SAVONIA

OPINNÄYTETYÖ - **YLEMPI AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

JÄRJESTELMÄINTEGRAATIO JA -TIEDONSIIRTO

TEKIJÄ/T: Erno Voutilainen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Teknologiaosaamisen johtamisen tutkinto-ohjelma	
Työn tekijä(t) Erno Voutilainen	
Työn nimi Järjestelmäintegraatio ja -tiedonsiirto	
Päiväys 16.05.2019	Sivumäärä/Liitteet 42/3
Ohjaaja(t) Lehtori Jussi Koistinen, Lehtori Keijo Kuosmanen, Tuntiopettaja Teija Tossavainen	
Toimeksiantaja/Yhteistyökumppani(t) Finnish Net Solutions Oy / Janne Huttunen	

Tiivistelmä

Tämä opinnäytetyö kohdistui kahteen tilaajayrityksen kehittämään verkkopohjaiseen, eri sukupolven terveydenhuollon toiminnanohjausjärjestelmään. Työn tilaajayrityksellä oli tahtotila siirtää vanhempaa järjestelmää käyttävät asiakkaat käyttämään uutta vastaavaa järjestelmää. Ennen kuin tämä olisi mahdollista, vanhassa järjestelmässä olleet kriittiset toiminnallisuudet piti saada siirrettyä vanhasta järjestelmästä uuteen. Lisäksi järjestelmien välillä piti toteuttaa asiakkaiden tietojensiirto. Tämän opinnäytetyön tavoitteena oli selvittää paras tapa toiminnallisuuksien siirtämiseen vanhasta järjestelmästä uuteen. Lisäksi tavoitteena oli selvittää kustannustehokkain tapa toteuttaa järjestelmien välinen tietojensiirto ja toteuttaa se yhden pilottiasiakkaan järjestelmästä.

Tietojensiirtoa varten kokeiltiin kahta kolmannen osapuolen tietojensiirtoalustaa ja vertailtiin testeistä saatuja tuloksia nykyiseen käsitykseen omatekoisen tietojensiirtoalustan toteutuksesta. Yrityksellä oli entuudestaan jonkin verran kokemusta omatekoisista tietojensiirtotyökaluista. Työn puitteissa tehdyn selvityksen perusteella tietojensiirtoalusta päätettiin rakentaa itse. Sen rakentamisen jälkeen kahden työhön liittyneen järjestelmän väliseen tietojensiirtoon tehtiin alustalle sopiva liitännäinen ja toteutettiin pilottiasiakkaan tietojensiirto vanhasta järjestelmästä uuteen.

Toiminnallisuuksien siirtoa varten toimintojen suoran uudelleentoteutuksen lisäksi mikropalveluarkkitehtuuri osoittautui hyväksi keinoksi irrottaa toiminnallisuutta vanhasta järjestelmästä ja tuoda se molempien järjestelmien saataville mikropalvelumuodossa. Työn puitteissa suunniteltiin ja toteutettiin tilausyhdyskäytävä, jonka kautta sekä vanha että uusi järjestelmä pystyivät tekemään tukku- ja laboratoriotilauksia eri toimittajille. Mikropalvelumuodon etu järjestelmien näkökulmasta oli, että tilaustoiminnot toimivat samalla tavalla toimittajasta riippumatta.

Avainsanat
Mikropalvelut, ETL, tietojensiirto

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Master's Degree Programme in Engineering Knowlege Management			
Author(s) Erno Voutilainen			
Title of Thesis Systems Integration and Data Migration			
Date	31 May 2019	Pages/Appendices	42/3
Supervisor(s) Mr. Jussi Koistinen, Senior Lecturer, Mr. Keijo Kuosmanen, Senior Lecturer, Mrs. Teija Tossavainen, Lecturer			
Client Organisation /Partners Finnish Net Solutions Oy / Janne Huttunen			

Abstract

This thesis centered around two web-based practice management systems from two different generations, developed by the Client Organisation. The Client Organisation wanted to move its customers using the older PMS to use the newer one. Before this would be possible, all critical functionalities and customer data that existed int the older system needed to be transferred over to the newer one. The goal of this thesis was to figure out the best way to move or reimplement missing functionalities from the older system to the new one. Another goal was to figure out the most cost-effective way to do the data migration between the systems and to make one actual migration with a pilot customer's data.

Two different third-party ETL-platforms were evaluated in the scope of this thesis. The evaluation results were compared to the Client Organisation's current understanding of the costs and what developing the ETL-platform internally would require. The Client Organisation did have some previous experience developing data migration scripts for the older PMS. As a result of the evaluation the Client Organisation decided to develop the required data migration tool internally. After the migration tool's core was ready, a suitable plugin for the data migration between these two PMS's was developed in the scope of this thesis and a pilot migration was done with the pilot customer's data.

In addition to the straight-out re-implementation of the missing PMS functionalities, microservices architecture proved out to be a good solution to extract existing functionalities and making them available for both systems as a webservice. In the scope of this thesis an order gateway microservice was planned and implemented, through which both systems could send wholesaler and laboratory orders in a unified way to different providers. A clear advantage of the microservices apporach from the perspective of the PMS's was that the ordering did function in a coherent way regardless of all the provider API's being different from each other.

Keywords
Microservices, ETL, data migration

ESIPUHE

Tämä opinnäytetyö tehtiin Finnish Net Solutions Oy:lle syksyn 2016 ja kevään 2019 välisenä aikana. Kahden ja puolen vuoden uurastus on nyt saatu päätökseen ja tämä esipuhe on itseasiassa viimeinen varsinainen kirjoitusosuus, minkä tähän työhön lisään.

Haluan kiittää Janne Huttusta työn tilaajan puolelta sekä Jussi Koistista, Keijo Kuosmasta ja Teija Tossavaista Savonian puolelta työn ohjaamisesta ja antamastanne tuesta. Prosessi on ollut pitkä ja teiltä saamanne neuvot ovat aina ohjanneet työtä oikeaan suuntaan.

Lisäksi haluan kiittää vaimoani ja lapsiani suurimmasta mahdollisesta tuesta ja kärsivällisyydestä tämän projektin osalta. Ilman uhraamaanne aikaa ja muita voimavaroja sekä satunnaista takapuoleen potkimista tämä työ ei olisi koskaan valmistunut. Tämän työn valmistumisen myötä on minun vuoroni antaa jakamaton huomioni teille.

Kuopiossa 26.05.2019
Erno Voutilainen

SISÄLTÖ

1 JOHDANTO

Tässä opinnäytetyössä käsitellään järjestelmäintegraatiota toimintojen siirtämisen välineenä, sekä järjestelmien välistä tiedonsiirtoa. Tavoitteena on selvittää keinoja vanhassa järjestelmässä olevien toimintojen ja ominaisuuksien siirtämiseksi uuteen järjstelmään, mistä toimintoja puuttui. Lisäksi tavoitteena on selvittää kustannustehokkain tapa siirtää tiedot vanhasta järjestelmästä uuteen. Molempiin tavoitteisiin sisältyy esimerkkitoteutuksen teko parhaiksi valikoituneita menetelmiä käyttämällä. Työ tehtiin Finnish Net Solutions Oy:lle syksyn 2016 ja kevän 2019 välisenä aikana.

Tämä raportti etenee pääpiirteittäin siinä järjestyksessä, kuin missä itse käytännön työ tehtiin. Toisessa luvussa esitellään tarkemmin työn tilaajayritys sekä työn tarve, tavoitteet ja haasteet. Kolmannessa luvussa käsitellään tietojensiirtotyökalun esiselvitystä, minkä tavoitteena oli selvittää, kannattaisiko yrityksen ostaa valmis kolmannen osapuolen tietojensiirtoalusta vai toteuttaa oma alusta tulevia tietojensiirtotarpeita varten. Neljännessä luvussa käsitellään ohjelmistomoduulin toteuttamista valitulle alustalle, jonka avulla tietojensiirto vanhasta järjestelmästä uuteen tapahtui. Viidennessä luvussa käsitellään toimintojen siirtämistä järjestelmästä toiseen mikropalveluarkkitehtuuria hyväksikäyttämällä sekä tilaustoimintoihin keskittyvän mikropalvelun toteuttamisen vaiheita. Lopuksi kuudenneesa luvussa käydään läpi työn tulokset ja tehdään yhteenveto koko työstä.

Taulukossa 1 on esitetty työssä esiintyvät lyhenteet ja niiden määritelmät.

Taulukko 1 Lyhenteet ja määritelmät

Lyhenne	Määritelmä
ETL	Extract, Transform, Load (suom. Pura, Muunna, Lataa)
API	Application Programmable Interface (suom. Ohjelmistorajapinta)
Alpha	Vanha järjestelmä, jota tietojensiirron osuus koski
Bravo	Uusi järjestelmä, jota tietojensiirron osuus koski
TOS-DI	Talend Open Studio for Data Integration. Kolmannen osapuolen integraatioalusta.
MVC	Model-View-Controller -arkkitehtuuri

2 TYÖN TAUSTAT

Tässä luvussa esitellään lyhyesti työn tilaajayritys Finnish Net Solutions Oy, taustatekijöitä työn tilaamiseen liittyen ja työhön liittyneitä haasteita. Lisäksi käydään läpi työn edistymisen sekä työn sisällön muuttuminen ja siihen vaikuttaneet tekijät.

2.1 Finnish Net Solutions Oy

Finnish Net Solutions Oy (FNS) on vuonna 2001 perustettu ohjelmistoalan yritys. Tämän raportin kirjoitushetkellä (keväällä 2019) yrityksessä työskenteli yli 100 työntekijää. Yrityksen alkuperäinen liiketoiminta koostui nettisivujen teosta yksityishenkilöille, yrityksille ja järjestöille. Vuosien varrella yrityksen tekemien liiketoimintakauppojen myötä se on keskittynyt tuottamaan ohjelmistoja ihmis- ja eläinterveydenhuollon tarpeisiin sekä koulutuspalveluja. Yritys tuottaa myös asiakkaiden tarpeiden mukaan räätälöityjä järjestelmiä.

Muutaman viimeisen vuoden aikana yrityksen toiminta on alkanut vahvasti kansainvälistymään uuden, erityisesti Euroopan markkinoita silmällä pitäen kehitetyn eläinklinikoiden toiminnanohjausjärjestelmän rantauduttua useisiin eri Euroopan maihin. Yrityksen palveluksessa toimii nykyään useita muista kansallisuuksista peräisin oleviä henkilöitä, jotta se pystyy paremmin palvelemaan eri Euroopan maita paremmin. Näistä syistä Englannin kieli on noussut vahvasti yrityksen toiseksi sisäiseksi viestintäkieleksi. Kansainvälistymistä yhä uusille markkinoille vauhdittaa tulevaisuudessa myös Pamoja Capital -sijoitusyhtiön osallistuminen yrityksen toimintaan sen hankkiman osake-enemmistön myötä FNS:n emoyhtiöstä Three Plus Group Oy:stä.

2.2 Työn tarve ja tavoitteet

Tämä opinnäytetyö laitettiin alulle vuoden 2016 marraskuussa pidetyllä aloituspalaverilla. Työ kohdistuu kahteen eri sukupolven verkkopohjaiseen terveydenhuollon toiminnanohjausjärjestelmään. Vanhempi järjestelmä oli työn aloitusvaiheessa palvellut asiakkaitaan jo yli 10 vuotta, joka on kyseisen tyyppiselle ohjelmistolle kunnioitettava ikä. Vanha järjestelmä oli aiemmin todettu liian kankeaksi taipumaan uusille markkinoille ja uutta järjestelmää päätettiin alkaa kehittää vanhan ohjelmiston rinnalla vuoden 2014 ja 2015 taitteessa. Kutsun tässä raportissa tästä eteenpäin vanhaa järjestelmää koodinimellä *Alpha* ja uutta järjestelmää koodinimellä *Bravo*.

Kahden tai useamman samankaltaisen järjestelmän ylläpito tulisi pitkässä juoksussa olemaan kallista, joten alusta asti oli selvää, että sopivassa tilanteessa Alpha-järjestelmän käyttäjät tulisi siirtää käyttämään Bravo-järjestelmää. Tässä siirtymässä tulisi olemaan kaksi ratkaisevaa tekjiää, jotta asiakkaiden siirtymä järjestelmien välillä olisi mahdollista:

- Järjestelmistä pitäisi löytyä samat tai toisiaan vastaavat toiminnallisuudet (vähintään kriittisiltä osin)
- Alpha-järjestelmistä pitäisi pystyä jollain keinoin siirtämään tiedot Bravoon

Nämä kaksi ennakkovaatimusta asettivat tavoitteet tälle opinnäytetyölle. Työn tavoitteena on

- selvittää Alpha- ja Bravo-järjestelmien välisiä toiminnallisuuksien eroavuuksia, sekä suunnitella, miten ne voidaan tehokkaimmin siirtää tai toteuttaa Bravo-järjestelmään
- selvittää kustannustehokkain tapa tietojensiirtotyökalun toteuttamiseen sekä toteuttaa työkalu, jolla Alpha-järjestelmän tiedot saadaan siirrettyä Bravo-järjestelmään

Tilaajayrityksellä ei ollut aiempaa kokemusta vastaavista tilanteista, missä toiminnallisuutta haluttiin siirtää järjestelmästä toiseen. Sillä oli kuitenkin joitain toteutuksia pienistä yleiskäyttöisistä palveluista, joita saattoi käyttää mikä järjestelmä tahansa. Tämä sama tekniikka osoittautui hyväksi tavaksi siirtää toiminnallisuutta järjestelmästä toiseen.

Tilaajayrityksellä oli entuudestaan jonkin verran kokemusta järjestelmien välisistä tietojensiirroista. Niitä oltiin tehty Alpha-järjestelmään sitä edeltävistä sekä kilpailijoiden järjestelmistä. Bravo-järjestelmään ei ollut tässä vaiheessa vielä tehty vakiintunutta toteutusta tietojensiirtotyökalusta, joka tultaisiin ehdottomasti tarvitsemaan. Työkalujen toteutusmallia haluttiin myös parantaa, koska vanha malli ei tulisi skaalautumaan hyvin mahdollisten lähdejärjestelmien moninkertaistuessa.

2.3 Haasteet

Työssä haastavaa oli se, että minulla ei ollut työn aloitusvaiheessa vielä selvillä, mitä muita etenkään kolmannen osapuolen keinoja tietojensiirtotyökalun toteutukseen voisi käyttää. Tätä tietoa oli aluksi vaikea löytää. Kun törmäsin termiin ETL (Extract, Transform, Load), alkoi sopivia kolmannen osapuolen työkaluja ja aiheeseen liittyvää kirjallisuutta löytyä. Haasteeksi jäi tämän jälkeen enää opiskella ko. työkalujen käyttöä, josta lisää luvussa 3.

Toiminnallisuuksien siirtämisessä haastavaa oli aluksi keksiä muita keinoja toimintojen siirtämiseen, kuin toimintojen uudelleen suunnitteleminen ja toteuttaminen uuteen järjestelmään. Pitkään olin sitä mieltä, että kyseinen tavoite pitäisi määritellää uudestaan ja keskittyä mm. toiminnallisten- ja teknisten määrittelyjen dokumentointiin. Toiminnallisuuksien siirtämisen haasteellisuuteen liittyy myös se, että järjestelmät on toteutettu eri ohjelmointikielillä, joten toimintojen siirto suoraan sellaisenaan ei tullut kysymykseen. Lopulta ratkaisuksi valikoitunut mikropalveluarkkitehtuuri tarjosi siinä mielessä uuden haasteen, että en ollut aiemmin toteuttanut (tai ollut osaltani toteuttamassa) vastaavalla menetelmällä tehtyä ratkaisua. Mikropalveluista lisää luvussa 5.

2.4 Työn eteneminen ja muuntuminen

Käytännön projektityö tapahtui kolmessa eri vaiheessa ja vaiheiden välillä ehti kulua paljon aikaa. Ensimmmäisessä vaiheessa tein vuoden 2016 loppupuolella ja vuoden 2017 aikana selvitys- ja testaustyötä kolmannen osapuolen valmiilla tietojensiirtoalustoilla ja selvitin, kuinka hyvin ne soveltuisi-

vat tietojensiirtotyökaluksi ja minkälaisia kustannuksia ko. työkalujen hankkiminen aiheuttaisi. Vertasin kahden ulkopuolisen eri alustan välisiä kustannuksia ja aiempiin kokemuksiin perustuvaa arviota itse tehdyn alustan toteutuskustannuksista. Tietojensiirtotyökalun esiselvityksestä lisää luvussa 3.

Erilaisista syistä johtuen en pystynyt esiselvityksen jälkeen enää suoraan keskittymään tietojensiirtotyökalun toteuttamiseen, vaan se toteutettiin muualla yrityksessä. Työkalu toteutettiin siten, että siihen voidaan toteuttaa kunkin asiakkaan lähdejärjestelmää vastaava liitännäinen. Työn toisessa vaiheessa pääsin keväällä 2018 toteuttamaan tämän työkalun päälle liitännäisen, joka vastasi Alpha-järjestelmän tietojen siirtämisestä uuteen järjestelmään. Lisää tästä prosessista luvussa 4.

Samaan aikaan keväällä 2018 ilmeni sopiva keino toiminnallisuuksien siirtämiseksi vanhasta järjestelmästä uuteen. Mikropalveluarkkitehtuuri tai yleisemmin mikropalvelut ovat tämän päivän muotisana ja kyseinen malli sopii hyvin toiminnallisuuksien jakamiseen useiden järjestelmien kesken. Työn kolmantena vaiheena aloimme kollegani kanssa suunnitella ja toteuttaa erilaisiin tilaustoimintoihin keskittyvää mikropalvelua. Palvelun avulla vanhaan järjestelmään suoraan tehdyt toiminnallisuudet voitaisiin pakata yleiskäyttöiseksi mikropalveluksi, jota voisi sen jälkeen käyttää yhtä aikaa sekä vanha että uusi järjestelmä. Mikropalvelusta lisää luvussa 5.

2.5 Huomioita

Tässä raportissa esiintyvissä kuvakaappauksissa ja teksteissä mahdollisesti esiintyvät web-osoitteet ovat anonymisoitu, eivätkä ne vastaa todellisuudessa käytettyjä osoitteita. Kaikki käytännön työvaiheet on tehty pääasiassa Mac-tietokoneella, joten tämän raportin näkökulmasta MacOS on oletuskäyttöjärjestelmä. Raportissa mainitaan erikseen, jos käytetään jotain muuta käyttöjärjestelmää.

3 TIETOJENSIIRRON ESISELVITYS

Tässä luvussa käsitellään tietojensiirtotyökalun ennakkoselvitystä. Luvun aluksi käydään läpi ennakkoselvityksen taustatietoja ja tarvetta. Taustatietojen jälkeen valitaan kaksi potentiaalista tietojensiirtoalustaa tarkempaan tarkasteluun ja määritellään testitoteutuksen sisältö, joka tullaan tekemään valituilla alustoilla. Tämän jälkeen käydään läpi testitoteutusten teko molemmille alustoille ja luvun lopuksi tehdään yhteenveto luvusta.

Olen suomentanut ohjelmistoissa käytetyt termit tätä raporttia varten. Viittaan alkuperäisiin termeihin sulkeilla. Käytin kaikissa tietojensiirtotesteissä lähdeaineistona testidataa Alpha-järjestelmän testiversiosta.

3.1 Taustatietoja

Kuten luvussa 2.2 kävi ilmi, toisena tämän työn tavoitteena on selvittää kustannustehokkain tapa toteuttaa tietojensiirtotyökalu, jolla vanhan Alpha-järjestelmän asiakastiedot saataisiin siirrettyä uuteen Bravo-järjestelmään.

Toiminnanohjausjärjestelmistä löytyvät kaikki asiakkaiden tarvitsemat, bisneskriittiset tiedot, kuten asiakasrekisterit sekä potilas- ja hoitotiedot, joita he tarvitsevat päivittäisessä työssään. Jotta he olisivat ylipäätään valmiita vaihtamaan järjestelmää, heille on taattava jokin keino siirtää nykyisessä järjestelmässä olevat tiedot uuteen järjestelmään. Joissain muissa järjestelmissä tämä on yleensä järjestetty niin, että asiakas voi tuoda joitain tietoja sisään itse esim. Excel-taulukkomuodossa.

Nykyiset yrityksen asiakkaat ovat kuitenkin aiemmin tottuneet siihen, että heidän siirtyessään käyttämään uutta järjestelmää heidän vanhassa järjestelmässään olevat tiedot on siirretty uuteen heidän puolestaan. Vastaavanlaisiin toiveisiin oltiin jo törmätty uusilla markkinoilla Bravo-järjestelmän osalta, joten hyvän asiakaspalvelutason säilyttämiseksi Bravo-järjestelmään haluttiin saada integroitua tietojensiirtotyökalu.

Yrityksellä oli entuudestaan kokemusta tällaisista tietojensiirroista Alpha-järjestelmään. Alphaan pystyttiin tekemään tietojensiirtoja muutamista kilpailijoiden- sekä omista vanhemmista järjestelmistä. Nämä tietojensiirtotyökalut olivat käsin kasattuja skriptejä, mitä pahimmassa tapauksessa monistettiin oma kappale jokaiselle asiakkaalle.

On itsestään selvää, että aiempien tietojensiirtotyökalujen toteutusmalli ei tulisi palvelemaan Bravo-järjestelmän tarpeita riittävän hyvin. Uusille markkinoille lähdettäessä on selvää, että mahdollisten lähdejärjestelmien määrä tulee moninkertaistumaan aiempaan verrattuna. Tästä syystä uuden tietojensiirtotyökalun pitäisi olla paremmin skaalautuva.

Samalla kun uutta tietojensiirtotyökalua lähdettään kehittämään, halutaan selvittää mahdollisuutta ottaa käyttöön jokin kolmannen osapuolen tietojensiirtoalusta, jonka avulla siirtotyökalut voitaisiin

rakentaa. Tavoitteena on tutustua muutamaan potentiaaliseen vaihtoehtoon ja vertailla niiden mahdollisia kustannuksia itse tehtyyn ratkaisuun.

3.2 Toteutustavan suunnittelu

Tietojensiirtotyökalun esiselvitys alkoi varteenotettavien kolmannen osapuolen alustojen selvittämisellä. Oikea termi, jolla tämän tyyppisiä ratkaisuja alkoi löytyä, oli ETL (Extract, Transform, Load). Tietojensiirron ja järjestelmäintegraatioiden tarve on globaali ilmiö, joten arvatenkin jotkin ohjelmistotalot ovat tarttuneet tähän mahdollisuuteen ja rakentaneet yleiskäyttöisiä alustoja, joilla pitäisi ainakin periaatteessa pystyä ratkaisemaan kaikki integraatiotarpeet.

Oikean termin löydyttyä ETL-ratkaisuja tarjoavia yrityksiä ja tuotteita alkoi löytyä runsaasti. Ensimmäinen haaste oli sopivalta tuntuvien tuotteiden valitseminen. Opinnäytetyöprojektin mittakaavaan suhteutettuna vertailuun ei ollut mahdollista valita kovin montaa kandidaattia. Katsoin aluksi läpi joitain ilmaisia vaihtoehtoja, mutta monet niistä olivat joko ominaisuuksiltaan puutteellisia tai liian vaikeita oppia nopeasti. (Software Testing Help, 2019)

Jätin vertailusta suosiolla pois myös isojen ohjelmistotalojen, kuten Microsoftin, Oraclen, IBM:n ja muiden vastaavien ratkaisut, koska ne olivat lähtökohtaisesti todella kalliita ja Bravo-ohjelmiston tarpeisiin nähden liian raskaita. Aikani vaihtoehtoja vertailtuani päätin ottaa lopulliseen lähempään tarkasteluun Talend Open Studio for Data Integration- sekä CloverDX-ohjelmistot. (Talend Inc.)

Vertailujen tekohetkellä molemmista ohjelmistosta on saatavilla kokonaan ilmainen sekä maksullinen versio. Molemmista yrityksistä oltiin minuun suoraan yhteydessä kokeilujeni aikana ja sain yhteyshenkilöiltä mm. lisätietoja ja hintatiedot molempien järjestelmien maksullisista versioista. Kustannuksista lisää luvussa 3.4.4. CloverDX-ohjelmistosta ei tämän raportin kirjoitushetkellä enää ole saatavilla kokonaan ilmaista versiota, vaan ainoastaan ilmainen kokeilu maksullisesta versiosta. (Javlin Ltd.)

3.2.1 Vertailutoteutuksen määrittely

Jotta pystyisin vertailemaan ohjelmistoja ja itse tehtyä toteutusta riittävällä tasolla, päätin tehdä hyvin rajatun toteutuksen tietojensiirtotyökalusta Bravo-järjestelmään käyttäen molempia valitsemiani kolmannen osapuolen vaihtoehtoja. Itse rakennetun tietojensiirtotyökalun rakennuskustannukset on helppo arvioida riittävällä tasolla aiempiin kokemuksiin nojaten. Tästä syystä päätin jättää sen rakentamatta ja verrata kahden ulkopuolisen järjestelmän kustannuksia arvioon oman toteutuksen kustannuksista.

Valitsin tietojensiirron lähdeaineistoksi toimenpidelistauksen. Toimenpidelistaus on molemmissa järjestelmissä selkeästi rajattu ja itsenäinen tietoalue, joten siitä on helppo lähteä liikkeelle. Molemmissa toteutuksissa toistuvat seuraavat vaiheet:

- luetaan lähdeaineisto MySQL-tietokannasta
- muokataan lähdetiedoista JSON-tietueita
- lähetetään JSON-tietueet HTTP POST-pyynnöillä Bravo-järjestelmään REST API:a pitkin
- kerätään virheellisistä ja onnistuneista lähetyksistä lokitiedot tekstitiedostoon.

Avasin tietojensiirtotestejä varten Bravo-järjestelmästä oman testiversion, jota vasten tein kaikki tähän liittyneet testiajot. Tyhjensin järjestelmän tietokantaa aina tarvittaessa ajojen välillä. Kaikissa kolmessa toteutuksessa seurattavia asioita ovat

- kuinka kauan toimenpidelistauksen siirron rakentamiseen kuluu aikaa
- kuinka nopeasti siirtotyökalu suoriutuu siirron tekemisestä.

3.3 Tiedonsiirtoalustojen testitoteutukset

Suunnitelman laatimisen jälkeen aloitin testitoteutuksen tekemisen kummallakin valitsemistani alustoista. Tein ensin toteutuksen Talend Open Studiolla ja sitten CloverDX:llä. Huomasin selvitystyön edetessä, että molemmat valitsemista alustoistani ovat hyvin paljon toisiaan muistuttavia ja niistä löytyy joitain samoja perusasioita, mutta niistä saatetaan käyttää eri nimityksiä. Näiden peruskäsitteiden tunteminen helpotti siirtotyökalujen edistämistä ja niihin tutustuminen ja opettelu vei huomattavan paljon aikaa. Seuraavaksi molemmissa alustoissa käytetyt peruskäsitteet.

3.3.1 Tiedonsiirtoalustojen peruskäsitteet

Molemmilla alustoilla tehdyt siirtokokonaisuudet muodostuvat metatiedoista ja yhteyksistä (Metadata / Connections), työmalleista ja graafeista (Job Designs / Graphs) sekä komponenteista (Component). Näiden lisäksi molemmista alustoista löytyy kummallekkin erityisiä ominaisuuksia, joita en lähtenyt tämän työn puitteissa erityisesti käymään läpi.

TOS-DI:n **työmallit** (vrt. CloverDX:n **graafit**) ovat ylimmän tason loogisia kokonaisuuksia, jotka voivat joko itsenäisesti tai osittain kattaa koko tiedonsiirtotyökalun. Malleja on mahdollista ryhmitellä kansioiden avulla ja niitä on mahdollista linkittää toisiinsa esim. niin, että ylätason työ tuo tietoja sisään alatason työhön tai ottaa vastaan alatason prosessoimaa tietoa. Tämän testauksen puitteissa minun ei tarvinnut syventyä erillisten töiden linkittämiseen, mutta mahdollista myöhempää käyttöä varten tämä on hyvä tietää. Loin molempiin alustoihin yhden työmallin, joka hoiti koko tietojensiirron, joskin se tehtiinkin vain hyvin rajatusta osasta koko järjestelmää. Jos koko järjestelmän tietojensiirto rakennettaisiin näillä alustoilla, yksi päätason malli vastaisi luultavasti kaikkien erillisten mallien ajosta.

Molemmista alustoista löytyy kattava valikoima erilaisia **komponentteja**, joista työmallit koostuvat. Komponenteilla voi esim. tehdä tietokantahakuja, lukea tiedostoja, muokata ja yhdistää datavirtoja, tehdä web-pyyntöjä jne. Komponenteilla voi myös ajaa omaa Java-koodia, jos valikoimasta ei löydy

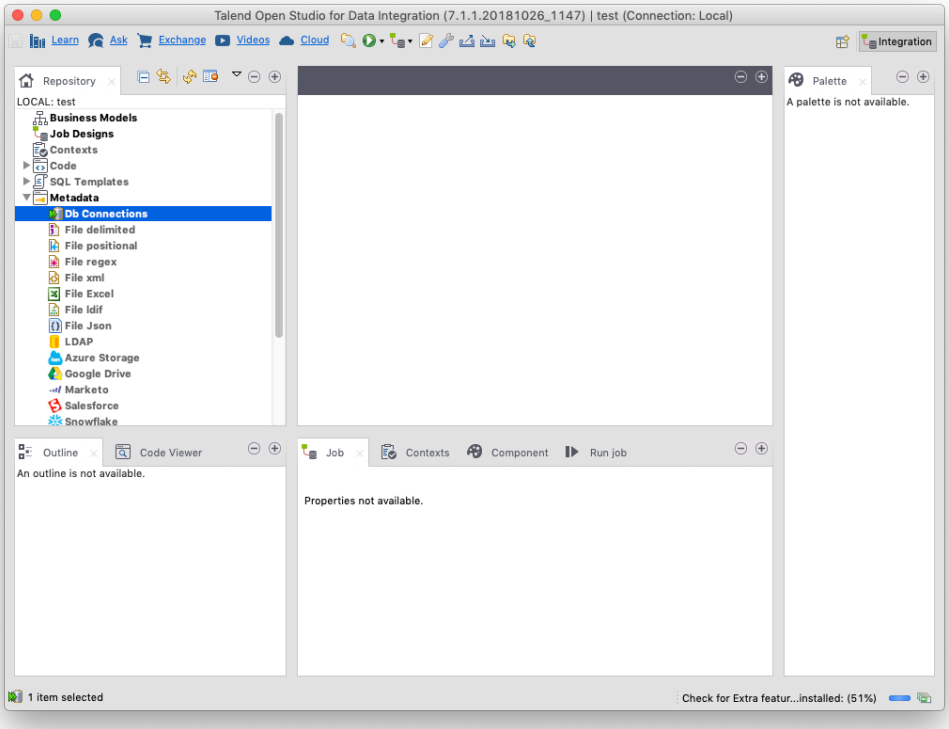
tarpeeseen sopivaa komponenttia. Tämän testin puitteissa minun ei tarvinnut turvautua omaan koodiin vaan sain testin tehtyä valmiita komponentteja käyttämällä.

Vain TOS-DI:stä löytyvät **kontekstit** ovat käytännössä kokoelmia muuttujataulukoista, joihin on mahdollista tallentaa avain/arvo pareja. Konteksteihin on hyvä tallentaa mm. tuotanto- ja kehitysympäristöjä vastaavia arvoja esim. tietokantayhteyksistä, verkkopalvelujen osoitteista jne. Kontekstit tulevat tarpeeseen varsinkin silloin, jos useampi työmallin komponentti tarvitsee saman kontekstin arvoja. Kulloinkin käytettävä konteksti valitaan aina ajoa aloitettaessa.

TOS-DI:n **metatietoihin** (vrt. CloverDX:n **yhteyksiin**) on mahdollista tallentaa koko projektille yhteisiä tietolähteitä, kuten tietokantayhteyksiä, erilaisia tiedostoja ja verkkopalveluita. Tämän testin puitteissa käytin metatiedoista ja yhteyksistä vain tietokantayhteyttä.

3.3.2 Talend Open Studio for Data Integration

Talend Open Studio for Data Integration (TOS-DI) on ensimmäinen valitsemistani alustoista, jolla lähdin kokeilemaan siirtotyökalun rakentamista. Ohjelmisto on Java-pohjainen, Eclipse-ohjelmistoalustan päälle kehitetty ja sitä voi käyttää Windows- ja MacOS-käyttöjärjestelmillä. Ohjelmisto asennetaan tietokoneelle samalla tavalla kuin mikä tahansa muukin ohjelma. Asennuksen ja alkumääritysten jälkeen pääsin lisäämään uuden projektin. (Weinstein;ym., 2013)

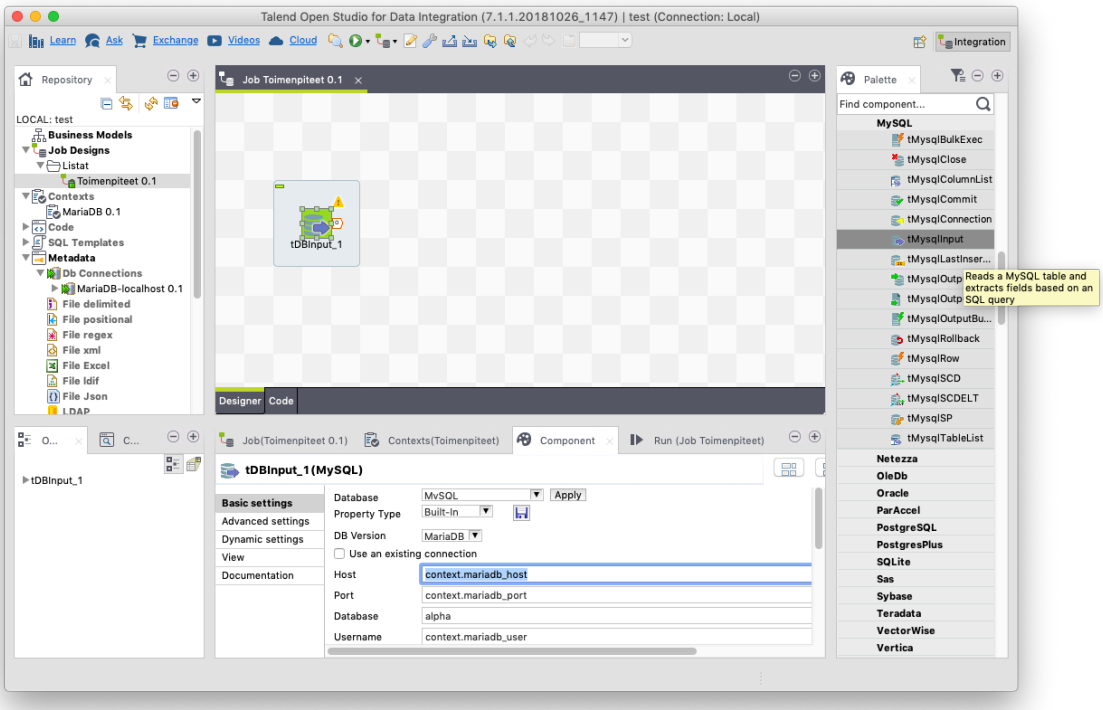


Kuvio 1. TOS-DI projektinäkymä

3.3.2.1 Siirtotyökalun rakentaminen

Ensimmäisenä vaiheena lisäsin projektin metatietoihin paikallisen työasemani MySQL-tietokantayhteyden. Tätä yhteyttä tarvitaan lähdeaineiston lukuun, koska Alpha-järjestelmä käyttää ko. tietokantaa. Yhteyden määrittämisen yhteydessä loin myös MySQL-kontekstin, jonne määritin kehitysympäristön palvelinasetukset.

Seuraavaksi loin työmallien ryhmittelyä varten työmalliosioon uuden kansion ”Listat” ja sen alle uuden työmallin ”Toimenpiteet”. Työmallin avaamalla avautuu keskelle ruutua kanvas, jolle itse työkalua aletaan rakentaa. Lähdeaineiston lukuun MySQL-tietokannasta käytin komponenttia "tMySQLInput". Komponentit löytyivät kehitysympäristön komponenttipaletista hakemalla, josta pystyin raahaamaan haluamani komponentit kanvakselle.

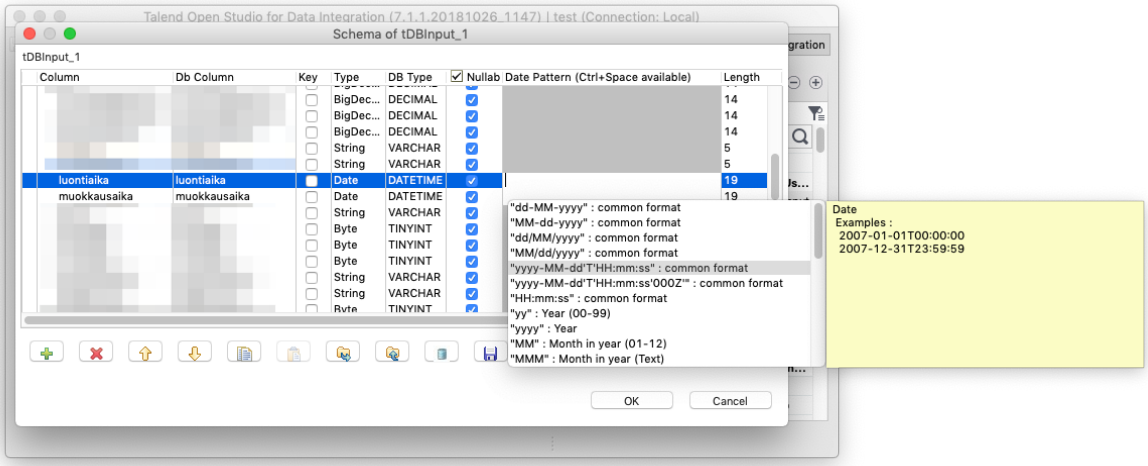


Kuvio 2. TOS-DI komponenttien lisääminen

Komponentin lisäyksen jälkeen pääsin tarkemmin määrittämään komponentin asetukset, kuten käytettävän tietokantayhteyden ja kyselyn, jonka halusin komponentilla suorittaa. Käytin tietokanta-asetuksiin aiemmin luomaani kontekstia. Komponentin lisäyksen jälkeen komponentin yläkulmassa kanvaksella näkyi vielä varoituskolmio. Kun sen päälle vei hiiren, komponentti listasi suoraan kaikki jäljellä olevat ongelmat, ennen kuin komponentti olisi toimintavalmis. Näitä ongelmia olivat

- skeema ei vastaa kyselyä
- skeemaa ei ole vielä määritelty
- tähän komponenttiin tulisi liittää ulosvienti.

Näiden virheilmoitusten perusteella määritin seuraavaksi lähdekannasta halutun taulun ja SQL-kyselyn. Kun kyselyyn laittoi kaikki taulun sarakkeet mukaan ja käytti toimintoa ”arvaa skeema”, ohjelma haki kyselyllä tietokannan rakenteen, parsi sen tietotyypit Javan tietotyyppejä vastaaviksi ja kasasi niistä itselleen sopivan skeeman. TOS-DI skeema on siis tässä suhteessa eri asia kuin tietokannan skeema. Skeeman arvaus ei osannut jostain syystä suoraan arvata oikeaa päivämääräformaattia, vaan se piti määrittää skeemaan käsin. Oikean päivämääräformaatin pystyi valitsemaan komponentin asetuksista. Tämän jälkeen virheviestit hävisivät ja kyseinen komponentti oli valmis liitettäväksi seuraavaan komponenttiin.



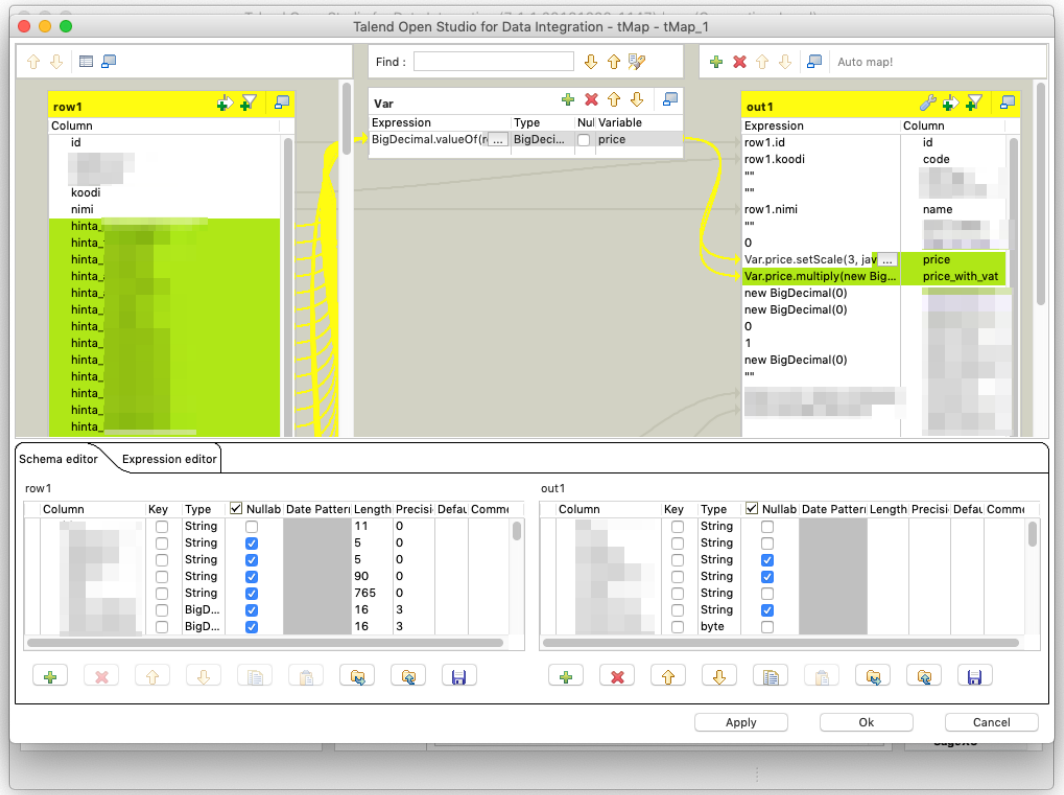
Kuvio 3. Skeeman aikaleimaformaatin määritys

3.3.2.2 Täsmäytyskomponentti

Seuraavaksi minun piti muodostaa sisään luettavista toimenpidetiedoista Bravo-järjestelmän kanssa yhteensopiva JSON-tietue, jonka lähettäisin siihen REST API:a pitkin. JSON-tietueen kenttien muodostamisen helpottamiseksi otin Bravo-järjestelmän toimenpiteiden säilytystaulusta kopion ja lisäsin toisen MySQL-komponentin samalla tavalla kuin edellisessä vaiheessa.

Tämän jälkeen lisäsin kanvakselle tMap-komponentin, johon yhdistin molemmat MySQL-komponentit: "Alpha-toimenpiteet" pääsisääntuloksi ja "Bravo-toimenpiteet" täsmäytyssisääntuloksi (Lookup input). tMap-komponentti vastasi näistä kahdesta lähteestä tulevien tietokenttien täsmäytyksistä, eli sen määrittämisestä, mikä Alpha-järjestelmän tietokenttä vastasi Bravo-järjestelmän tietokenttää. Näistä kahdesta tietoviirrasta tMap-komponentti muodosti yhden ulostulovirran, jonka muoto voitiin määrittää komponentin hallinnasta.

Seuraavaksi määritin täsmäytyksen tuloksena lähtevän tietovirran. Kopioin kaikki Bravo-järjestelmästä tulevat kentät skeemaeditorissa ja liitin ne ulostuloon. Tämän jälkeen poistin ylimääräiset tietokentät, joita sisääntuonnissa ei tarvittaisi. Skeeman määrittämisen jälkeen yhdistin Alpha-järjestelmän kentät Bravo-järjestelmän kenttiin.



Kuvio 4. Täsmäytyskomponentin määritys

Alpha-järjestelmässä toimenpidelistaus on rakennettu siten, että hinnat voidaan koostaa monesta eri osahinnasta. Osahinnat ja niiden summa tallennetaan erikseen tietokantaan. Osahintojen kohdalla halusin kokeilla tMap-komponentin muuttujaominaisuutta ja laskea sen avulla osahinnat muttujaan. Lisäsin BigDecimal-tyyppisen hintamuuttujan ”price” ja laskin sen operaatiossa Java-koodia käyttämällä osahintojen desimaaliarvot yhteen. Lopputuloksena syntyneeseen hintamuttujaan viitattiin ulostulopuolella syntaksilla *Var.price*.

Bravo-järjestelmä tallentaa hinnan sekä verollisessa että verottomassa muodossa, kun Alpha-järjestelmä tallentaa vain verottomassa. Bravo-järjestelmä haluaa verottoman hinnan kolmella desimaalilla. Sain pyöristettyä ja rajattua verottoman hinnan kolmeen desimaaliin ja laskettua verollisen hinnan hintamuuttujasta setScale-funktiolla. Verollinen summa tallennetaan kahden desimaalin tarkkuudella.

3.3.2.3 JSON, REST- ja lokikomponentit

Tietokenttien täsmäytyksen jälkeen seuraava vaihe oli muuntaa muodostettu uusi, Bravo-järjestelmän kanssa yhteensopiva skeema JSON-muotoon, jotta se voitaisiin lähettää REST-rajapintaa pitkin ko. järjestelmään. TOS-DI:n komponenttivalikoimasta löytyy tWriteJSONField-komponentti tätä tarkoitusta varten.

Komponentin lisäyksen jälkeen tMap-komponentissa määritetty ulostulo piti viedä JSON-komponentin sisääntuloksi. Koska JSON-rakenne oli jo määritetty valmiiksi tMap-komponentissa, pystyin määrittämään JSON-tietueen rakenteen suoraan sisääntulosta. Ainoaksi asetukseksi riitti, että JSON-tietueista poistettiin sinne vakiona tuleva juurielementti.

JSON-rakenteen muodostamisen jälkeen seuraava vaihe oli lähettää se REST-rajapintaa pitkin Bravo-järjestelmään. Tätä tarkoitusta varten komponenttivalikoimasta löytyy tRESTClient-komponentti. Toin komponentille sisääntuloksi JSON-komponentin ulostulon ja määritin perusasetuksiin Bravo-testijärjestelmän rajapintaosoitteen, HTTP-metodin (POST) ja sisältötyypin (JSON).

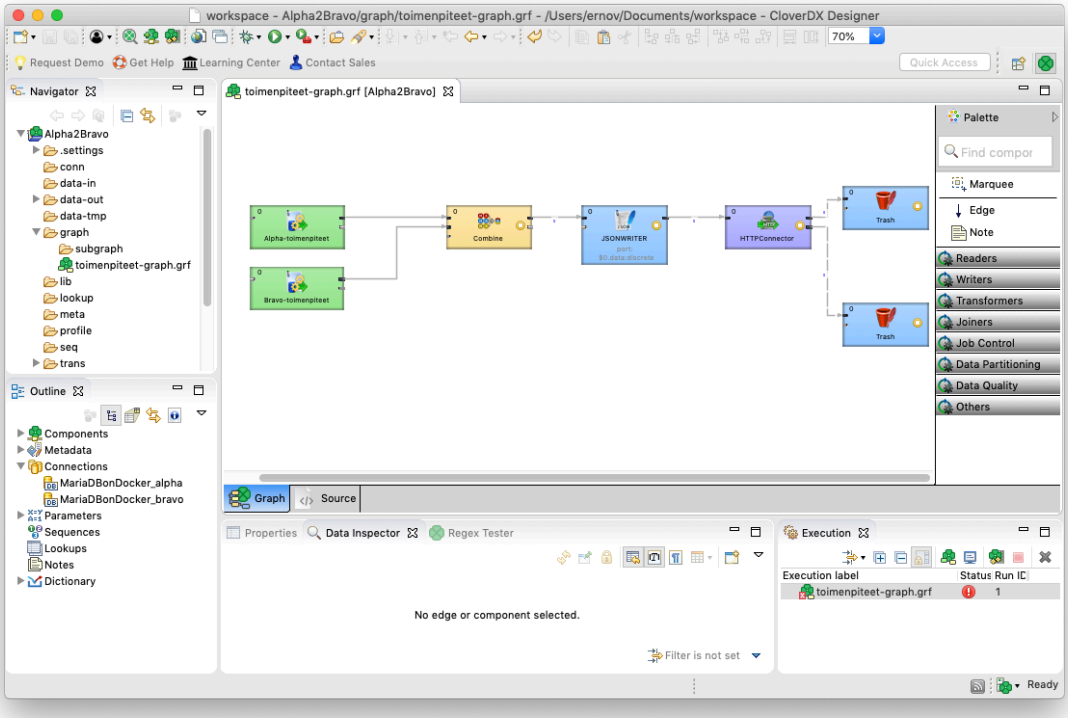
Komponentin lisäasetuksista kytkin päälle rajapinnan vastausten vastaanottamisen, jotta pystyin ajojen jälkeen näkemään rajapinnan vastaukset lokitiedostosta. Lisäksi kytkin päälle muita elämää helpottavia asetuksia, kuten uudelleenohjausten seurannan, tietotyyppien muunnokset tekstiksi jne. Näiden lisäksi lisäasetuksista sain syötettyä rajapintapyynnöissä tarvittavan tunnisteotsikon (Authorization Header) sekä sisältötyypin.

Lopuksi lisäsin tRESTClient-komponentin ulostuloon kaksi tLogRow-komponenttia, mitkä ottavat vastaan rajapintapyyntöjen vastaukset. Toinen komponenteista tallentaa onnistuneet ja toinen epäonnistuneet pyynnöt. tRESTClient-komponentin asetuksissa määritetään tietorakenteet sisääntulevalle tiedolle, rajapinnan vastauksille ja virheviesteille. Sisääntuleva rakenne on sama kuin JSON-komponentin ulostulo (eli yksi tekstikenttä). Onnistuneiden- ja epäonnistuneiden vastausten rakenteen määritin rajapinnan palauttamien arvojen mukaan.

3.3.3 CloverDX

CloverDX on toinen valitsemistani ohjelmista/alustoista, jolla kokeilin tietojensiirron tekemistä. Sen toimintaperiaate ja siihen liityvät konseptit ovat hyvin samankaltaisia TOS-DI:n kanssa, joten pääsin sen kanssa nopeammin vauhtiin. CloverDX on samalla tavoin Java ja Eclipse-pohjainen kuin TOS-DI ja se asennettiin samalla tavalla kuin mikä tahansa muu ohjelma.

Alkumääritykset ja projektin luonti tapahtui samalla tavalla kuin TOS-DI:ssä. Loin tätä työtä varten graafin nimellä "toimenpiteet", johon lähdin kasaamaan työkalua.



Kuvio 5 CloverDX

3.3.3.1 Tietokanta-asetukset ja -komponentit

Tein työkalun rakentamisen samassa järjestyksessä kuin TOS-DI:n kohdalla. Ensimmäisessä vaiheessa määritin projektin käytössä olevat tietokanta-asetukset. TOS-DI:stä poiketen CloverDX:ään määritettiin yksi yhteys per tietokanta. Projektin alta löytyi suoraan oma paikka tietokantayhteyksille, eivätkä ne olleet TOS-DI:n tavoin piilossa metatietojen alla. Lisäsin asetuksiin oman yhteysmäärityksen Alpha- ja Bravo-tietokannoille.

Tämän jälkeen lisäsin lähdetietojen lukua varten kaksi DBInputTable-komponenttia samalla tavoin kuin TOS-DI:ssä, eli etsimällä sopivan komponentin valikoimasta ja raahaamalla sen kanvakselle. Komponenttien asetuksiin määritettiin käytettävä tietokantayhteys, kantaan tehtävä SQL-kysely ja tietolähteen merkistökoodaus.

3.3.3.2 Täsmäytyskomponentti

Tietojen sisäänluvun jälkeen tein tietojen täsmäytyksen ja lopullisen tietorakenteen Combine-komponentillä. Se otti vastaan kaksi sisääntuloa, joitten perusteella määritettiin yksi ulostulo. Kenttien täsmäytys tapahtui hyvin samankaltaisesti TOS-DI:n kanssa: Sisääntulevat kentät täsmäytettiin ulostuloon määritettyyn rakenteeseen. Ulostuloksi sain määritettyä yhden päätason kentän "data", jonka alle sain suoraan kopioitua Bravo-järjestelmän tietorakenteen samalla tavoin kuin TOS-DI:ssä. Siitä kuitenkin poiketen mm. hintatietojen yhdistyksessä tekemäni laskennat oli helpompi määritellä CloverDX:ssä.

Tietomuunnoksissa pystyi käyttämään CloverDX:n omaa syntaksia ilman Javan tarvitsemia tyyppimäärityksiä. Syntaksissa sisääntuloihin viitattiin muuttujilla *$in.0* ja *$in.1* ja niiden perään erotettiin pisteellä haluttu tietotaulun sarake. Kun sisääntulosta valitsi kaikki hintakentät ja raahasi ne ulostulopuolelle määritettyyn sarakkeeseen, CloverDX teki automaattisesti tähän väliin tietomuunnoksen ja oletuksena summasi sarakkeet yhteen. Verollisen hinnan pystyi laskemaan vastaavasti, mutta veron osuus piti lisätä käsin kertolaskulla.

3.3.3.3 JSON-, HTTP- ja lokikomponentit

Tietojen täsmäytyksen ja lopullisen rakenteen määrittämisen jälkeen muutin tiedot JSON-muotoon JSONWriter-komponentillä. Komponentin "täsmäytys" -asetuksen alta sisääntulevista tiedoista piti koostaa lopullinen JSON-rakenne. Koska olin jo määrittänyt haluamani sarakkeet aiemmin, pystyin suoraan valitsemaan kaikki sisään tulevat tiedot ja vetämään ne JSON-juurielementin alle. Komponentin ulostulon pääsi määrittämään vasta sen jälkeen, kun sen oli yhdistänyt seuraavaan komponenttiin.

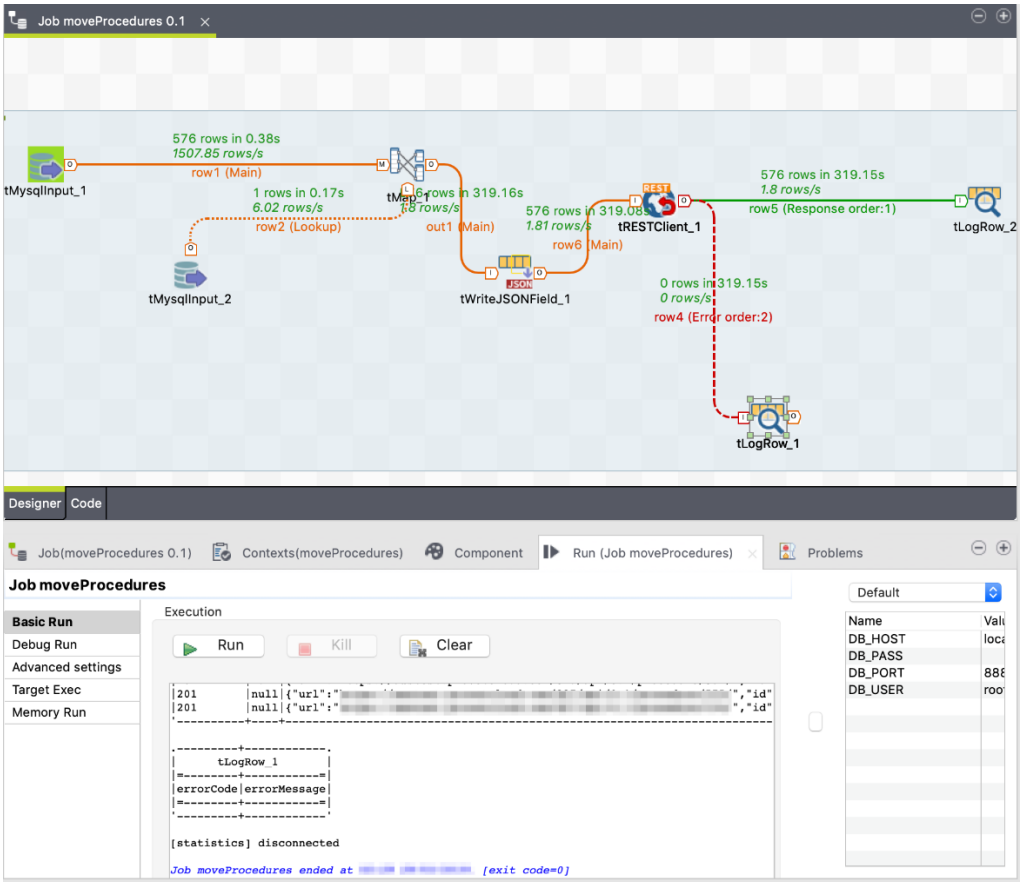
Seuraava vaihe oli tietojen lähetys REST API:in, joten lisäsin tätä varten HTTPConnector-komponentin. Kun olin yhdistänyt nämä komponentit, JSON-komponentin ulostulon pystyi määrittämään komponentit yhdistävän linkin asetuksista ja välittämään HTTP-pyynnön sisällöksi. Muut HTTP-asetukset sain laitettua paikalleen komponentin asetuksista. TOS-DI:stä poiketen CloverDX:stä ei löydy kontekseja, vaan käytettävät osoitteet ym. tiedot määritetään suoraan komponenttiin.

Lopuksi lisäsin onnistuneille ja epäonnistuneille HTTP-pyynnöille omat Trash-komponentit, jotka tallensivat tilapäisesti ajojen tuottamat tulokset.

3.3.4 Tietojensiirron testiajot

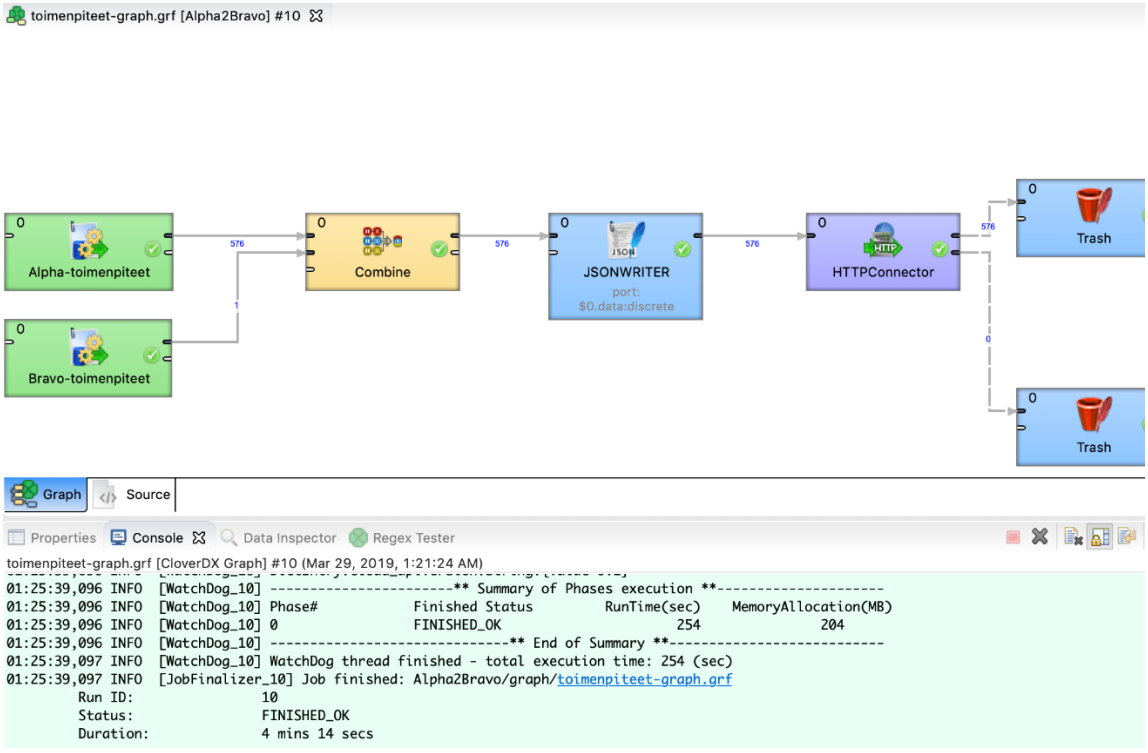
Tietojensiirron rakentamisen jälkeen seurasi luonnollisesti työkalujen testiajot. Pystytin testiajoja varten oman Bravo-järjestelmän verkkoympäristön, jota vasten tein ajot sekä TOS-DI:stä että CloverDX:stä. Tyhjensin testiympäristön tiedot ajojen välissä. Lähdeaineistona käytin Alpha-järjestelmän testiversion toimenpideluetteloa, jossa oli 576 lähetettävää toimenpidettä.

TOS-DI:ssä työn ajo tapahtuu suoraan omalta välilehdeltään graafin alapuolelta. Lokikomponentit kirjoittivat tietoja suoraan ajon konsoliin. Ajon aikana ohjelma näytti suoraan graafin päällä, kuinka kauan missäkin välivaiheessa kesti. Tämä ominaisuus on mielenkiintoinen eikä sellaista löytynyt CloverDX:stä. Ajoissa huomasin JSON-komponentin muodostuvan ns. pullonkaulaksi, mikä rajoitti tietojen kokonaislähetysaikaa. En alkanut tämän työn puitteissa selvittämään syytä tähän vaan tyydyin siihen, että työkalu yleensä ottaen toimi.



Kuvio 6 TOS-DI työn ajon tulos

CloverDX:ssä työn ajo tapahtuu omasta pääohjelman valikostaan. Tämä avasi TOS-DI:n tavoin oman konsoli-ikkunan graafin alle, mistä ajon etenemistä pystyi seuraamaan. CloverDX oli saman tietomäärän siirrossa jostain syystä n. minuutin nopeampi kuin TOS-DI.



Kuvio 7 CloverDX ajon tulos

Taulukkoon 2 olen listannut tieojensiirron rakentamiseen kuluneen ajan, itse tietojensiirtoajon keston sekä alimman hintaluokan työkalun maksullisesta versiosta, joka sopisi tuotantoajojen ajoa varten. Hintaan sisältyy käyttöönottokulut sekä ensimmäisen vuoden käyttökustannukset. Hinnat ovat saattaneet muuttua hintatiedustelujen jälkeen (kesä-/heinäkuu 2017).

Taulukko 2 Työkalujen vertailu

	TOS-DI	CloverDX
Rakennusaika	31h	18h
Siirtoaika	5m 19s	4m 14s
Alin hintaluokka	~ 10 000 €/v	~ 45 000 €/v

3.4 Yhteenveto

Yo. taulukkoon 2 viitaten järjestelmien siirtoajat tuntuvat yleensä ottaen melko hitaille. Tämä johtuu luultavasti siitä, että ajoissa käytetään yhtä http-kanavaa lähetysten tekoon. Molempien järjestelmien maksullisista versioista löytyy mahdollisuuksia töiden rinnakkaisajoon käyttämällä useita koneita (tai yhteyksiä) töiden ajamiseen. Tämä toisi lisää nopeutta töiden ajoon. Lisäksi graafeissa käyttämiäni komponentteja ja niiden asetuksia on luultavasti mahdollista jonkin verran optimoida, mikä toisi hieman lisää nopeutta ajoihin.

Tiedonsiirtotestien rakennusaikojen välisiin eroihin on monia syitä:

1. TOS-DI oli ensimmäinen alusta, millä tein tietojensiirtotestin. Tätä ennen en tuntenut tarvittavaa termistöä enkä ollut aiemmin käyttänyt mitään vastaavaa alustaa. CloverDX oli ylipäätään hyvin saman kaltainen kuin TOS-DI, joten sen peruskäyttö onnistui nopeammin.
2. TOS-DI:n komponenttivalikoima oli kattavampi kuin CloverDX:n. Tämä hidasti hieman sopivien komponenttien löytämistä. Molemmissa ohjelmissa komponenttien asetukset olivat yhtä työläitä käydä läpi.
3. TOS-DI:n oikea dokumentaatio oli vaikea löytää ja sen löydettyänikin kaipaamiani asioita oli vaikea löytää sieltä. Keskustelupalstat osoittautuivat virallista dokumentaatiota paremmaksi avun lähteeksi. CloverDX:ssä dokumentaatio oli tehty paremmin.
4. En ole juurikaan ohjelmoinut Javalla ja TOS-DI vaati juuri Java-osaamista täsmäytyskomponentin muokkaukseen. CloverDX:ssä käytettiin enemmän C:tä muistuttavaa omaa syntaksia, joka oli helpompi omaksua lennosta kuin puhdas Java.

TOS-DI on selvästi edullisempi vaihtoehto kuin CloverDX. Hinta ei kuitenkaan ole tavattoman houkutteleva, koska käyttökokemus oli TOS-DI:ssä heikompi kuin CloverDX:ssä. CloverDX oli mukava käyttää, mutta koska tietojensiirrot eivät ole yrityksen pääliiketoimintaa, se olisi varsinkin tämän työn aloitushetkellä ollut kannattamaton hankinta (tai se olisi asettanut hinnankorotuspaineita).

Oma toteutus tietojensiirtoalustasta valikoitui tämän testauksen, saatujen hintatietojen ja aikaisempien kokemusten myötä yrityksen lähestymistavaksi tietojensiirtoon.

4 TIETOJENSIIRTO

Tässä luvussa käsitellään varsinaista tietojensiirtoa Alpha-järjestelmästä Bravo-järjestelmään. Luvussa 3 käsitellyn tietojensiirron esiselvityksen perusteella tietojensiirtotyökalu päätettiin toteuttaa itse. Esiselvitys tapahtui loppuvuoden 2016 ja kesän 2017 välisenä aikana, jonka jälkeen (ja osittain jo tänä aikana) alettiin tehdä pohjaa Bravo-järjestelmään sopivalle tietojensiirtotyökalulle. En osallistunut tässä vaiheessa itse työkalun kehittämiseen, vaan tein pelkän esiselvityksen. Työkalun ”ydin” toteutettiin muualla yrityksessä. Työkalusta haluttiin kehittää modulaarinen siten, että samaa työkalua voitaisiin käyttää Bravoon tietojen tuomiseen monista eri järjestelmistä.

Tiedonsiirtomenetelmien selvittämisen lisäksi tämän työn tavoitteena on toteuttaa tietojensiirtoliitännäinen omatekoiselle alustalle Alpha-järjestelmän tietojen siirtämiseksi Bravo-järjestelmään. Tähän työhön liittyy erään Alpha-järjestelmää käyttävän asiakkaan halu siirtyä käyttämään Bravoa. Tässä järjestelmävaihdossa halutaan siirtä rajatuilta osin heidän Alphaan tallentamansa tiedot uuteen järjestelmään. Tämän opinnäytetyön puitteissa toteutettiin liitännäinen Alphan tietojen siirtämiseksi Bravoon sekä kyseisen asiakkaan järjestelmän vaihto.

Luvun aluksi käydään läpi yrityksessä rakennetun tietojensiirtoalustan pohjatoteutus. Sen jälkeen käsitellään tiedonsiirtoliitännäisen toteutusta ko. alustalle Alphan tietojen siirtämiseksi. Lopuksi käsitellään itse tietojen siirtoa kyseistä liitännäistä käyttämällä.

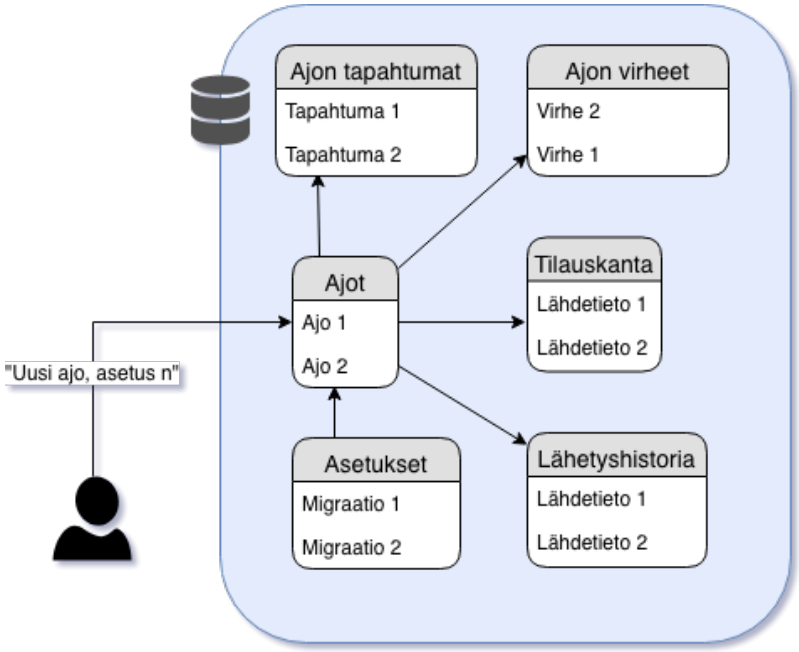
4.1 Tietojensiirtotyökalun pohjatoteutus

Tietojensiirtotyökalu rakentuu PHP-sovelluskoodista ja MySQL-tietokannasta. Työkalulla ei ole graafista käyttöliittymää, vaan sitä ajetaan suoraan komentoriviltä. Työkalun perusajatus on, että jokaista lähdejärjestelmää varten tehdään oma ”migraatioliitännäinen” omaan tiedostoonsa, jonka kautta ajettaisiin kaikki ko. järjestelmästä peräisi olevat tiedot. Kaikki eri liitännäiset käyttävät samaa työkalun pohjatoteutusta, joka vastaa siirtojen asetuksista, lokikirjauksista, tietojen lähettämisestä Bravoon sekä täydennysajojen kasaamisesta ja ajosta.

Työkaluun on toteutettu ns. ”deltaominaisuus”, eli mahdollisuus tehdä aiempia ajoja täydentäviä lisäajoja. Tällä tavoin alustava suurempikokoinen tietojensiirto voidaan tehdä ennen varsinaista Bravo-järjestelmän käyttöönottoa ja juuri ennen sitä tehdä pelkkä nopeammin valmistuva täydennysajo. Tällöin varsinainen käyttöönottoajo saadaan valmistumaan nopeammin, kun suurin osa tiedoista on ajettu sisään jo aiemmin.

Tietojensiirtojen taustalla työkalu käyttää omaa tietokantaa

- erillisten migraatioajojen määrittämiseen
- ajojen tuottamien JSON-tietueiden laatimiseen ja tallentamiseen, jotka sisälsivät lähetettävät tiedot
- ajojen lokitietojen kirjaamiseen
- tietojen muuttumisen vertailuun täydennysajoissa.



Kuvio 8 Tietojensiirtoalustan tietokanta

Tietojensiirrossa työkalu kasaa tietokantaan ajon aikana lukemistaan lähdetiedoista JSON-tietueita yhdessä kyseisiin tietoihin liittyvien, REST API -pyyntöjen tekemiseen tarvittavien tietojen kanssa. Kun työkalu on saanut käytyä kaikki lähdeaineistot läpi ja valmisteltua ne tietokantaan lähetystä varten, se lukee ne lähetysvaiheessa tietokannsta. Täydennysajoja tehtäessä työkalu tarkistaa, löytyykö kyseistä ajoa vastaavia tietoja jo kannasta. Jos löytyy ja tiedot ovat muuttuneet, kyseisistä tiedoista tehdään päivityspyynnöt. Muuttumattomista tiedoista tätä ei tehdä. Delta-ominaisuus pitää erikseen rakentaa jokaisen liitännäisen jokaisen tietoalueen käsittelyyn, joten kaikkien tietojen tuonnissa ei välttämättä käytetä ko. ominaisuutta.

Työkalun käyttö tapahtuu ajamalla komentoriviltä ns. ”päätiedostoa”, jolle annetaan muuttujien avulla tieto siitä, mitä tietokantaan ennalta määritettyä ajoa ollaan ajamassa ja mihin ympäristöön (testi-, koulutus- vai tuotantoajo). Ajon käynnistämisen jälkeen sen etenemistä pystyy seuraamaan tietokannan lokitauluista, minne työkalu päivittää edetessään tietoja kuluvasta tilanteesta.

Työkalun asetustauluun kasataan tiedot kustakin ajettavasta migraatiosta. Tiedoista käy ilmi lähdejärjestelmän tyyppi (ja samalla ajettava migraatiokooditiedosto), kyseistä ajoa vastaavat Bravo-järjestelmän testi-, koulutus- ja tuotantoversioiden verkko-osoitteet, tarvittavat todennuspoletit jne. Kun asetusrivi on määritetty, migraatiotiedosto tehty ja lähdetietokanta viritetty saataville, migraatioajoja voidaan tehdä.

Jokaisesta migraatioajosta syntyy projektin lokitauluun uusi rivi, eli jokainen migraatioajo saa oman ID:n. Kun migraatio käsittelee ja valmistelee lähetettäviä tietueita, ne tallennetaan kyseisen migraation ID-viittauksen kanssa. Tämän ansiosta tietoja voidaan tarvittaessa erikseen ensin valmistella lähetystä varten ja erillisellä kerralla lähettää käyttämällä valmistellun migraation ID:tä.

Luvussa 3 esitellyissä tietojensiirtoalustoissa yksi pullonkaula näytti olevan, että tietoja lähetetään yhtä http-yhteyttä pitkin, mikä on osasyy lähetysten hitauteen. Tähän seikkaan törmättiin myös omaa alustaa kehitettäessä. Omaan toteutukseen tietojen lähetys on tehty käyttämällä useita cURL-yhteyksiä (cURL multi handle). Tietojen lähetys tapahtuu asynkronisesti. Avattavien yhteyksien määrää pystyy säätämään pääohjelman muuttujalla tilanteen mukaan. Hiljaisempina aikoina, kuten myöhään illalla tai aikaisin aamulla, yhteyksiä voidaan varata enemmän. Jos ajoja ajetaan yhtä aikaa tuotantokäytön kanssa, yhteyksien määrää tulee pudottaa palvelimen kuorman helpottamiseksi.

4.2 Migraatio-osan toteutus

Tietojensiirron pilottikäyttäjäksi tätä projektia varten saatiin eräs entinen Alpha-järjestelmän käyttäjäyritys. Projektin alussa pidimme heidän kanssaan aloituspalaverin, jossa sovimme tietojensiirron laajuudesta, aikatauluista sekä muista Bravo-järjestelmän käyttöönottoon liittyvistä asioista. Tietojen siirtoon päätettiin sisällyttää pilottivaiheessa vain kriittiset Alpha-järjestelmän tiedot, jotta siirtotyökalun toteutus valmistuisi nopeammin ja että he pääsisivät käyttämään Bravo-järjestelmää nopeammin. Tietojensiirtoa voitaisiin jatkossa viedä pidemmälle. Tietojensiirtoon sisällytettiin potilas-, käynti-, kutsu- ja ajanvaraustiedot sekä potilaisiin liittyvät liitetiedostot.

4.2.1 Migraatiotiedoston rakenne ja sisältö

Aloitin Alpha-järjestelmän tietojensiirtoliitännäisen toteutuksen lisäämällä sitä varten uuden kooditiedoston projektiin. Liitännäiset on toteutettu proseduraalisesti ja koska niiden modulaarisuutta ei ole viritetty huippuunsa, jouduin tuomaan joitain osia muista liitännäisistä. Tällaisia osia ovat mm.

- sisäisten loki-, delta- ja asetustietojen käsittelystä vastaavien luokkien sisääntuonti
- lokien alustus ajon aluksi ja päättäminen ajon lopuksi
- ajon tarvitsemien muuttujien alustukset sekä
- ajojen sisällön määrittävän osan kopiointi malliksi.

Liitännäinen vastaa itse lähdejärjestelmän tiedonhausta. Tietojenhakua varten ei ole erikseen määritetty mitään tiettyä keskitettyä tapaa, vaan lähdetietojen luku tulee itse hoitaa jotenkin. Koska Alpha-järjestelmä käyttää MySQL-tietokantaa, muodostin tietokantayhteyden käyttämällä PHP:stä suoraan löytyvää PDO-kirjastoa. Loin yhteyden ajon aluksi omaan muuttujaansa ja toin sen globaalina muuttujana tietojenkäsittelyfunktioihin sisään.

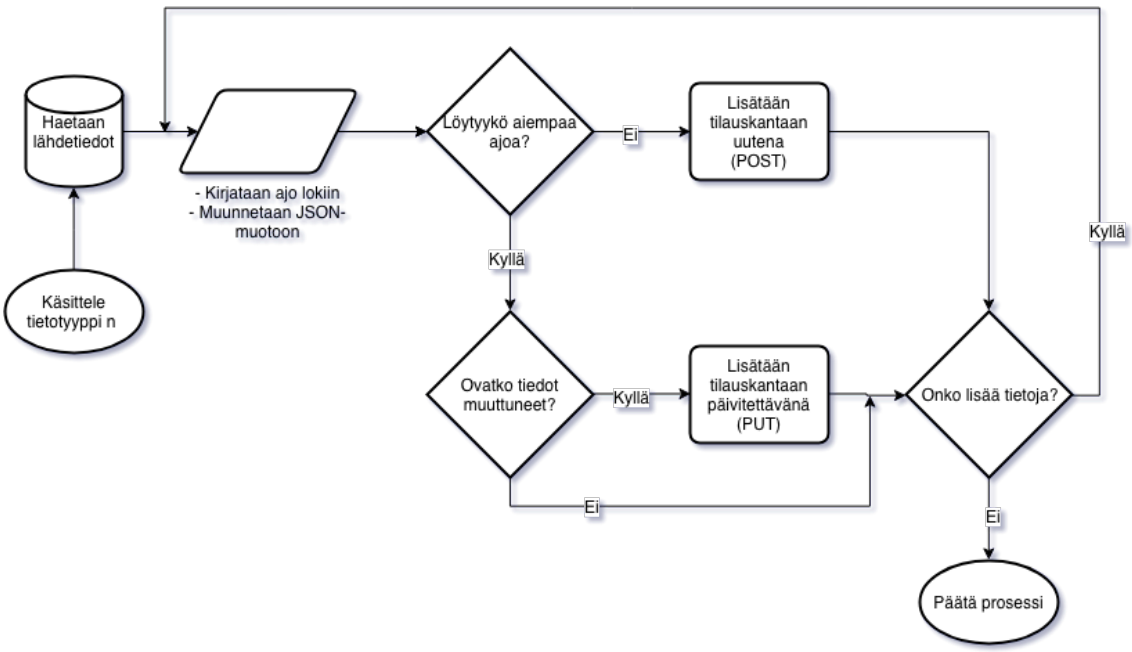
Varsinaisessa ajo-osassa tiedostoa määritetään, mitä osia tietojensiirrosta kulloinkin ajetaan. Liitteessä 1 on listattuna esimerkki ajo-osasta. Ajoissa toistuu eri import-funktioiden ajoja, jotka vastaavat tietojen luvusta lähdetietokannasta, niiden muovaamisesta JSON-tietueiksi ja tietueiden tallennuksesta ns. ”tilauskantaan” (backlog) lähetystä varten. Tietueiden lähetys Bravoon tapahtuu processBacklog-funktiokutsuissa. Funktioon annetaan parametreinä migraation ID, tietueiden tila (ei lähetetty vs. lähetetty), rinnakkaisten yhteyksien määrä, lähetettävä tietotyyppi, kyseisen migraation

loki-instanssi, Bravo-järjestelmän toimipisteen API-osoite sekä API:n todennuspoletti. Näillä tiedoilla kyseiset tiedot haetaan tilauskannasta ja lähetetään Bravoon. Eri ajoja tehtäessä tätä ajo-osaa kommentoidaan tarvittavilta osin päälle ja pois, jos esim. halutaan ajaa vain tietyt tiedot tietyllä ajokerralla.

4.2.2 Lähdetietojen käsittely

Varsinainen suurin työosuus migraatioliitännäisen toteutuksessa oli lähdetietojen käsittely. Liitteessä 1 näkyvät import-funktiot vastaavat näistä käsittelyistä. Jokaiselle lähdetietotyyppille löytyy oma import-funktio. Funktioissa tapahtuva tietojenkäsittely ja tietojen muutosten tarkistaminen on kuvattu kuviossa 9. Funktioiden sisältö noudattaa pitkälti samaa kaavaa:

- tuodaan funktioon tarvittavat globaalit muuttujat (tietokantayhteys, Bravon API-osoite, todennuspoletti jne.)
- haetaan tiedot lähdetietokannasta
- kirjataan juoksevaan tietokantalokiin tietojenkäsittelyn eteneminen
- muovataan lähdetiedoista Bravon API:a varten sopiva JSON-tietue
- tarkistetaan, löytyykö tästä tietotyypistä aiempaa ajoa (ns. Delta-tarkistus)
    - jos löytyy ja tiedot ovat muuttuneet, niin lisätään tieto tilauskantaan päivityspyyntönä (PUT)
    - jos löytyy ja tiedot eivät ole muuttuneet, niin ei lisätä tilauskantaan
    - jos ei löydy, niin lisätään tieto tilauskantaan uutena tietona (POST).



Kuvio 9 Tietojenkäsittely ja deltatarkistus

Mm. ajanvarausten ja kutsujen kohdalla tästä järjestyksestä on poikettu, eikä niihin ainakaan vielä tässä vaiheessa ole toteutettu deltaominaisuutta. Tämä johtuu siitä, että ajanvarauksia tai kutsuja

on voitu poistaa lähdejärjestelmästä, jolloin ne pitäisi poistaa myös Bravosta DELETE-pyynnöllä. Muut tiedot merkitään vain poistettaessa inaktiivisiksi, joten niille ei tarvitse toteuttaakaan varsinaista poistoa, vaan niiden tila päivitetään inaktiiviseksi. Deltaominaisuuden puuttuminen näissä kahdessa kohdassa oli alkuvaiheessa täysin siedettävää.

Typistetty versio potilaiden import-funktiosta löytyy liitteestä 2. Liitteestä on poistettu paljon globaaleja muuttujia, varsinaisia tietokenttiä ja tietojen tarkistuksia, mutta siitä käy hyvin ilmi funktioiden rakenne. Suurin osa ajasta funktioiden työstössä kuluu tietokenttien täsmäytykseen lähdejärjestelmästä kohdejärjestelmään. Samalla tulee ottaa huomioon Bravon API:n asettamat rajoitteet: jotkin tietokentät ovat pakollisia ja joitain valinnaisia tietoja ei saa tuoda tyhjänä (jos tiedot puuttuivat lähdejärjestelmästä). Tästä syystä osa lähdetiedoista voidaan asettaa suoraan JSON-vastaukseen ja osa tiedoista tulee ensin tarkistaa ja lisätä vasta, jos tieto löytyy. Samat asiat tulee huomioida kaikkien tietotyyppien API:n kohdalla.

Potilaat, muistutukset ja liitetiedostot ovat suoraviivaisimpia siirrettävistä tiedoista. Niiden kohdalla kaikki tarvittavat tiedot löytyvät 1:1 suhteella ja niiden siirto onnistuu kivuttomasti. Ajanvarauksissa ja käyntitiedoissa on jonkin verran enemmän työtä. Ajanvarauksiin liittyy se Bravon API:n puute, että ajanvarauksiin liittyviä lääkäreitä ei ole mahdollista hakea API:sta. Tästä syystä ajanvarauksia varten Bravoon pitää ennakolta lisätä kaikki Alphassa olleet lääkärit ja täsmäyttää ne ID:n perusteella ajanvarausten tuontiin käsin. Tälle osalle tein migraatio ID rajauksen, jolla sain kyseiset täsmäytykset koskemaan vain tiettyä asetuksiin määritettyä asiakkuutta. Tulevaisuudessa jos/kun API:n puute saadaan korjattua, voidaan ajanvarausten tuontia tämän osalta suoraviivaistaa.

Käyntitietojen monimutkaisuus johtuu siitä, että eri tyyppisiä käyntitietoja pitää tuoda useita. Käyntitiedot koostuvat käynneistä, käyntikertomuksista, diagnooseista, lääkityksistä, kotiutusohjeista, tehdyistä toimenpiteistä, käytetyistä lääkityksistä ja tarvikkeista, laboratorio- sekä patologialähetteistä ja niiden vastauksista.

Käyntitietojen tuonti on niputettu saman import-funktion alle. Yksittäisiä käyntitietojen tuonteja varten on tehty omat funktiot, mitkä käytännössä tekevät tietojen haut lähdetietokannasta. Käyntitietoja ei pystynyt tämän työn tekovaiheessa tuomaan ns. rakenteellisessa muodossa, vaan ainoastaan tekstimuodossa. Tämä aiheutti jonkin verran lisätyötä tietojensiirron osalta, koska rakenteellisista lähdetiedoista piti koostaa riittävän tarkat tekstiversiot ja yhdistää ne käyntikohtaisesti yhdeksi pitkäksi tekstiksi. Tämän raportin kirjoitusvaiheessa käyntitietoja voi nykyään tuoda rakenteellisesti.

4.3 Migraation määritys ja ajaminen

Tietojensiirron lähde- ja kohdejärjestelmien tiedot pitää syöttää alustan asetustietokantaan tietojensiirron määrittämiseksi. Kun tiedot on syötetty, kyseistä ajoa voidaan ajaa esim. komennolla

- *php converter.php --settings_id=10 --conversion_type=test*

missä settings_id-parametrin arvo on halutun, tietokantaan määritetyn ajon ID. conversion_type määrittää, onko kyseessä testi, koulutus vai tuotantoympäristöön kohdistuva ajo. Kutakin ympäristöä vastaavat osoitteet määritetään samalle asetusriville tietokantaan. Tietojensiirron toteutuksen edetessä tein siirtotestejä yksittäisille tietoalueille sitä mukaa, kun ne valmistuivat. Käytin testeihin erikseen niitä varattua pilviasennusta Bravosta. Tyhjensin järjestelmän tietokantaa sitä mukaa, kun halusin tehdä uusia ajoja. Tein tietojen tyhjennyksen suoraan tietokantaan tyhjentävällä SQL-skriptillä.

Kun olin tyytyväinen tekemiini testeihin, tein alustavat tuotantojärjestelmän ajot pilottikäyttäjän uuteen Bravo-järjestelmään. Järjestelmä ei ollut tässä vaiheessa vielä tuotantokäytössä, joten sinne pystyi ajamaan tietoja rauhassa etukäteen. Ajojen jälki tarkistettiin sekä meidän että pilottikäyttäjän toimesta. Ajojen jälkeen oltiin molemmin puolin melko tyytyväisiä. Käyntitekstien ulkoasuun tehtiin joitain pieniä korjauksia ensimmäisten ajojen jälkeen ja ne saatiin päivitettyä onnistuneesti ns. delta-ajolla. Ennen varsinaista käyttöönottoa ajettiin kaikki muut paitsi ajanvaraus- ja kutsutiedot, jotka ajettiin vasta käyttöönottoa edeltävän täydennysajon mukana.

Uuden järjestelmän tuotantokäyttöönotto sujui hienosti. Delta-ominaisuuden myötä käyttöönottovaiheessa sisään ajettava tiedon määrä on paljon pienempi, kuin mitä aiemman sukupolven tietojensiirtotyökaluilla se olisi ollut. Tästä syystä käyttöönottoajo kesti pilottiasiakkaan tiedoilla käyttökatkoineen noin puoli tuntia, kuin se koko tietomäärän kanssa olisi voinut viedä useita tunteja. Tietojensiirtotyökalun kehittämistä jatketaan kattamaan vielä puuttuvia tietoalueita.

5 TOIMINNALLISUUKSIEN SIIRTO

Alpha ja Bravo-järjestelmien ikäero on 10 vuotta. Tämän selvän etumatkan vuoksi Alphasta löytyy enemmän ominaisuuksia kuin Bravosta. Jotta Alpha-järjestelmän käyttäjät voisivat siirtyä käyttämään Bravoa, siitä pitäisi löytyä vähintäänkin kriittisiltä osin samat toiminnallisuudet kuin Alphasta. Toisena tämän työn pääkysymyksenä onkin, miten nämä Bravosta puuttuvat toiminnallisuudet saataisiin parhaiten siirrettyä Alphasta tai muuten toteutettua siihen.

Luvun aluksi käydään läpi kysymykseen liittyviä taustatietoja ja syitä. Tämän jälkeen käsitellään yleisesti mikropalveluarkkitehtuuria, sen syntyperää ja siihen liittyviä käsitteitä. Luvun lopuksi keskitytään erään Alphassa olleen toiminnon siirtämiseen Bravoon käytännön tasolla.

5.1 Taustatietoja

Alpha ja Bravo-järjestelmät on toteutettu toisistaan eri ohjelmointikielillä. Siksi toimintojen siirtäminen sellaisenaan ei ole mahdollista. Tästä rajoitteesta johtuen ajattelin aluksi, että toiminnallisuuksien siirtämiseen ei liene muuta keinoa, kuin puuttuvien toimintojen listaaminen, niiden teknisten- ja toiminnallisten kuvausten tekeminen sekä niiden pohjalta uudelleentoteuttaminen suoraan Bravoon. Etenkin ydintoimintojen osalta tämä on luultavasti ainut oikea ratkaisu, mutta en aluksi osannut ajatella tämän pidemmälle. Aluksi ajattelin, että tämä luku tulisi enemmän käsittelemään dokumentaatioon liittyviä asioita itse toiminnallisuuksien siirron sijaan. Alpha-järjestelmään kehitetyistä toiminnoista on tehty vain hyvin pintapuolisia toteutussuunnitelmia, joten olemassa olevaa dokumentaatiota ei voi juurikaan käyttää uudelleentoteutuksissa hyväksi.

Kuulun itse Alpha-järjestelmän kehitys- ja ylläpitoryhmään. Keväällä 2018 ryhmämme sai tehtäväksi toteuttaa Bravo-järjestelmän käyttöön tilausyhdyskäytävän. Yhdyskäytävän perusajatuksena oli yhdenmukaistaa Bravon kautta tehtävien tilausten lähetys- ja vastaanottotoiminnot sen näkökulmasta siten, että ne toimisivat samalla tavalla tilauksen toimittajasta riippumatta. Tunnemme Alpha-järjestelmän ajoilta muutamia eri toimittajia ja tiesimme ennalta, että kullakin heistä on erilaisia vaatimuksia tilausten vastaanoton suhteen: joku heistä haluaa tilaukset XML-tiedostoina SFTP-yhteyden yli, toinen haluaa tilaukset sähköpostilla ja kolmas käyttää REST-rajapintaa.

Yksinomaan Alpha-järjestelmän käyttöön oli joitain vuosia aiemmin toteutettu vastaavanlainen tilausyhdyskäytävä, joka oli pohja-ajatukseltaan samanlainen. Tilausyhdyskäytävä oli toteutettu ns. ”ulkoiseksi palveluksi” Alpha-järjestelmän näkökulmasta. Tämän tehtävän yhteydessä tutkittiin, voisiko Bravo käyttää tätä samaa yhdyskäytävää. Pikaisen projektiin tutustumisen jälkeen kävi ilmi, että teknisesti Bravo voisi käyttää sitä. Tulin kuitenkin siihen tulokseen, että se kannattaisi rakentaa uudestaan. Tälle oli useita perusteita:

1. Yhdyskäytävän rajapinta ei noudattanut hyviä käytäntöjä eikä se ollut suoraviivainen. Tilauksen lähettäminen oli turhaan tehty kaksivaiheiseksi: ensimmäisessä vaiheessa tehtiin tilauksen lisäys

yhdyskäytävälle ja toisessa vaiheessa tapahtui varsinainen lähetys toimittajalle. Lähettävän järjestelmän näkökulmasta ei tullut koskaan tilanteita, milloin nämä vaiheet olisi haluttu tehdä toisitaan erillään, joten vaiheita olisi lähettävän järjestelmän näkökulmasta voinut olla vain yksi. Lisäksi kaikki rajapintapyynnöt tehtiin GET-, eikä POST-pyynnöillä eri osoitteisiin sen mukaan, oliko kyseessä tilauksen lisäys vai lähetys.

2. Uusien toimittajien lisääminen yhdyskäytävälle oli työlästä. Kullakin toimittajalla oli toisistaan erilliset toteutukset, eikä niillä ollut juurikaan mitään yhteistä.
3. Yhdyskäytävä käytti vanhentunutta CakePHP-ohjelmointikehyksen versiota, joten sen päivittäminen olisi joka tapauksessa vaatinut panostusta. Lisäksi minulla ja kollegoillani ei juurikaan ollut aiempaa kokemusta CakePHP:stä.

Mm. näistä syistä yhdyskäytävä päätettiin rakentaa uudestaan. Vanhasta yhdyskäytävästä saatiin tiedot muutamien sinne lisättyjen toimittajien käyttämistä siirtotavoista ja ne huomioitiin uuden yhdyskäytävän toteutuksessa. Yhdyskäytävä päätettiin samalla rakentaa suoraan niin, että sitä voisivat vaivatta käyttää Alpha, Bravo tai mikä tahansa muu järjestelmä. Uusien toimittajien lisäämisestä ja yhdyskäytävän ylläpitämisestä haluttiin myös tehdä aiempaa helpompaa.

Tämän toimeksiannon yhteydessä oivalsin, että mikropalvelut voisivat olla varteenotettava keino toimintojen siirtämiseksi vanhasta järjestelmästä uuteen. Jotkin tukkutilaustoiminnot ovat toimineet jo entuudestaan Alphassa yhdyskäytävän kautta, mutta mm. laboratoriotilaukset ja niiden vastausten haku on toteutettu suoraan siihen. Alphaan suoraan toteutetut laboratorioliitännät ovat hyviä kandidaatteja toiminnoista, jotka voitaisiin siirtää mikropalvelumuotoon.

Tähän saakka Bravosta Alphaan nähden puuttuvia ominaisuuksia on toteutettu suoraan Bravoon, joten tämä on ensimmäinen varsinainen kokeilu siirtää vanhaa toiminnallisuutta mikropalvelun avulla. Kokeilun onnistuessa muitakin Alphan toiminnallisuuksia tultaisiin varmasti soveltuvilta osin siirtämään mikropalveluiksi. Tämän raportin kirjoitushetkellä tällaisia ominaisuuksia on jo tunnistettu ja alettu siirtämään mikropalvelumuotoon.

Mikropalvelut olivat siitäkin syystä hyvä vaihtoehto toimintojen siirtämiseksi, koska niiden kehitykseen pystytään käyttämään Bravon kehitystiimin ulkopuolisia resursseja, eli tässä tapauksessa Alphan kehitystiimiä. Bravon käyttöön tulevia ominaisuuksia saadaan siten kehitettyä nopeammin. Alphan kehitystiimillä ei ole valmiuksia kehittää itse Bravoa. Samalla palvelua pystyisi tarpeen tullen käyttämään muutkin järjestelmät, joten samalla vaivalla voidaan tuottaa hyötyä muillekin järjestelmille.

5.2 Mikropalvelut ja mikropalveluarkkitehtuuri

Monoliittiohjelmistoksi kutsutaan yleensä suureksi paisunutta ohjelmistoa, missä järjestelmän monimutkaisuus pyritään pitämään hallinnassa abstraktiota ja modulaarisia toteutuksia hyväksikäyttämällä. Ajan kuluessa ja ohjelmiston kasvaessa sen pitäminen modulaarisena ja yhtenäisenä vaikeutuu. Tämän myötä myös järjestelmän jatkokehitys ja vikojen paikallistaminen on hankalampaa.

Palvelupohjainen arkkitehtuuri (Service-oriented Architecture, SOA) nousi aikoinaan esiin tapana hallita suuriksi paisuneiden monoliittiohjelmistojen monimutkaisuutta. Se perustuu sovellusten uudelleenkäytettävyyteen: useampi asiakassovelus voidaan määrittää käyttämään samoja osiin pilkottuja palveluja. Monista yrityksistä huolimatta SOA:n ympärille ei ole kuitenkaan muodostunut yleistä käsitystä, kuinka se voidaan toteuttaa hyvin. Lisäksi sen ongelma on, että se ei tarkalleen määrittele, kuinka suuria tai pieniä palvelujen tulisi olla, eikä se anna käytännöllisiä keinoja välttää palvelujen liian tiivistä linkittymistä toisiinsa. Sittemmin esiin noussut mikropalveluarkkitehtuuri on puolestaan saanut alkunsa nimenomaan tosielämän käyttötapauksista. Mikropalveluarkkitehtuuri voidaan mieltää SOA:n yhtenä toteutustapana, samoin kuin esim. Scrum on ketterän ohjelmistokehityksen yksi toteutusmuoto.

Mikropalvelut ovat pieniä, itsenäisiä järjestelmiä, jotka toimivat yhteistyössä muiden järjestelmien kanssa. Mikropalvelujen perusajatuksena on toteuttaa yksi selkeästi rajattu toiminto hyvin. Mikropalvelu kommunikoi muiden järjestelmien kanssa ohjelmointirajapintojen kautta. Mikropalvelujen pitäisi pystyä päivittymään toisistaan erillään ja ne pitäisi myös asentaa toisistaan erilleen, jotta ne olisivat toisistaan vähemmän riippuvaisia. (Newman, 2015 ss. 2-9)

Mikropalvelut alkoivat yleistyä 2010-luvun taitteessa. Netflix on tiettävästi yksi aikaisimmista mikropalvelujen kehittäjistä. Sen oma mikropalveluarkkitehtuurin kehitys alkoi vuonna 2008 ja tuli päätökseen vuonna 2016. Arkkitehtuurin näkökulmasta he pilkkoivat yhden ”monoliittisovelluksen” ja samalla hajauttivat siihen liittyvän tietomallin sadoiksi mikropalveluiksi. Aiemmin heidän toimintaansa hidastaneet budjettien hyväksymiskäytännöt, keskitetty versiojulkaisujen hallinta ja viikkojen mittaiset laitteistojen huoltosyklit vaihtuivat mikropalvelujen myötä jatkuvaan toimitusmalliin, missä insinööritiimit pystyivät tekemään itsenäisiä päätöksiä valitsemiaan työkaluja käyttämällä. DevOps-ympäristön sallima jousto kiihdytti tiimien innovaatiota. Arkkitehtuurimuutoksen ansiosta jo sen kehitysvaiheiden aikana (vuosien 2008 ja 2016 välillä) Netflixin katselumäärät tuhatkertaistuivat. (Izrailevsky, 2016)

Mikropalveluarkkitehtuuri voidaan Netflixin Adrian Cockcroftin mukaan mieltää palvelupohjaisena arkkitehtuurina, joka muodostuu väljästi linkitetyistä elementeistä, jotka omaavat jaettuja kontekseja. Väljällä linkityksellä tarkoitetaan, että palveluja voidaan päivittää itsenäisesti: yhden palvelun päivittäminen ei vaadi toimia muilta palveluilta. Mikropalvelut, joissa on oikein jaetut kontekstit, ovat omavaraisia ohjelmistokehityksen suhteen. Niiden kehittäminen on siis mahdollista ilman, että kehittäjä tuntee muita palveluita, joihin se on kytköksissä. Mikropalvelujen tietomallit vastaavat jaettujen kontekstien osalta toisiaan, mutta se ei välttämättä tarkoita, että ne ovat rakenteeltaan samoja. (Mauro, 2015)

Sekä Alpha- että Bravo-järjestelmät voidaan käsittää ns. ”monoliittiohjelmistoina”, koska suurin osa järjestelmän toiminnoista on toteutettu suoraan niihin. Molemmissa niistä on kuitenkin joitain ominaisuuksia, mitkä nojaavat järjestelmän ulkopuolisiin palveluihin. Suurin osa näistä ulkoisista palveluista on kolmannen osapuolen tarjoamia, mutta joukossa on myös muutamia yrityksen sisäisiä palveluita. Yksi esimerkki yrityksen sisäisestä, mutta järjestelmien näkökulmasta ulkoisesta palvelusta on tekstiviestiyhdyskäytävä, jota käyttävät monet yrityksen järjestelmät. Järjestelmissä tekstiviestien lähetysvaiheessa taustalla tapahtuva ulkoisen palvelun käyttö ei näy millään tavalla loppukäyttäjälle.

Mikropalvelujen perusajatukseen tutustuessani tulin siihen tulokseen, että tämän työn puitteissa on mahdotonta yrittää pystyttää täydellistä mallia mikropalveluarkkitehtuurista. Netflixilläkin meni oman arkkitehtuurinsa rakennukseen kahdeksan vuotta. Jotta palvelut voisivat aidosti olla toisistaan riippumattomia ja täysin vikasietoisia, niiden kaikkien käytettävissä tulisi olla jonkin tyyppinen palvelujen kartoituspalvelu (service discovery), jonka avulla palvelut saavat tietää, mistä mikin mikropalvelu löytyy. Palvelujen kartoituksen lisäksi palvelujen käytettävissä tulisi olla yhteinen tapahtumaväylä (event bus), jonka kautta palvelut voisivat välittää toisilleen tietoa uusien tietueiden lisäyksistä tai tietueiden poistoista. Esim. jos kaksi palvelua pitää yllä jaettua kontekstia käyttäjistä ja jokin olemassa oleva käyttäjä poistetaan jostain palvelusta, toisen palvelun tulisi saada tästä tieto tapahtumaväylän kautta.

Nämä mikropalveluarkkitehtuurin toteutusyksityiskohdat ovat mielenkiintoisia, mutta eivät varsinaisesti kuulu työn tavoitteisiin. Tavoitteena on tutkia mikropalveluja toimintojen siirtämisen välineenä monoliitistä yleiskäyttöiseksi palveluksi. Tulevaisuudessa työn tilaajayrityksellä on hyvä mahdollisuus jatkaa mikropalveluarkkitehtuurin kehittämistä yllä kuvattuun suuntaan.

5.3 Tilausyhdyskäytävän toteutus

Tilausyhdyskäytävän toteutus alkoi keväällä 2018 ns. puhtaalta pöydältä Laravel-nimisen PHP-ohjelmointikehyksen päälle. Yhdyskäytävän käyttötarkoituksen takia se ei tarvitse web-käyttöliittymää vaan se toimi ainoastaan rajapinnan ja tarvittaessa komentorivikutsujen kautta. Projektin tekniikkapuoli koostuu perinteisestä LAMP-pinosta. MariaDB:stä ja PHP:stä käytetään uusimpia saatavilla olevia versioita. Apachen osalta tyydytään palvelimen pakettienhallinnasta saatavaan versioon. Tekniikkapinoon voidaan lisäksi laskea Linuxin Supervisor-prosessienhallintatyökalu, joka vastaa yhdyskäytävän työjonojen hallinnasta, sekä Cron-ajastus, joka huolehtii projektiin määritetyistä, ajastetuista tehtävistä.

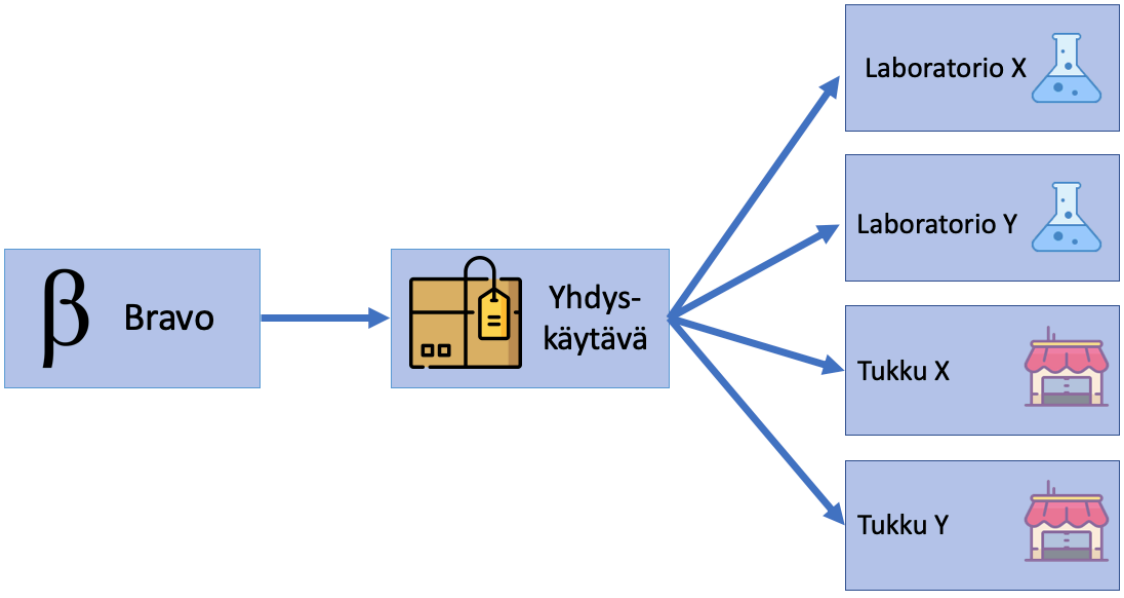
Ensimmäisiä vaiheita tyhjän Laravel-projektin alustuksen ja versiohallinnan pystytyksen jälkeen olivat palvelun tietomallien ja rajapinnan määrittäminen. Rajapintakuvaukset piti toimittaa Bravon kehitystiimille mahdollisimman aikaisessa vaiheessa, jotta he voisivat ottaa sen ajoissa huomioon tilaustoimintoja kehitettäessä. Tämän jälkeen tietomallit muutettiin kehyksen Eloquent-malleiksi ja tietokantataulujen rakenne määritettiin kehyksen migraatiotiedostoihin. Malleja vastaavat API-osoitteet ja kontrollerit määritettiin kehyksessä niille varatuille paikoille. MVC-pinosta View-osiota käytetään ainoastaan sähköposti-ilmoitusten ja XML-muotoisten tilausten luontiin.

Seuraavaksi projektiin lisättiin työjonot tilausten ja niiden lisätietopyyntöjen hallintaan. Lisätietopyyntöjä käytetään joidenkin tietyntyyppisten laboratoriotilausten osalta. Lopuksi lisättiin konsoliko-mennot tukkureiden tuotevalikoimien, laboratorioiden tutkimusvalikoimien ja laboratoriotulosten ajastettua hakua varten.

Projektia varten pystytettiin sekä testi- että tuotantoympäristöt. Testiversio keskusteli Bravo- ja Alpha-järjestelmien testiversioiden kanssa ja vastaavasti tuotantoympäristöt keskenään.

5.3.1 Yleiskuvaus

Kuten luvussa 5.1 mainitsin, yhdyskäytävän tehtävä on yhdenmukaistaa tilausten käsittely Bravon näkökulmasta siten, että Bravo toimittaa tilaukset aina samassa muodossa yhdyskäytävälle, joka puolestaan huolehtii tilauksen lähettämisestä toimittajalle sen haluamassa muodossa ja haluamalla tavalla. Bravon ja yhdyskäytävän välistä kommunikointia varten yhdyskäytävään toteutetaan REST-rajapinta. Bravossa oli entuudestaan myös REST-rajapinta ja sitä laajennetaan tämän projektin yhteydessä. Bravo toimittaa tilaukset yhdyskäytävälle rajapintaa pitkin ja yhdyskäytävä puolestaan ilmoittaa Bravolle valmistuneista tilauksista.



Kuvio 10 Tilausyhdyskäytävä

Taulukossa 4 on esitetty kaikki yhdyskäytävään liittyvät perustietomallit. Kutakin mallia varten järjestelmään luotiin oma tietokantataulu ja Eloquent-malli (eli Laravel-kehyksen MVC-malli).

Käsite	Selite
Maat	ISO 3166-1 alpha-2 -standardin mukainen maalistaus. (International Organization for Standardization, 2013)
Toimittajat	Listaus eri toimittajista ja niihin liittyvistä tiedoista, kuten rajapinta- ja sähköpostiosoitteista jne.
Maiden toimittajat	Tämä on monen-suhde-moneen -yhteydet linkittävä taulu, mikä määrittää kussakin maassa toimivat toimittajat
Toimittajatyypit	Eri tyyppisiä toimittajia oli yhdyskäytävän rakennusvaiheessa tiedossa kaksi: tukut ja laboratoriot. Toimittajatyyppejä voidaa lisätä jatkossa tarpeen mukaan
Tilaukset	Toimittajille lähetettävät tilaukset
Tuotteet	Tuotteet muodostavat toimittajien tuoteluettelot. Ne päivitetään ajastetusti kerran vuorokaudessa hakemalla tuotelistat toimittajan palvelusta
Tilauksen tuotteet	Tilauksiin sisältyvät tuotteet
Tuotetyypit	Tuoteryhmien ylläpito. Eri tuotetyyppejä ovat esim. tarvikkeet, ruokatuotteet ja lääkkeet
Käyttäjät	Tänne lisättiin palveluun tunnistautuvat käyttäjät, mitkä ovat käytännössä muita järjestelmiä. Taulu olisi voinut nimetä kuvaavammin, mutta Laravel-kehys nojasi autentikoinnin osalta oletusarvoisesti kyseiseen tauluun

Taulukko 3 Yhdyskäytävän tietomallit

Tiedon kiertokulku Bravosta yhdyskäytävän kautta aina valitulle tukkurille tai laboratoriolle asti lähtee siitä, että Bravo lataa yhdyskäytävältä listan toimittajista järjestelmän maan mukaan. Tämän jälkeen Bravo lataa kunkin toimittajan tuotelistat, johonka nojaten Bravon käyttäjä pystyy tekemään tilauksen. Tilauksen tuotteiden ja toimittajan valinnan jälkeen tilaus lähtee Bravosta yhdyskäytävälle POST-pyynnöllä. Pyynnössä kulkee mukana tilattavat tuotteet, toimittaja ja Bravo-järjestelmän maakoodi. Tietojen perusteella yhdyskäytävä tallentaa lähetteen ja lisäsää sen tilausten käsittelyjonoon.

Työjonojen käsittelyvaiheessa työt lähetetään kullekin tukulle ja laboratoriolle sen haluamalla toimitustavalla. Onnistuneesta tilauksen lähetyksestä lähetetään sähköpostiviesti Bravo-järjestelmän käyttäjäyritykselle ja vastaavasti epäonnistuneesta lähetyksestä sähköposti kehitystiimillemme. Jotkin laboratoriotilaukset vaativat Bravon käyttäjiltä lisämäärittelyjä tilauksen lisäksi. Tällaisten tilausten lähetyksessä yhdyskäytävä tekee erillisen pyynnön Bravolle, jonka seurauksena Bravon kautta pystytään lisäämään tilauksen tarvitsemat lisätiedot. Lisätiedot kulkeutuvat suoraan Bravosta laboratoriolle ilman yhdyskäytävää. Kun lisätiedot on täytetty, tilauksen tila laboratorion järjestelmässä muuttui. Kun yhdyskäytävä käy läpi ”keskeneräiseksi” merkittyjä tilauksia ja huomaa, että tiedot johonkin tilaukseen on täydennetty ja tilaus odottaa hyväksyntää, yhdyskäytävä viimeistelee tilauksen tilan.

Tukkutilauksissa yhdyskäytävän viimeinen tehtävä on onnistuneesta tai epäonnistuneesta tilauksen lähetyksestä ilmoittaminen sähköpostilla oikealle taholle. Laboratoriotilausten kohdalla yhdyskäytävä

tarkistaa ajastetusti keskeneräisten tilausten tilaa siihen saakka, kunnes laboratorio saa tutkimukset tehtyä ja tulokset lisättyä tilaukseen. Yhdyskäytävä tekee valmistuneista tuloksista ilmoituksen Bravolle tilauksen numerolla ja Bravo käy hakemassa tulokset yhdyskäytävältä.

5.3.2 Rajapintakuvaus

Yhdyskäytävän rajapinta määritettiin seuraavin reunaehdoin:

- Tilauksille haluttiin sallia CRUD-operaatiot, eli tilauksia piti pystyä luomaan, listaamaan/lukemaan, päivittämään ja poistamaan rajapinnan kautta
- Toimittajien osalta haluttiin sallia vain kaikkien listaus maakohtaisesti. Lisäksi sallittiin kunkin toimittajan tuotevalikoiman haku.

Tukuille ja laboratorioille sekä tukku- ja laboratoriotilauksille määritettiin omat rajapintaosoitteet, koska ne ovat kaikki hieman toisistaan eroavia ja ne käsitellään tietokantarakenteissa erikseen. Kukin näistä neljästä tietomallista lisättiin rajapintaresurssiksi rajapinnan reititystiedostoon. Mallien määrittäminen rajapintaresursseiksi määrittää automaattisesti kullekin resurssille omat API-päätepisteet (endpoint) CRUD-operaatioille. Reititysten lisääminen resursseina vähentää manuaalisen reittien lisäämisen kuudesta eri osoitteesta yhteen, tehden kehitys- ja ylläpitotyöstä helpompaa.

Kullekin resurssille lisättiin oma kontrolleri Phil Sturgeonin (2015 s. 22) ohjeen mukaan. Ei-halutut rajapintakutsut, kuten tukkureiden ja laboratorioiden lisäys estettiin niiden kontrolleritasolla palauttamalla JSON-vastauksessa virheviesti ja http-tilakoodi 404. Edellä mainittujen perusosoitteiden lisäksi rajapintaan lisättiin muutama osoite, joita ei saatu mukaan automaattisesti. Taulukossa 5 on listattu kaikki erikoisemmat rajapintaosoitteet, joita Laravel-kehys ei tarjonnut automaattisesti.

Rajapinnan toiminto	**Osoitteen määritys**
Tukkujen tuoteluettelojen haku	Route::get('/wholesaler/{id}/product/{product?}', ['as' => 'product', 'uses' => 'API\WholesalerController@product']);
Laboratorioiden tutkimusvalikoiman haku	Route::get('/laboratory/{id}/analyses/{analyses?}', ['as' => 'research', 'uses' => 'API\LaboratoryController@analyses']);
Laboratoriotutkimusten vastausten haku	Route::get('/referral/{id}/result/{result?}', ['as' => 'result', 'uses' => 'API\ReferralController@result']);
Laboratoriotutkimusten liitetiedostojen haku	Route::get('/referral/{id}/attachment/{attachment?}', ['as' => 'attachment', 'uses' => 'API\ReferralController@attachment']);

Taulukko 4 Erikoisempien rajapintaosoitteiden määritys

Rajapintapyynnöt autentikoidaan käyttäjätauluun lisätyillä todennuspoleteilla. Pyynnössä täytyy tulla Authorization-otsake voimassa olevalla poletilla varustettuna, jotta pyyntö hyväksytään. Tietueiden lisäys- ja muokkauspyynnöissä sisään tulevat tiedot otetaan vastaan JSON-muodossa pyynnön body-osassa. Rajapinta määritettin siten, että pyynnössä sisään tulevat tiedot täsmäävät tietomalleihin.

Joissain pyynnöissä, kuten uusien tilausten lisäyksessä itse tilaus ja siihen liittyvät tuotteet lähetetään sisäkkäisessä taulukkorakenteessa, jolloin ne erotetaan pyynnöstä omiin malleihinsa.

5.3.3 Tilausten käsittelyjonot

Yhdyskäytävän käsittelemät tilaukset eivät ole luonteeltaan kiireellisiä. Laboratoriotutkimuksien käsittely on hidasta, koska tilaajan täytyy ensin toimittaa näytteet laboratorioon, jonka jälkeen laboratorio analysoi ne ja toimittaa tulokset yhdyskäytävän kautta tilaajalle. Tukkutilaukset valmistuvat sitä mukaa, kun toimittaja ehtii käsittelemään tilaukset. Tästä syystä tilausten käsittelyyn päätettiin käyttää Laravel-kehyksen tarjoamaa jono-ominaisuutta.

Uuden tilauksen saapuessa yhdyskäytävälle siitä irrotetaan tilausosa ja tilaukseen liittyvät tuotteet ja ne tallennetaan tietokantaan. Tallennuksen jälkeen varsinainen tilauksen muotoilu toimittajan haluamaan muotoon ja toimitus toimittajan järjestelmään tapahtuu jonon kautta.

Työjonojen käsittelyyn voi käyttää monia eri kolmannen osapuolen työkaluja, kuten Amazon SQS, Beanstalkd tai Redis. Teknisen monimutkaisuuden minimoimiseksi ja tilausten kiireettömän luonteen vuoksi jonojen hallintaan päätettiin käyttää yhdyskäytävän tietokantaa. Työjonolle ja töiden ajovirheille luotiin omat tietokantataulut Laravel-kehyksen artisan-komentoriviapurilla. (Otwell, 2019)

- php artisan queue:table

Työjonojen käsittelyjä varten luotiin myös omat työluokat artisan-komennoilla. Tukku- ja laboratoriotilauksilla sekä laboratoriotilausten lisäpyynnöillä on omat käsittelyjononsa.

- php artisan make:job ProcessOrder
- php artisan make:job ProcessReferral
- php artisan make:job ProcessReferralInformation

Tilausten lisääminen jonoon tapahtuu tietokantaan tallentamisen jälkeen kutsumalla halutun jonoluokan dispatch-funktiota, jolle annetaan parametreinä tilausobjekti. Funktiokutsuun putkitetaan samalla delay- ja onQueue-funktiokutsut, joilla työlle määritetään 30 sekunnin käsittelyviive sekä kerrotaan, mihin jonoon työ halutaan lisätä. Tämä lisää työn jonotauluun odottamaan käsittelyä. Jonojen ajot tapahtuvat pitkäkestoisissa PHP-prosesseissa, jotka pystytään käynnistämään artisan-komennolla. Artisan-ajoissa on kuitenkin se huono puoli, että jonojen suoritus loppuu, jos terminaali suljetaan. Kehitysvaiheessa tämä oli siedettävä ongelma, mutta tuotannossa prosessien pitää pysyä jatkuvasti päällä.

Tähän ongelmaan sopiva ratkaisu oli käyttää Linux-käyttöjärjestelmälle saatavilla olevaa supervisor-prosessienhallintatyökalua, joka huolehtii komentojen automaattisesta ajamisesta ja uudelleenkäynnistyksistä virheiden sattuessa. Supervisor piti määrittää asetustiedoston avulla ajamaan haluttuja

jonoja artisan-komennoilla. Kaikki yhdyskäytävään liittyvät jonot on mahdollista lisätä samaan tiedostoon. Tiedoston lisäksi supervisord-prosessi pitää asettaa palvelimen käynnistyksessä käynnistyväksi palveluksi. Esimerkki supervisorin asetustiedostosta löytyy liitteestä 3.

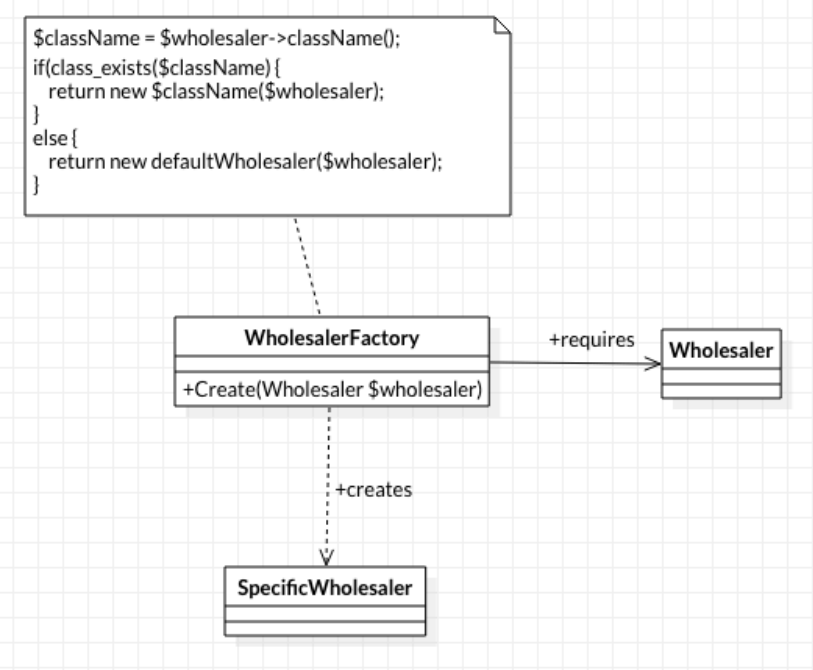
Jonot toimivat supervisorin ajamina hyvin. Kullekin jonolle annetaan supervisorin kautta kaksi PHP-prosessia, joten palvelin käsittelee kahta kunkin jonon työtä yhtä aikaa. Supervisorin kanssa muistettava seikka on kuitenkin se, että jos yhdyskäytävän koodia päivitettiin palvelimelle, jonojen käsittely pitää käynnistää uudestaan, jotta koodimuutokset tulevat jonojen osalta voimaan.

5.3.4 Toimittajakohtaisten erojen käsittely

Työjonoluokat vastaavat tilausten muuntamisesta toimittajille sopivaan muotoon ja tilausten lähetyksistä toimittajan toivomalla tavalla. Kuten luvussa 5.1 mainitsin, kullakin toimittajalla on erilaisia vaatimuksia tilausten tietomuodon ja toimitustavan suhteen. Tästä syystä toimittajien käsittelyyn piti kehittää joustava tapa.

Koska kukin toimittaja tulee todennäköisesti olemaan jollain tavalla muista poikkeava, eri toimittajia varten päätettiin perustaa omat luokkatiedostot ja ne ryhmiteltiin kansioittain tukkuihin ja laboratorioihin. Luokkatiedostot nimettiin toimittajan ja maakoodin mukaan sen perusteella, missä maassa kyseinen tukku toimi. Kunkintyyppisiä toimittajia varten tehtiin omat tehdasluokat, jotka vastasivat tilausta vastaavan toimittajan luonnista.

Työjonoluokat vastaavat tehdasluokkien käytöstä. Tehdasluokilla on ainoastaan yksi create-metodi, joka ottaa parametrinä tilaukseen liittyvän toimittajan. Tehdas palauttaa työjonolle toimittajaa vastaavan luokan, joka vastaa tilausten muotoilusta ja lähetyksestä sopivalla tavalla. Tehtaan palauttamat luokat sisältävät keskenään samat metodit, joten työjonossa voidaan tehdä samat metodikutsut riippumatta siitä, mistä toimittajasta on kyse.



Kuvio 11 Tehdasluokka

Tämän opinnäytetyön työstämisen aikana kaikki yhdyskäytävälle lisätyt tukut halusivat tilaukset XML-tiedostoina SFTP-yhteyden kautta siirrettynä heidän palvelimilleen. XML-tiedostojen muodostamiseen käytetään Laravel-kehyksen Blade-pohjia (template). Kullekin tukulle luotiin heidän kaipaamansa XML-tiedoston pohjarakenne, joihin ajon aikana täydennetään tilauksen tiedot. Toimittajaluokat vastaavat kullekin kuluvan XML-pohjan lataamisesta ja täyttämisestä.

SFTP-yhteyksien asetukset, eli Laravel-termein ”tiedostoajurit”, lisättiin toimittajittain kehyksen tiedostojärjestelmäasetuksiin. Kullekin toimittajalle lisättiin tietokantaan tieto ajurista, mitä kukin toimittaja käyttää. Työjonossa kutsutaan tilaustiedoston luonnin jälkeen siirtofunktiota, joka lähettää tilauksen toimittajalle.

5.3.5 Ajastetut komennot

Yhdyskäytävä nojaa tukkureiden tuotevalikoimien, laboratorioiden tutkimusvalikoimien ja laboratoriotulosten osalta toimittajilta peräisin oleviin tietoihin. Bravo-järjestelmät hakevat nämä nimikkeistöt yhdyskäytävältä ja uusien tilausten teko Bravosta nojaa niihin. Koska toimittajat saattavat milloin tahansa tehdä muutoksia näihin nimikkeisiin, yhdyskäytävän pitää huolehtia, että nimikkeistöt ovat aina ajan tasalla. Nimikkeiden päivitykset tapahtuvat kuitenkin sen verran harvoin, että niiden osalta riittää, että ne päivitetään kerran vuorokaudessa. Laboratoriotulokset päivittyvät useita kertoja päivässä, joten niitä haetaan minuutin välein.

Tukkujen kanssa oli aiemmin sovittu käytännöstä, että he toimittavat meille tuotelistat Excel-tiedostoissa. Nämä tiedostot ladataan yhdyskäytävän saataville meidän toimestamme ja yhdyskäytävä huolehti itse, että päivitykset ajetaan niistä. Laboratorioiden osalta tutkimusvalikoimat ja -vastaukset saadaan ladattua niiden SOAP-rajapinnoista, joten päivitykset tehdään hakemalla nimikkeistöt sieltä.

Molempien toimittajatyyppien osalta nimikkeiden päivitykset piti automatisoida, koska ne tapahtuvat vähintään kerran päivässä. Tähän käytetään Laravel-kehyksen tehtävien ajastusominaisuutta. Tehtäviä pystyy ajastamaan kehyksen komentoriviytimen kautta, joten päivitysrutiinit päätettiin tehdä omien artisan-komentojen kautta. Artisan-komennot käyttävät hyväkseen aiemmin luotuja tehdasluokkia. Kullekin toimittajalle lisättiin nimikkeistöjen päivitysmetodit. Artisan-komennot käyvät läpi kaikki eri tyyppiset toimittajat ja kutsuvat tarvittavia päivitysmetodeja.

Uudet artisan-komentoluokat saadaan lisättyä omalla artisan-komennolla:

- php artisan make:command WholesalerProducts
- php artisan make:command LaboratoryTests
- php artisan make:command LaboratoryResults

Komentojen kutsutapa määritetään komentoluokkien signature-parametriin. Esim. tukkureiden tuoteluettelojen päivityskomennoksi määritettiin *wholesaler:products* ja kyseistä komentoa voitiin kutsua suoraan artisanin kautta: *php artisan wholesaler:products*. Komentojen käsittelymetodissa haetaan kaikki kyseisen tyyppiset toimittajat ja kutsutaan toimittajaluokan päivitysmetodia. Kuviossa 10 on esimerkkinä laboratoriotulosten ajastettu päivitys.

```php
/**
 * Execute the console command.
 *
 * @return mixed
 */
public function handle()
{
    //
    $laboratories = Laboratory::all();
    $laboratories->each(function($laboratory)
    {
        LaboratoryFactory::create($laboratory)->getReferralResults();
    });
}
```

Kuvio 12 Ajastettava komentoluokka

Komentojen ajastukset määritetään komentoriviytimeen schedule-metodiin. Ajastukset lisättiin kutsumalla Schedule-luokan command-metodia, jolle annetaan parametrinä ajettava artisan-komento, esim. *wholesaler:products*. Näihin funktiokutsuihin putkitetaan lisäksi haluttu ajosykli, esim. *daily* tai *everyMinute*.

- $schedule->command('wholesaler:products')->daily();

Lopuksi palvelimelle määritettiin kaikille yhdyskäytävän ajastetuille komennoille yhteinen cron-ajastus. Komentoja ajetaan todellisuudessa komentoriviytimeen määritetyin aikavälein, vaikka itse cron-ajo suoritetaan joka minuutti. Cron laitettiin ajamaan artisanin *schedule:run* -komentoa.

5.4 Yhteenveto

Lopputuloksena yhdyskäytävästä muodostui aito, oikea ja käyttökelpoinen mikropalvelu. Yhdyskäytävä kommunikoi Bravo-järjestelmien ja eri tukkureiden välillä itsenäisesti rajapintoja ja muita toimittajien määrittämiä yhteysmenetelmiä käyttämällä. Ensimmäisessä vaiheessa siihen lisättiin kolme jo entuudestaan vanhassa yhdyskäytävässä ollutta tukkuliitäntää sekä yksi laboratorioliitäntä. Tämän raportin kirjoitushetkellä tukkuliitäntöjä on pian valmistumassa toiset kolme lisää ja uusia laboratorioliitäntöjäkin on lisätty muutamia.

Suurimmat haasteet yhdyskäytävän käyttöön liittyvät itse Bravo-järjestelmään, jonka osalta sen käyttäjien ei ole vielä itsenäisesti mahdollista ottaa yhdyskäytävän kautta kulkevia laboratorio- tai tukkuliitäntöjä järjestelmässään käyttöön. Uusien liitäntöjen käyttöönotto vaatii toistaiseksi vielä kehittäjien apua. Tämä seikka tullaan toivottavasti lähitulevaisuudessa huomioimaan ja järjestämään siten, että Bravo-käyttäjät pääsevät helposti liitäntöihin tarvitessaan käsiksi.

Uusien tukkujen ja laboratorioiden lisääminen yhdyskäytävälle on suhteellisen helppoa, riippuen ainoastaan tukun tai laboratorion yksilöllisistä prosesseista ja niiden monimutkaisuuksista. Seuraava askel yhdyskäytävän ylläpitotehtävien helpottamiseksi on luoda automatisoitu testaus- ja päivitysinfrastruktuuri, joka toivottavasti saadaan tehtyä muutamien seuraavien kuukausien kuluessa. Tämän jälkeen yritys voi alkaa varovasti miettiä mikropalveluarkkitehtuurin laajennusta luvun 5.2 osoittamaan suuntaan.

6 LOPPUPÄÄTELMÄT

Tietojensiirron esiselvitys antoi hyvin osviittaa kolmannen osapuolen toteuttamien tietojensiirtoalustojen kustannuksista, ominaisuuksista ja kehitysnopeudesta verrattuna omatekoisen alustan toteutukseen. Esiselvityksen puitteissa tehdyt testit onnistuivat hyvin ja loppujen lopuksi tilaajayritys päätyi toteuttamaan omat tietojensiirtotyökalut, lähinnä tietojensiirtotarpeiden luonteen ja ulkopuolisten työkalujen hinnoittelun vuoksi. Tuotteet oli hinnoiteltu tilaajayrityksen tarpeisiin nähden melko raskaasti.

Tietojensiirron esiselvityksen puitteissa pääsin tutustumaan kolmannen osapuolen ETL-työkaluihin, joista minulla ei ollut entuudestaan kokemusta. Kokemus oli rikastuttava ja antoi hyvän peruskäsityksen tarjolla olevista vaihtoehdoista. Tämän kokemuksen perusteella on mahdollista arvioida, missä vaiheessa näistä työkaluista voisi alkaa olla hyötyä.

Tietojesiirtoliitännäisen toteutus omatekoiselle alustalle onnistui hyvin ja aikataulussa. Liitännäisen toteutuksen lisäksi työn puitteissa tehtiin tietojensiirto pilottiasiakkaan Alpha-järjestelmästä Bravoon onnistuneesti sovitussa laajuudessa. Tietojensiirtoalustassa on omat puutteensa, mutta koska se on kehitetty talon sisäisesti, sen kehitystä varmasti jatketaan ajan saatossa.

Tietojensiirtoa tehdessä opin, että tietojen lähetystä on mahdollista tehdä useita http-säikeitä pitkin kerrallaan. Tähän käytettiin Curl Multihandle -ominaisuutta. Lisäksi alustan pohjatoteutuksessa käytetty deltaominaisuus tuli tutuksi ja on tulevaisuudessakin tarpeellinen.

Mikropalvelut osoittautuivat tehokkaaksi keinoksi irrottaa vanhaa toiminnallisuutta verkkopohjaisesta järjestelmästä ja saattaa toiminnot useampien järjestelmien saataville. Tämä työn puitteissa rakennettu tilausyhdyskäytävä yhdenmukaisti Alpha- ja Bravo-järjestelmien tilausprosessit lähdejärjestelmien näkökulmasta ja siten helpotti uusien liitäntöjen lisäämistä sekä toi muutamia jo olemassa olleita liitäntöjä Bravo-järjestelmän käytettäväksi. Tilausyhdyskäytävän kaltaisia muita toimintojen siirtoja on tämän raportin kirjoitushetkellä jo tiedostettu ja alettu toteuttaa.

Työn puitteissa mikropalveluarkkitehtuuriin lähemmin tutustuminen oli mielenkiintoista. Työtä tehdessä opin samalla REST-rajapintojen hyviä toteutuskäytäntöjä, Laravel-kehyksen käyttöä sekä supervisor-prosessiohjausten tekoa. Kaikki nämä asiat olivat minulle entuudestaan jonkin verran vieraita. Mikropalvelumalli tarjoaa jatkossakin vielä paljon opittavaa ja odotan siihen jatkossa syventymistä innolla.

7 LÄHDELUETTELO

International Organization for Standardization. 2013. Country Codes - ISO 3166. [Online] 2013. [Viitattu: 10. 05 2019.] https://www.iso.org/iso-3166-country-codes.html.

Izrailevsky, Yury. 2016. Netflix Company Blog. [Online] Netflix Inc., 11. 02 2016. [Viitattu: 18. 04 2019.] https://media.netflix.com/en/company-blog/completing-the-netflix-cloud-migration.

Javlin Ltd. CloverDX. [Online] [Viitattu: 10. 05 2019.] https://www.cloverdx.com/product.

Mauro, Tony. 2015. NGINX Technology Blog. *Adopting Microservices at Netflix: Lessons for Architectural Design.* [Online] NGINX inc., 19. 02 2015. [Viitattu: 18. 04 2019.] https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/.

Newman, Sam. 2015. *Building Microservices.* Boston : O'Reilly Media Inc., 2015.

Otwell, Taylor. 2019. Queues: Laravel Documentation. [Online] 04. 26 2019. [Viitattu: 22. 04 2019.] https://laravel.com/docs/5.8/queues.

Software Testing Help. 2019. 15 Best ETL Tools in 2019. [Online] 08. 05 2019. [Viitattu: 10. 05 2019.] https://www.softwaretestinghelp.com/best-etl-tools/.

Sturgeon, Phil. 2015. *Build APIs You Won't Hate.* s.l. : Philip J. Sturgeon, 2015.

Talend Inc. Talend Data Integration. [Online] [Viitattu: 10. 05 2019.] https://www.talend.com/products/data-integration/.

Weinstein, Jesse;ym. 2013. Eclipse Project. *Eclipse Foundation Wiki.* [Online] 13. 04 2013. [Viitattu: 10. 05 2019.] http://wiki.eclipse.org/Eclipse_Project.

LIITE 1: TIETOJENSIIRTOLIITÄNNÄISEN AJO-OSA

```php
<?php

importPatients();
conversion_backlog::processBacklog($log->id, 0, $limit, 2, $log, $department_url,
$token);
$bravo_patients_array = getPatients(); //Used by journals, appointments and attachments

importAppointments();  // NOTE! ONE TIME RUN ONLY!! DOESN'T SUPPORT DELTA MIGRATIONS!!

importConsultationHistory();
conversion_backlog::processBacklog($log->id, 0, $limit, 10, $log, $department_url,
$token);

// NOTE !! NO DELTA FOR REMINDERS !!!
importReminders();
 conversion_backlog::processBacklog($log->id, 0, $limit, 8, $log, $department_url,
$token);

importAttachments();
conversion_backlog::processBacklog($log->id, 0, $limit, 3, $log, $department_url,
$token);
conversion_backlog::processBacklog($log->id, 0, $limit, 6, $log, $department_url,
$token); // Sends attachments to notes
```

LIITE 2: TIETOJENSIIRRON LÄHDETIETOJEN KÄSITTELY

```php
<?php

function importPatients() {
    global $connection;
    global $base_url;
    // Other variables omitted

    conversion_log_events::logEvent($log->id,'Starting patients');

    $sql = 'SELECT * FROM '.$db_table_prefix.'potilas';

    try {
        foreach($connection->query($sql) as $row) {
            $old_patient_id = $row['id'];
            $name = $row['nimi'];

            $patient_data = array(
                "name" => $name,
                "old_patient_id" => $old_patient_id,
            );

            $new_patient_url = $base_url . 'patient/';
            $method = 'POST';

            $delta =
conversion_delta::CheckDelta(false,$department_url,2,$old_patient_id,array2json($patient_
data),0,0);
            if($delta['delta'] == true) {
                if(!empty($delta['bravo_data_id']) && $delta['bravo_data_id'] != '') {
                    $new_patient_url = $delta['bravo_data_id'];
                    $method = 'PUT';
                }

                $patient_delta_data = $patient_data;
                conversion_backlog::addToBacklog($log->id, 2, 0, $token,
$new_patient_url, $old_patient_id, array2json($patient_delta_data),
array2json($patient_data), $method, time(), 0);
                conversion_log_events::logContinuousEvent($log->id,$patient_log-
>id,'Patients prepared: '.$laskuri.'. Rows processed:'.$row_laskuri.' Latest client id:
'.$old_patient_id);
            }
        }
    }
    catch(PDOException $e){
        echo("SQL Error! ".$e->getMessage()."<br/>"); die();
    }
    conversion_log_events::logEvent($log->id,'Patients complete');
}
```

LIITE 3: SUPERVISORD ASETUKSET

```
[program:ordergw-orders]
process_name=%(program_name)s_%(process_num)02d
command=php72 /path/to/gateway/artisan queue:work purchaseorder --sleep=3 --tries=3
autostart=true
autorestart=true
user=myuser
numprocs=2
redirect_stderr=true
stdout_logfile=/path/to/gateway/queues.log

[program: ordergw-referrals]
process_name=%(program_name)s_%(process_num)02d
command=php72 /path/to/gateway/artisan queue:work referral --sleep=3 --tries=3
autostart=true
autorestart=true
user=myuser
numprocs=2
redirect_stderr=true
stdout_logfile=/path/to/gateway/queues.log

[program: ordergw-referralinformations]
process_name=%(program_name)s_%(process_num)02d
command=php72 /path/to/gateway/artisan queue:work referralinformation --sleep=3 --
tries=3
autostart=true
autorestart=true
user=myuser
numprocs=2
redirect_stderr=true
stdout_logfile=/path/to/gateway/queues.log
```