulukuu 2016-tammikuu 2017) Angular 2.x oli täysin uusi, vasta virallisesti julkaistu ohjelmistokehys. Käytetty versio vaihtui useaan otteeseen projektin aikana, ja joulukuussa 2017 päädyttiin jäädyttää versioiden päivittäminen mainittuun versioon.

Angular perustuu yksinkertaisimmillaan erilaisten komponenttien puusta, jotka voivat kommunikoida keskenään niiden syöterajapintojen kautta. Jokainen komponentti voidaan yhdistää toiseen Angularin käyttämän Dependency Injection-mekanismin avulla. (Deeleman, Pablo)

Dependency Injection on mekanismi, jonka avulla komponentteja voidaan kehittää vapaammin. Jos jokin komponentti vaatii vaikkapa kykyä yhdistää palvelimeen ja hakea sieltä tietoa, komponenttiin kirjoitetaan rivi koodia, joka kertoo, että tämä komponentti

vaatii kyseisen toiminnallisuuden. Mekanismi tarjoaa vastaavan toiminnallisuuden toteuttavan komponentin käytettäväksi ilman, että kehittäjän tarvitsee itse määritellä, mikä komponentti otetaan käyttöön. Kehittäjän tarvitsee vain tietää miten komponenttia tulisi käyttää. Tämä helpottaa kehitystä niin, että jos toiminnallisuuden tarjoama komponentti joudutaan vaihtamaan toiseen, toiminnallisuutta käyttävää komponenttia ei tarvitse muuttaa.

Sovelluksen suunnittelussa otettiin mallia vanhasta käyttöliittymästä toiminnallisuuden puolesta. Sovelluksessa haluttiin pitää sekä navigointipuu että metatietojen esittelynäkymän samoilla paikoilla kuin aikaisemmin. Sen lisäksi että haluttiin toteuttaa vanhan toiminnallisuuden uudella pohjalla, käyttöliittymästä haluttiin myös tehdä helpommin muokattava uusia ominaisuuksia varten.

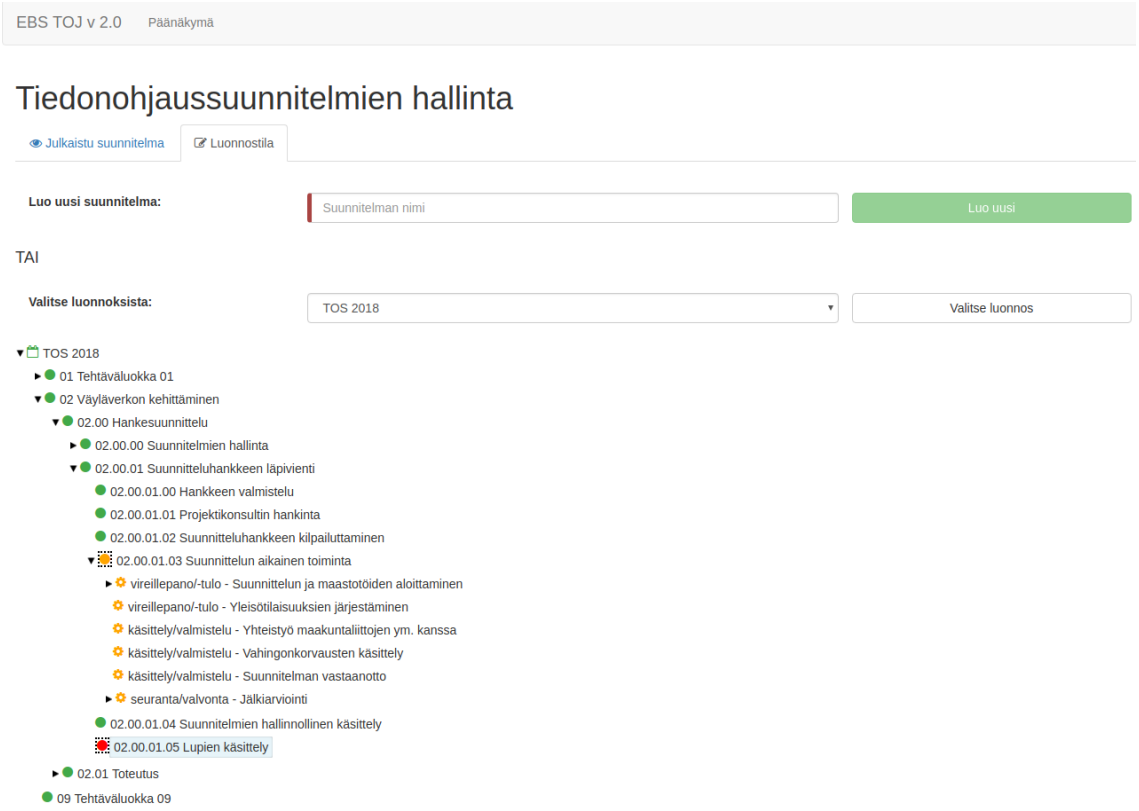
Angular soveltui tähän käyttötarkoitukseen hyvin. Angular on pääasiassa kehitetty TypeScript- ohjelmointikielellä, mikä on JavaScript-kielen pohjalle rakennettu kieli. TypeScript käännettään ajettaessa muotoon, mitä myös vanhemmat selaimet pystyvät lukemaan. Tällä tavalla saatiin käyttöön enemmän ominaisuuksia kuin pelkällä JavaScriptillä kirjoittaessa, ja ne ovat silti toimivia vanhemmillakin selaimilla. TypeScript tuo myös mukanaan tukea olio-ohjelmoinnille tyypitysten sekä rajapintojen muodossa. (Microsoft)

TypeScriptin avulla pystyttiin rakentamaan sovelluksen rakenteesta kestävämpi tekemällä enemmän uudelleenkäytettäviä komponentteja tiettyihin käyttötarkoituksiin. Esimerkiksi navigointipuun rakentaminen tapahtuu yhdessä palvelussa, navigointipuun malli on tallennettu yhteen rajapintaan, ja viimein yksi komponentti ottaa vastaan rajapintaa vastaavan olion joka sisältää puurakenteen tiedot ja piirtää puun tietojen perusteella.

Puurakenteen muokkaaminen ei ole mahdollista, ellei käyttäjä ole lukinnut rakenteen osaa itselleen. Lukitusten täytyy olla visuaalisesti selkeitä käyttäjälle yhdellä silmäyksellä, jotta puurakenteessa navigointi olisi mahdollisimman yksinkertaista.

Luonnostila on myös uusi ominaisuus tähän versioon. Aikaisempi versio salli useampien suunnitelmien luomisen yhtäaikaisesti, mutta suunnitelmien hallinta ei ollut kauhean selkeää käyttäjälle. Aikaisemmassa versiossa kaikki suunnitelmat olivat Arkistonmuodostaja-osan alla riippumatta siitä, olivatko ne tällä hetkellä voimassa olevia suunnitelmia

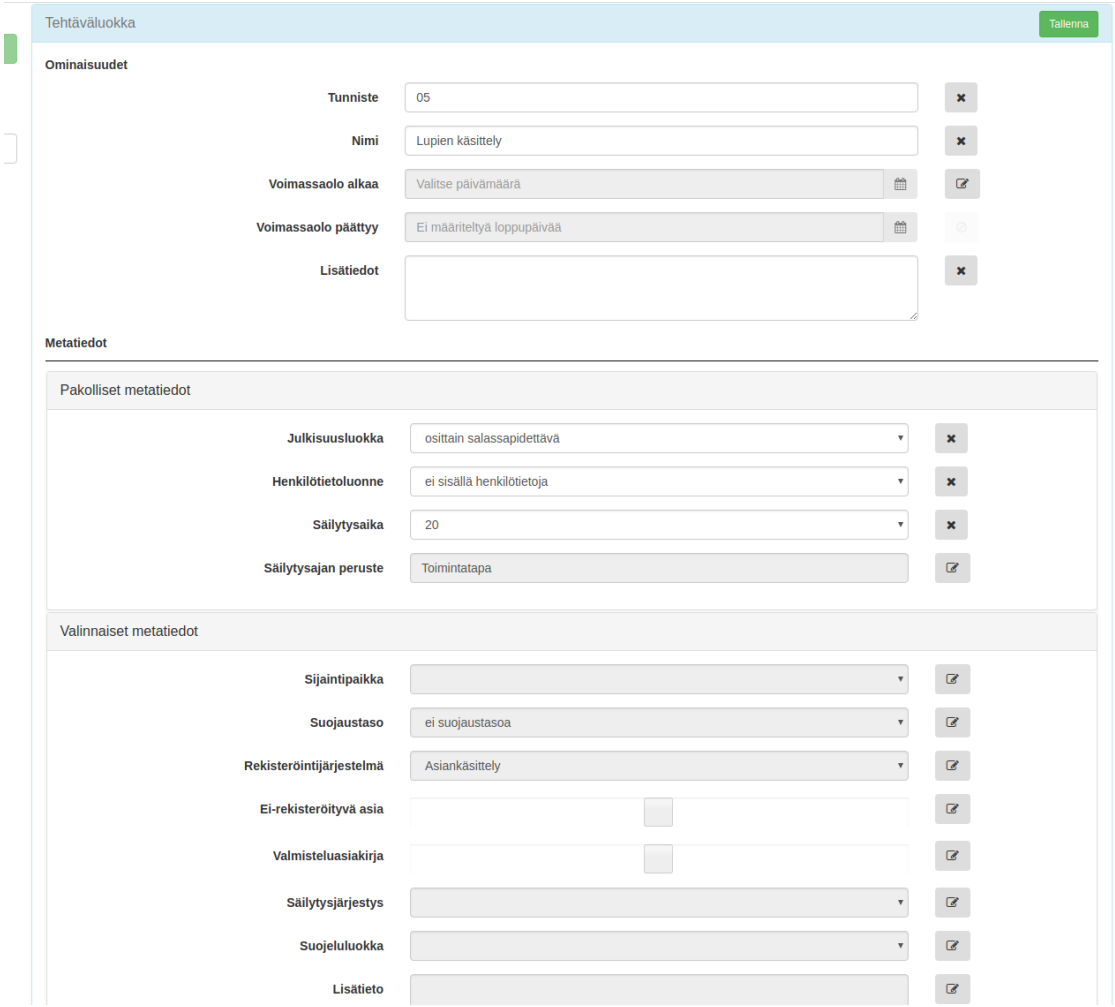
vai eivät. Lisäksi luonnossuunnitelmien hallinta oli sekavaa edellä mainitusta syystä. Tällä hetkellä voimassa oleva suunnitelma sekä uudet suunnitelmat haluttiin erottaa selkeästi. Kuvassa 3 nähdään otanta toteutetusta luonnosnäkymästä.



KUVA 3: Erään luonnoksen navigointipuu

Kuvassa punaisella näkyvät osat on lukittu käyttäjälle itselleen, ja keltaisella näkyvät osiot ovat lukittuja muille käyttäjille. Vihreät osiot ovat vapaita osioita muokkausta varten. Osioita ei kuitenkaan voi lukita, jos rakenteessa niiden alapuolella on lukittuja osioita. Tämä estää lukitusten menemisen ristiin. Värien lisäksi osan ikoni näyttää osan tyypin.

Osion valitsemalla saa näkyviin metatietojen esitysnäytön ruudun oikealle puolelle (Kuva 4). Metatietoja voi tarkastella kaikista rakenteen osista; myös niistä, jotka ovat lukittuna muille käyttäjille. Jos tietojen muokkaaminen on mahdollista, näytetään tietokenttien oikealla puolella ikoni, jolla kenttä voidaan avata muokkausta varten. Jos halutaan peruuttaa tehdyt muutoksen, voidaan nappia painaa uudemman kerran ja tieto palaa aikaisempaan tilanteeseen. Nappi kuvastaa tilannetta kenttäkohtaisesti ikonilla.



KUVA 4: Erään osan metatietojen esitysnäyttö

Suunnitelman voi ajastaa julkaistavaksi laittamalla Arkistointisuunnitelma-osalle voimassaolopäivämäärän. Julkaistut suunnitelmat ovat voimassa niin kauan, kun joku toinen suunnitelma tulee voimaan tai suunnitelmalle säädetty mahdollinen voimassaolon päättymispäivä tulee vastaan.

Puurakennetta ja esitysnäyttöä hallitsevien komponenttien lisäksi sovellukseen täytyi tehdä muun muassa reititys eri komponenttien välille, palvelu jolla kutsutaan palvelinsovelluksen rajapintoja datan manipulointia varten, validointipalvelut metatietojen esitysnäyttöä varten sekä kasa pienempiä toimintaan liittyviä osia.

Käyttöliittymäsovellus käännetään TypeScript-tulkin avulla vanhempien selaimien tukemaksi JavaScriptiksi, jonka jälkeen se paketoidaan yhdeksi ajettavaksi tiedostoksi. Samalla ohjelmakoodista tehdään sellaista, että sen lukeminen on vaikeaa selaimien tarjoamien työkalujen avulla. Tällä tavalla palvelun käyttäjät eivät saa selvää selaimeen toimitetusta ohjelmakoodista ja sen toiminnallisuuden purkaminen on vaikeampaa.

5.4 Käyttöliittymän ja palvelinsovelluksen yhdistäminen ja paketointi

Kun sekä käyttöliittymä- että palvelinsovellus ovat rakennettu, ne paketoidaan yhdeksi ajettavaksi tiedostoksi. Tämä helpottaa sovelluksen asentamista uusiin ympäristöihin sekä sovelluksen testaamista. Apache Maven on mm. Java-pohjaisten ohjelmistojen pakkaamiseen ja rakentamiseen tarkoitettu työkalu (The Apache Software Foundation).

Mavenilla voidaan antaa ohjelman rakentamiseen erilaisia ohjeita. Ohjeilla voidaan esimerkiksi määrittää erilaisia versioita sovelluksesta tai määrittää rakennettavaksi vain osa sovelluksesta. Siihen on myös tarjolla paljon liitännäisiä, joista käytämme projektissamme esimerkiksi käyttöliittymäsovelluksen rakentamista varten tehtyä liitännäistä. Liitännäinen tekee samat asiat kuin käyttöliittymäsovellusta rakentaessa, jonka jälkeen sovellus voidaan sijoittaa haluttuun kohteeseen pakkausta varten.

Pakkausvaiheessa Mavenilla rakennetut palvelin- sekä käyttöliittymäsovellukset laitetaan yhteen tiedostoon. Tämä tiedosto on tässä tapauksessa WAR-tiedosto (Web Application Archive). WAR-tiedostot ovat paketoituja verkkosovelluskokoelmia, jotka voidaan käynnistää missä tahansa ympäristössä, mitkä sallivat tällaisten kokoelmien ajamisen. Ne sisältävät ohjelmakoodin ja resurssien lisäksi ohjeita siitä, miten sovelluskokonaisuus voidaan ajaa ympäristössä sekä missä järjestyksessä sovellus ajetaan (Oracle). Kun WAR-tiedosto on koottu onnistuneesti, se voidaan ottaa talteen ja siirtää haluttuun ympäristöön ajettavaksi.

5.5 Lopputuotteen ajaminen ja suojaus

Lopputuotteen paketoinnin jälkeen sovellus on valmis. Tässä kohtaa ongelmaksi tulee kuitenkin ympäristö, missä sovellus ajetaan. Asiakkaiden omat ympäristöt (jos on) voivat

olla todella erilaisia käyttöjärjestelmästä ja niihin asennetuista ohjelmistoista lähtien. Sovelluksen ajamiseen vaadittiin tietynlainen ympäristö ja tietyt ohjelmistot, jotta sovelluskokonaisuus toimii. Tässä kohtaa käyttöön otettiin Docker.

Docker on avoimen lähdekoodin työkalu, jolla voidaan luoda käyttöjärjestelmästä riippumattomia kontteja. Näille konteille voidaan antaa ohjeet niihin asennettavista ohjelmistoista, käyttöjärjestelmästä sekä siitä, mitä ne tekevät käynnistyessään. Konteista voidaan tehdä joko itsenäisesti ajettavia tai isäntäkoneeseen tukeutuvia, jolloin ne jakavat resursseja, esimerkiksi käytettyjä ohjelmistoja tai käyttöjärjestelmän ominaisuuksia, isäntäkoneensa kanssa. (Opensoftware.com)

Sovellusta varten toteutettiin itsenäisesti ajettava Docker-kuva, josta voidaan käynnistää kontti missä tahansa Docker-sovellusta tukevassa ympäristössä. Tähän kuvaan sisällytettiin kaikki ohjelmistot mitä sovelluskokonaisuuden ajamiseen tarvittiin, esimerkiksi Java-ohjelmointikielen ajamiseen vaadittavat ohjelmistot. Lisäksi kontin käynnistyksen jälkeen kontti ohjeistettiin tekemään uusi suojattu palvelin, jonka taakse koko sovelluskokonaisuus käynnistettiin.

Tämä palvelin suojaa pääsyn itse sovellukseen, sekä antaa sille sen tarvitsemat tiedot Springin Pre-Authentication -prosessia varten. Tämä palvelin voidaan asiakkaalle asentamisen yhteydessä vaihtaa toiseen ilman, että sovelluksen toiminta estyy. Tämä mahdollistaa sovelluksen integroinnin, eli yhdistämisen, olemassa oleviin järjestelmiin. Tilalle voidaan esimerkiksi vaihtaa toisen järjestelmän palvelin, jonne ohjeistetaan reitit sovellukseemme. Näin voidaan myös antaa pääsy toisille sovelluksille näihin reitteihin esimerkiksi tietojen hakua varten.

6 POHDINTA

Tavoitteena oli kehittää uusi sovelluskokonaisuus korvaamaan toimeksiantajan vanhaa sovellusta sekä tutkia uudistamisprosessia korvaamisen ohella. Sovellus on tällä hetkellä toiminnassa ja testikäytössä potentiaalisia uusia asiakkaita varten. Sovelluksen jatkokehitys jatkuu sitten, kun saadaan ensimmäiset asiakkaat siirtymään uuden järjestelmän pariin.

Verrattuna vanhaan versioon uusi versio toimii huomattavasti nopeammin. Vaikka datarakenteen muutos oli iso työ ja se toi mukanaan suuriakin suunnitteluongelmia, selvittiin niistä hyvin. Pääasiassa sovelluksen nopeus johtuu tästä datarakenteen muutoksesta, mutta suurena osatekijänä on myös uusi käyttöliittymäsovellus. Datarakenteen muutokset vähensivät palvelinsovellukseen tehtäviä rajapintakutsuja yli puolella, ja tätä määrää voidaan edelleen optimoida jatkokehityksen aikana.

Jatkokehityksen kannalta Angular osoittautui hyväksi vaihtoehdoksi käyttöliittymäsovelluksen tekemiseen. Uusien toiminnallisuuksien lisäämien käyttöliittymään ei ollut alkukankeuden jälkeen erityisen haastavaa, ja ne saatiin myös toimintaan nopealla aikataululla. Muutama uusi ominaisuus saatiin myös toteutettua sovellukseen uudistamisprosessin aikana.

Uudistamisprosessi vei paljon enemmän aikaa kuin alun perin suunniteltiin. Projektin alkuaikana projektissa arvioitiin olevan työtä noin vajaaksi vuodeksi käyttöönottotoimenpiteet mukaan lukien. Projektin aikana kuitenkin huomattin, että aikataulu venyi muun muassa uuden teknologian oppimisen haastavuuden vuoksi. Uudistamisprosessi kesti kokonaisuudessaan n. 1,5 vuotta. Tänä aikana saatiin kuitenkin arvokasta kokemusta Angular-ohjelmistokehyksestä.

Ohjelmiston uudistaminen oli tämän projektin puitteissa tarpeellinen tehtävä. Refaktorointi tai muut ylläpidolliset toimet olisivat voineet korjata osan ongelmista ainakin lähitulevaisuutta ajatellen. Vanhan version pohja oli kuitenkin vaikeaselkoista ja refaktorointi sen takia haastavaa. Lisäksi tietokantarakenne vaati uudistamisprosessia, jolloin tässä projektissa se oli loogisempi vaihtoehto.

Entä jos ohjelmiston pohja ei ole vanhentunut liikaa tai ohjelmakoodi on tarpeeksi selkeää? Tässä projektissa huomattiin, että uudistaminen vie paljon aikaa ja prosessin suunnittelu on vaikeaa. Refaktorointi vie useimmissa tapauksissa todennäköisesti vähemmän aikaa, jos pohjatyö on tehty kestävällä tavalla. Lisäksi uudistamisessa täytyy ottaa huomioon uudet käytetyt teknologiat. Vaikka uudistettu sovellus toimisikin paremmin, ei pidä unohtaa tämän uuden version ylläpidollisia haasteita. Täytyy ottaa huomioon ylläpitäjien taitotaso ja teknologiatietämys sekä mahdolliset kouluttamisen tuomat kustannukset.

Määrittelyn aikana sovelluksen haluttuihin ominaisuuksiin listattiin suuri määrä toiminnallisuuksia, joista noin neljäsosa jäi kuitenkin toteuttamatta aikataulukiireiden vuoksi. Nämä jätetyt ominaisuudet on tarkoitus tehdä jatkokehitysprojektina myöhemmässä vaiheessa sovelluksen elinkaarta. Kehitetty kokonaisuus kuitenkin soveltuu siihen paremmin ja on myös uusille kehittäjille selkeämpi tutustuttava.

LÄHTEET

Amazon.com Inc. What is a Document Database? n.d. Luettu 4.11.2018. https://aws.amazon.com/nosql/document/

Ambysoft Inc. Relational Databases 101: Looking at the whole picture. n.d. Luettu 4.11.2018. http://www.agiledata.org/essays/relationalDatabases.html

Alex, B. & Taylor, L. Spring Security Reference Documentation. Pre-Authentication Scenarios. n.d. Luettu 8.11.2018. https://docs.spring.io/spring-security/site/docs/3.0.x/reference/preauth.html

Deeleman, Pablo. 2016. Learning Angular 2. Birmingham, Englanti: Packt Publishing Ltd. ISBN 978-1-78588-853-3.

Harsu, Maarit. 2003. Ohjelmien ylläpito ja uudistaminen. Jyväskylä: Gummerus Kirjapaino Oy. ISBN 951-762-829-3

Holzner, Steven. 2006. Design Patterns for Dummies. John Wiley & Sons Inc. ISBN: 9780471798545

Kansallisarkisto 2008. SÄHKE2-määräys. Määräys sähköisen asiakirjallisen tiedon käsittelystä, hallinnasta ja säilyttämisestä. Luettu 4.11.2018. https://www.arkisto.fi/uploads/normit/valtionhallinto/maarayksetjaohjeet/normiteksti_suomi.pdf

Kansallisarkisto 2012. SÄHKE2-sertifiointi. Luettu 4.11.2018. https://www.arkisto.fi/uploads/Viranomaisille/S%C3%A4hke2/S2_SERTIF_%20eAMS_jarjestelma_v1.1.pdf

Microsoft. Typescript in 5 minutes. n.d. Luettu 8.11.2018. http://www.typescript-lang.org/docs/handbook/typescript-in-5-minutes.html

Opensource.com. What is Docker? n.d. Luettu 8.11.2018. https://opensource.com/resources/what-docker

Oracle. 5.3 Packaging Web Archives. n.d. Luettu 8.11.2018. https://docs.oracle.com/javaee/7/tutorial/packaging003.htm

The Apache Software Foundation. Maven – Introduction. n.d. Luettu 8.11.2018. https://maven.apache.org/what-is-maven.html

Tutorialspoint. Spring MVC Framework. n.d. Luettu 4.11.2018. https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm

LIITTEET

Liite 1. Esimerkki toteutetusta datarakenteesta


```
[
    {
        "id": "3178496",
        "root": {
            "id": "root",
            "identification": "Roundabout",
            "name": "Roundabout",
            "description": null,
            "startDate": null,
            "endDate": null,
            "values": {},
            "children": [
                "312404518814"
            ]
        },
        "organizationFunctions": [
            {
                "id": "256095010235",
                "identification": "00",
                "name": "Tempest",
                "description": "00",
                "startDate": null,
                "endDate": null,
                "values": {},
                "children": [
                    "205998967396",
                    "460375676445"
                ]
            },
            {
                "id": "460375676445",
                "identification": "00",
                "name": "Feint",
                "description": "00",
                "startDate": null,
                "endDate": null,
                "values": {},
                "children": [
                    "428303872059"
                ]
            },
            {
                "id": "205998967396",
                "identification": "01",
                "name": "Evade",
                "description": "01",
                "startDate": null,
                "endDate": null,
                "values": {},
                "children": []
            },
```


(jatkuu)

```
            {
                "id": "312404518814",
                "identification": "00",
                "name": "SKILLS",
                "description": "01",
                "startDate": null,
                "endDate": null,
                "values": {},
                "children": [
                    "300919442323",
                    "256095010235"
                ]
            },
            {
                "id": "300919442323",
                "identification": "01",
                "name": "Punch",
                "description": "01",
                "startDate": null,
                "endDate": null,
                "values": {},
                "children": []
            }
        ],
        "processingSteps": [
            {
                "id": "428303872059",
                "identification": "ei määritelty",
                "name": "{\"fi\":\"\",\"en\":\"\",\"sv\":\"\"}",
                "description": "ei määritelty",
                "startDate": null,
                "endDate": null,
                "values": {},
                "children": [
                    "482055343610"
                ]
            }
        ],
        "documentTypes": [
            {
                "id": "482055343610",
                "identification": "Aloite",
                "name": "{\"fi\":\"\",\"en\":\"\",\"sv\":\"\"}",
                "description": "Aloite",
                "startDate": null,
                "endDate": null,
                "values": {}
            }
        ],
        "locks": [
            {
                "user": "niemivai",
                "entityId": "root"
            }
        ]
    }
]
```