jamk.fi

Laskutusprosessin tehostaminen järjestelmäintegraation avulla

Mika Lakanen

Opinnäytetyö
Helmikuu 2018
Luonnontieteiden ala
Tradenomi (AMK), tietojenkäsittelyn tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Lakanen, Mika	Opinnäytetyö, AMK	Helmikuu 2018
	Sivumäärä 26	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi

Laskutusprosessin tehostaminen järjestelmäintegraation avulla

Tutkinto-ohjelma

Tietojenkäsittelyn tutkinto-ohjelma

Työn ohjaaja(t)

Tommi Tuikka

Toimeksiantaja(t)

Bitfactor Oy

Tiivistelmä

Tehokas laskutusprosessi on yritykselle kriittinen ja sen merkitys kasvaa edelleen laskumäärien kasvaessa. Tutkimuksen toimeksiantaja on oululainen IT-palveluyritys ja tavoitteena oli kehittää sovellus, joka integroi kaksi käytössä olevaa järjestelmää poistaen toistuvia vaiheita ja tehostaen prosessia.

Tutkimusmenetelmänä käytettiin suunnittelutiedettä, joka pyrkii ratkaisemaan organisatorisen ongelman teknologia-vetoisella ratkaisulla – artefaktilla. Tiedonkeruu tehtiin sähköpostihaastatteluna henkilölle, joka vastaa toimeksiantajayrityksen laskutusprosessista. Näin saatiin kattava kuva prosessin nykyisestä tilasta. Yksityiskohtainen prosessikuvaus osoitti vaiheisiin sisältyvän paljon manuaalista työtä sekä toistuvia työvaiheita. Artefaktin kehitys aloitettiin vaatimusmäärittelyiden jälkeen syksyllä 2017.

Toimeksiantajayrityksen laskutusprosessi on monivaiheinen, ja kehitystyön alussa havaittiin, ettei kaikkia vaiheita voitaisi automatisoida. Automatisoitaviin vaiheisiin tuli kuitenkin verrattuna lähtötilanteeseen merkittäviä parannuksia. Sovellus kehitettiin käyttäen moderneja web-teknologioita, pääasiassa Node.js-sovellusalustaa ja Express-sovelluskehystä.

Tutkimus osoitti, että suhteellisen pienellä vaivannäöllä voidaan saavuttaa merkittäviä parannuksia. Tutkimuksessa myös havaittiin, että pilvipohjaiset järjestelmät voivat olla helppoja integroida REST-ohjelmointirajapintoja hyödyntävällä nk. väliohjelmistolla. Tutkimuksessa kehitetty sovellus pystyi automatisoimaan vain yhden osan monimutkaisesta prosessista, käytössä olleen rajapinnan puutteellisuuden vuoksi.

Avainsanat (asiasanat)

Suunnittelutiede, digitaalinen taloushallinto, järjestelmäintegraatio, web-sovelluskehitys, prosessin kehitys

Muut tiedot (salassa pidettävät liitteet)

jamk.fi

Description

Author(s)	Type of publication	Date
Lakanen, Mika	Bachelor's thesis	February 2018
		Language of publication: Finnish
	Number of pages 26	Permission for web publication: x

Title of publication

Improving billing process with enterprise application integration

Degree programme

Business Information Systems

Supervisor(s)

Tuikka, Tommi

Assigned by

Bitfactor Ltd

Abstract

Effective billing process is critical to the business and its importance grows as more bills are sent. The study was assigned by an Oulu-based IT-service company, with the aim to develop an application that integrates two existing systems thus eliminating repetition and enhancing the process.

The research method used was design science, which aims to solve the organizational problem with a technology-based solution – an artifact. The person responsible for the invoicing process was interviewed by email, which provided a comprehensive picture of the current state of the process. A detailed description of the process showed that there were plenty of manual work and recurring phases. The development of the artifact started after the requirements specification was obtained from the client in autumn 2017.

The billing process in the client company is multi-stage, and at the beginning of the development it became clear that not all phases can be automated; however, those that can will be significantly enhanced. The application was developed using modern web technologies, mainly Node.js and Express.

The study showed that a relatively small amount of effort can achieve great benefits and that cloud-based systems can be easily integrated with middleware utilizing REST interfaces, although the developed application was able to automate only a small part of the complex process due to the lack of interface used.

Keywords/tags (subjects)

Digital financial management, billing process, application integration, design science

Miscellaneous (Confidential information)

1 Sisältö

Kuviot

Käsitteet

API (Application Programming Interface): Ohjelmointirajapinta. Sen tehtävänä on määritellä sovellukselle tai järjestelmälle väylä, jonka kautta ulkopuolelta voidaan pyytää tai lähettää siihen tietoja.

Artefakti: Ihmisen tietoisen toiminnan tuloksena syntyvä esine tai asia, kuten sovellus. Suunnittelutieteellisen tutkimuksen tulosta kutsutaan artefaktiksi.

Express: Suosittu sovelluskehys Node.js-ympäristöön.

Järjestelmäintegraatio (engl. EAI): Kahden tai useamman järjestelmän yhdistämistä keskenään, jotta niiden välillä voidaan vaihtaa tietoa.

Node.js: Google Chromen V8 JavaScript -moottorin päälle rakennettu palvelimella ajettava sovellusalusta.

OAuth: Valtuutusprotokolla, jota käytetään, halutessa päästä käsiksi yksityisiin tietoihin ohjelmointirajapinnan kautta.

REST (Representational State Transfer): Hypermediajärjestelmän (ohjelmointirajapinnan) arkkitehtuurinen tyyli.

Väliohjelmisto (engl. middleware): Sovellus, joka toimii tiedon välittäjänä kahden tai useamman muun sovelluksen välillä.

1 Johdanto

Tässä tutkimuksessa tarkastellaan, kuinka web-teknologioita sekä järjestelmäintegraatiota hyödyntämällä voidaan tehostaa yrityksen kriittistä toimintoa: laskutusprosessia. Kyseessä on laadullinen, yrityksen prosessiin kohdistuva suunnittelutieteellinen tutkimus, jonka toimeksiantaja on oululainen IT-palveluyritys. Suunnittelutieteellisessä tutkimuksessa yrityksen ongelmaa ratkaistaan teknologiavetoisesti ja sen tuloksena syntyy esimerkiksi sovellus, kuten tässä tutkimuksessa.

Digitaalinen taloushallinto ja automatisaatio tuovat mukanaan paljon mahdollisuuksia. Käytössä olevien järjestelmien lisääntyessä on kuitenkin hyvä pysähtyä pohtimaan, kuinka saadaan paras mahdollinen hyöty kaikista uusista työkaluista, kuormittamatta liikaa työntekijöitä. Järjestelmäintegraatio on yksi tapa vähentää eri työvaiheissa tapahtuvaa toistoa ja manuaalista työtä. Johtamisen näkökulmasta taloushallinnon sähköistäminen tuo uusia mahdollisuuksia, kun yrityksen strategiseen suunnitteluun voidaan ottaa ajantasaisempaa informaatiota yrityksen taloudesta.

Tämän tutkimuksen tavoitteena on kehittää järjestelmäintegraation väliohjelmisto web-teknologioita hyödyntämällä ja sen avulla tehostaa yrityksen laskutusprosessia, vähentämällä toistuvia työvaiheita. Tutkimuksen lopuksi tarkastellaan prosessin tehokkuutta ilman sovellusta ja sovelluksen kanssa sekä kuinka sovellusta voisi vielä jatkokehittää.

2 Teoreettinen viitekehys

Tässä luvussa käsitellään tutkimukseen liittyviä, keskeisiä käsitteitä ja näkökulmaa josta tutkittavaa ilmiötä tarkastellaan.

2.1 Taloushallinto

Taloushallinto hallinnoi ja tarkkailee yrityksen taloudellisia resursseja. Sen tehtäviin kuuluu mm. palkanlaskenta, tilintarkastus ja erilaiset lakisääteiset velvoitteet, kuten

kirjanpito. Taloushallinto tuottaa tärkeää, yrityksen talouteen liittyvää tietoa johdolle. Tietoa tuotetaan myös ulkoiseen käyttöön, kuten esimerkiksi viranomaisille ja muille organisaation ulkopuolisille tahoille. (Taloushallinto 2018.)

2.1.1 Digitaalinen taloushallinto

Digitaalinen taloushallinto on laaja käsite, joka kätkee sisälleen useita yrityksen prosesseja, käytäntöjä ja tietojärjestelmiä, jotka liittyvät yrityksen talouteen, kuten esimerkiksi laskutus, maksuliikenne, kirjanpito ja palkanlaskenta. Yhdistävänä tekijänä eri komponenteilla on kuitenkin se, että tieto säilytetään ja käsitellään sähköisessä muodossa. Turhat työvaiheet pyritään poistamaan automatisoinnilla. (Lahti & Salminen 2014, 16–25.) Digitaalisessa taloushallinnossa yrityksen johto saa reaaliaikaista ja yksityiskohtaista informaatiota yrityksen taloudellisesta tilanteesta, mikä mahdollistaa nopean reagoinnin ja uudenlaisen johtamisen (Sähköinen taloushallinto n.d.).

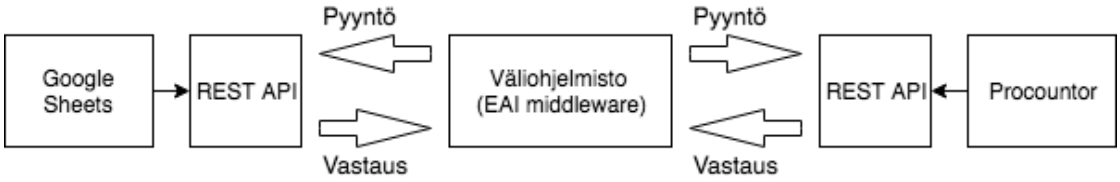
Hyvin toimivalla integraatiolla on merkittävä rooli taloushallinnon tehokkuudessa, mutta myös yrityksen valinnoilla, liittyen taloushallinnon järjestelmiin ja palveluihin on suora vaikutus tehokkuuteen, minkä vuoksi valintojen tulisi heijastaa yrityksen strategiaa (Lahti & Salminen 2014, 34–42). Tämän tutkimuksen tuotoksena syntynyt sovellus pyrkii poistamaan turhia työvaiheita ja siten tehostamaan prosessia.

2.2 Järjestelmäintegraatio

Järjestelmäintegraatiossa (engl. Enterprise Application Integration, EAI) kytketään kaksi tai useampia tietojärjestelmiä tai ohjelmistoja yhteen siten, että niiden välillä voidaan vaihtaa tietoa. Tämä on mahdollista toteuttaa monella tapaa ja integroidut järjestelmät voivat olla löyhästi tai tiukasti toisiinsa kytkettyjä. Hyvin tiukka integrointi ei ole aina mahdollista eikä välttämättä kannattavaa, koska silloin osapuolet ovat toisistaan myöskin tiukasti riippuvaisia ja jokaisen muutoksen yhteydessä, täytyy molemmat järjestelmät ottaa huomioon (Linthicum, 2003).

Löyhän riippuvuuden integraatio voidaan toteuttaa esimerkiksi hyödyntämällä järjestelmien tarjoamia ohjelmointirajapintoja ja väliohjelmistoa. Väliohjelmistolla (engl. middleware) tarkoitetaan ohjelmistoa, joka mahdollistaa kahden tai useamman järjestelmän kommunikoinnin keskenään (Ruh, Maginnis & Brown, 2001).

Kuvio 1 havainnollistaa tiedon kulkua integroitavien järjestelmien ohjelmointirajapintojen ja väliohjelmiston välillä. Tässä tutkimuksessa kehitetään paikallisesti tietokoneella ajettava Node.js-web-sovellus, joka kommunikoi kahden järjestelmän välillä, käyttäen niiden REST-ohjelmointirajapintoja. Sovellusta hallitaan käyttöliittymästä internetselaimella.



Kuvio 1. Kaksi järjestelmää integroiva väliohjelmisto

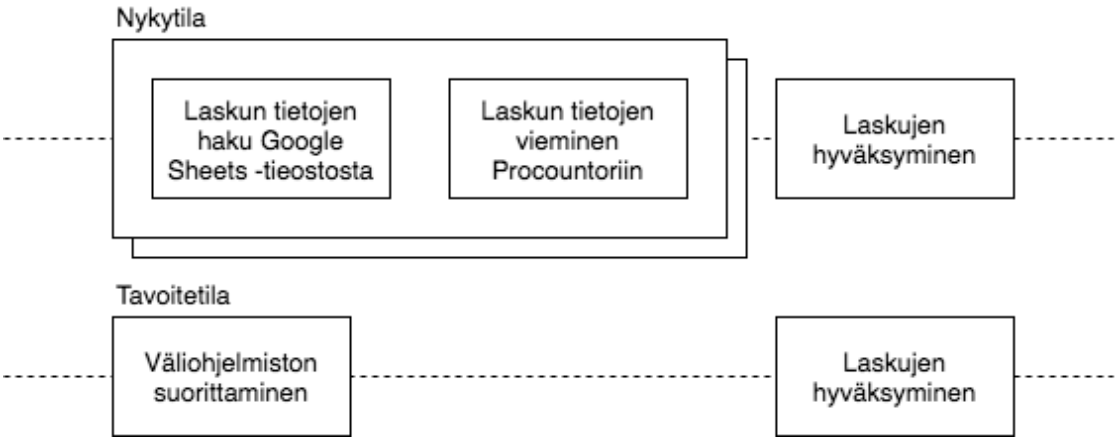
Järjestelmäintegraatiolla pyritään tehostamaan liiketoimintaprosessia, jossa käytetään monesta eri lähteestä saatavaa informaatiota. Prosessin tehostaminen merkitsee kustannussäästöjä ja kilpailukyvyn kasvua, etenkin isommissa yrityksissä, joissa prosessit voivat olla monimutkaisia. Järjestelmäintegraatiolla varaudutaan myös tulevaisuuden muutoksia varten ja mahdollistetaan nopea reagointi isoihin muutoksiin (esim. fuusiot) sekä vähennetään riippuvuutta yksittäisistä ohjelmistoista. (Tähtinen 2005, 23–27.)

Integraatioratkaisut pilvessä

SaaS (Software as a Service) -malli on muuttanut ajattelumaailmaa ja tarjoaa uuden tavan toimittaa integraatioratkaisuja. Aikaisemmin sovelluksista tehtiin kopioita ja yrityksellä saattoi olla samasta integraatiosovelluksesta useita kopioita. Tällaisen ratkaisun ylläpitäminen on hankalaa, koska jokainen sovellus täytyy päivittää erikseen. SaaS-mallissa sovelluksia on yksi, joka kustomoidaan asiakkaan tarpeisiin sopivaksi. (Moul 2010.)

3 Tutkimusasetelma

Tässä luvussa tarkastellaan tutkimuksen tavoitetta, tutkimusmenetelmiä sekä ongelmaa, joka yritetään ratkaista. Tutkimuksen tavoitteena on kehittää toimeksiantajayrityksen sisäistä prosessia kehittämällä tiettyjä laskutusprosessiin liittyviä toimintoja tehostava EAI-sovellus. Kuviossa 2 voidaan nähdä automatisoitava osa prosessista kaaviona. Nykyisellään ensimmäinen vaihe toistetaan käsin niin monta kertaa, kuin laskurivejä on valmiina Google Sheets -tiedostossa. Tavoitetilassa suoritettava sovellus hoitaa toiston ihmisen puolesta.



Kuvio 2. Prosessikaavio nykytilasta ja tavoitetilasta

3.1 Tutkimusongelma ja -kysymykset

Tutkimusongelmaksi voidaan nähdä suuri määrä manuaalisesti tehtävää työtä, kun työntekijä kopioi tietoja internetselaimessa, kahden järjestelmän välillä. Lähemmin tarkasteltuna havaitaan ongelman olevan itse asiassa siinä, että nuo kaksi järjestelmää eivät kommunikoi keskenään, minkä vuoksi tieto täytyy kopioida käsin järjestelmästä toiseen.

Tutkimuskysymys muotoutui seuraavanlaiseksi:

Kuinka paljon järjestelmäintegraatio tehostaa yrityksen laskutusprosessia?

3.2 Tutkimusmenetelmät

Tutkimusmenetelmänä käytetään suunnittelutiedettä (engl. design science), jossa organisaation ongelmaa ratkaistaan teknologia-vetoisilla ratkaisuilla. Suunnittelutieteellisen tutkimuksen tuloksena syntyy IT-artefakti – tässä tapauksessa web-sovellus –, joka ratkaisee organisatorisen ongelman. Artefaktilla tarkoitetaan ihmisen tietoisen toiminnan tuotosta (Lång 1998).

3.2.1 Suunnittelutiede (design science)

Suunnittelutieteellisen tutkimuksen päämääränä on kehittää teknologiaan perustuvia ratkaisuja oleellisiin liiketoiminnan ongelmiin. Tutkimuksen tulee tuottaa elinkelpoinen sovellus, jonka hyödyllisyys, tehokkuus ja laatu täytyy tieteellisin menetelmin osoittaa. (Hevner, March, Park & Ram 2004.)

Suunnittelutieteellisen tutkimuksen tekemiselle antavat Hevner ja muut (2004) seuraavat seitsemän ohjetta:

1. artefaktin tuottaminen
2. ongelman oleellisuus
3. suunnitellun ratkaisun arviointi
4. tutkimuksen tuottama hyöty tiedeyhteisölle
5. täsmälliset menetelmät
6. suunnittelu etsimisprosessina
7. tutkimustyön kommunikointi.

Artefaktin tuottaminen

Suunnittelutieteellisen tutkimuksen tuloksena syntyy tarkoituksenmukainen IT-artefakti (myöhemmin sovellus), joka ratkaisee ongelman. Esimerkiksi prosessia kehittävä ohjelmisto, kuten tässä tutkimuksessa. Artefaktiin katsotaan liittyvän myös mahdolliset menetelmät ja mallit, jotka syntyvät tutkimuksen tuloksena.

Ongelman oleellisuus

Tavoitteena on saada tutkittavasta ilmiöstä, tärkeästä liiketoiminnan ongelmasta ymmärrys, jonka avulla ongelma voidaan ratkaista teknologiaan perustuvilla menetelmillä.

Suunnitellun ratkaisun arviointi

Tutkimuksen tuloksena syntynyt sovellus arvioidaan sen toimintaympäristöön soveltuvilla arviointikriteereillä ja -menetelmillä. Sovellusta voidaan arvioida mm. seuraavien ominaisuuksien perusteella: toimivuus, suorituskyky, luotettavuus, käytettävyys sekä soveltuvuus kyseiselle yritykselle.

Tutkimuksen tuottama hyöty tiedeyhteisölle

Kuten tutkimuksessa yleensä, myös suunnittelutieteellisen tutkimuksen täytyy tuottaa uutta ja mielenkiintoista tietoa tiedeyhteisölle. Usein tutkimuksen tärkein tuotos on itse sovellus, joka voi käyttää olemassa olevaa tietoa tai yhdistellä sitä innovatiivisesti.

Täsmälliset menetelmät

Sekä sovelluksen kehittämisessä, että myöskin arvioinnissa käytetään tarkkoja, yleensä matemaattisia menetelmiä. Tarkkuus saavutetaan hyödyntämällä olemassa olevaa tietoa ja menetelmiä. Arviointi kohdistuu siihen, kuinka hyvin sovellus toimii.

Suunnittelu etsimisprosessina

Suunnittelutieteellisen tutkimuksen voidaan katsoa olevan iteratiivinen etsimisprosessi ratkaisun löytämiseksi ongelmaan.

Tutkimustyön kommunikointi

Suunnittelutieteellinen tutkimus täytyy esittää siten, että sen ymmärtää niin teknisesti, kuin myös hallinnollisesti orientoitunut yleisö. Tutkimuksen esittelyn täytyy olla riittävän yksityiskohtainen, jotta sovellus voidaan kuvailun avulla rakentaa ja implementoida sekä tutkimuksen tulokset toistaa. Myös hallinnollisesta näkökulmasta tarvitaan tietoa, jotta tutkimuksen perusteella voidaan tehdä päätös, kannattaako sovellusta ottaa käyttöön.

3.3 Tutkimuksen rajaus

Tutkimus rajautuu järjestelmäintegraation toteuttavan web-sovelluksen kehittämiseen määrätyillä teknologioilla sekä prosessin tehostumisen tutkimiseen. Toimeksiantaja on oululainen, vuonna 2013 perustettu IT-palveluyritys. Kehitetyn sovelluksen hyödyllisyyttä ja laatua tarkastellaan tutkimuksen lopuksi tuloksissa.

Tiedonkeruu toteutettiin sähköpostihaastatteluna, johon osallistui yrityksen laskutusprosessista vastaava henkilö, sillä hän on lähimpänä tutkittavaa ilmiötä ja osaa kuvailla prosessin vaiheita parhaiten.

4 Laskutusprosessin kehittäminen

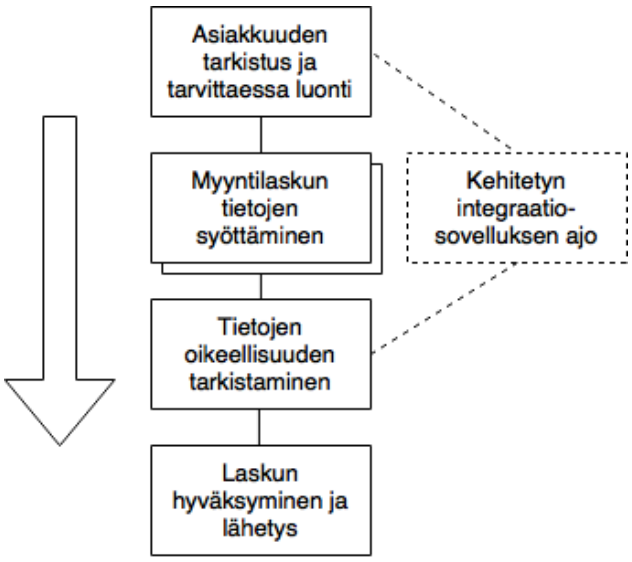
Tehokas laskutusprosessi on yritykselle kriittisen tärkeää, etenkin jos laskuja lähetetään paljon. On yleistä, että yrityksellä on käytössään kaksi järjestelmää: toisesta katsotaan laskutettavan tiedot sekä varsinainen laskutusjärjestelmä, jonne tiedot syötetään. Käytännössä siis tehdään käsin työtä, joka olisi automatisoitavissa. (Lahti & Salminen 2014, 78–85.)

Tässä tutkimuksessa tutkittava laskutusprosessi on monivaiheinen:

1. Procountorista tarkistetaan, löytyykö asiakkuus asiakasrekisteristä ja tarvittaessa luodaan se

2. haetaan laskulle tulevat tiedot Google Sheets -tiedostosta ja viedään Procountoriin uudelle laskulle (vaihe toistetaan niin monta kertaa, kuin laskurivejä on valmiina)
3. varmistetaan tietojen oikeellisuus
4. hyväksytään lasku ja lähetetään se valitulla laskutuskanavalla.

Procountorin ohjelmistokehittäjille tarjoama ohjelmointirajapinta ei anna mahdollisuutta toteuttaa ensimmäisen vaiheen automatisointia, joten tutkimuksessa kehitettävä sovellus keskittyy automatisoimaan toista vaihetta, eli laskun tietojen siirtämistä Google Sheets -palvelusta Procountoriin. Kuviossa 3 on havainnollistettu katkoviivoin tutkimuksen tuloksena syntyvää uutta vaihetta koko prosessin näkökulmasta.



Kuvio 3. Automatisoitava osa prosessista

Tässä tutkimuksessa syntyneen paikallisen web-sovelluksen kehityksessä käytetyt teknologiat määräytyivät toimeksiantajan kanssa käydyn keskustelun perusteella ja heijastavat yrityksen asiakasprojekteissaan käyttämiä työkaluja sekä alalla vallitsevia trendejä.

4.1 Node.js

Tutkimuksessa kehitetty sovellus kehitettiin Node.js-ympäristössä käyttäen JavaScript-ohjelmointikieltä ja siinä hyödynnetään npm-moduuleja. Node.js on Google Chromen V8 JavaScript -moottorin päälle kehitetty JavaScript-ajoympäristö (engl. runtime), jonka npm-työkalulla päästään käsiksi “maailman suurimpaan avoimen lähdekoodin kirjasto-ekosysteemiin”. (Node.js n.d.)

Ei-blokkaavat kutsut

Blokkaamisella (engl. blocking) Node.js:ssä tarkoitetaan sitä, kun ohjelman prosessin suoritus keskeytyy siksi aikaa, kunnes ei-JavaScript operaatiot ovat suoritettu. Blokkaamisen aiheuttaa yleensä nk. I/O-operaatio, eli ohjelman interaktio kiintolevyn tai verkon kanssa kuten esimerkiksi tiedostoon kirjoittaminen tai internetistä tiedon hakeminen. (Node.js n.d.)

Node.js:n kaikista I/O-metodeista löytyy myös asynkroniset versiot jotka ovat ei-blokkaavia, eli ne eivät keskeytä sovelluksen suorittamista. Asynkronisille funktioille annetaan argumenttina toinen funktio (nk. callback-funktio), joka käsittelee ensimmäisen funktion paluuarvon (Gackenheimer, 2013).

Event Loop

Node.js (n.d.) kuvailee kotisivujensa dokumentaatiossa ominaisuutta, joka mahdollistaa ei-blokkaavat I/O-operaatiot. Ominaisuuden nimi on event loop, eli tapahtumasilmukka. Tapahtumasilmukan ja asynkronisen toiminnallisuuden tarjoaa alustaan implementoitu libuv-niminen C-kirjasto. Silmukka koostuu useasta eri vaiheesta, jossa jokaisessa suoritetaan niihin kuuluvat operaatiot:

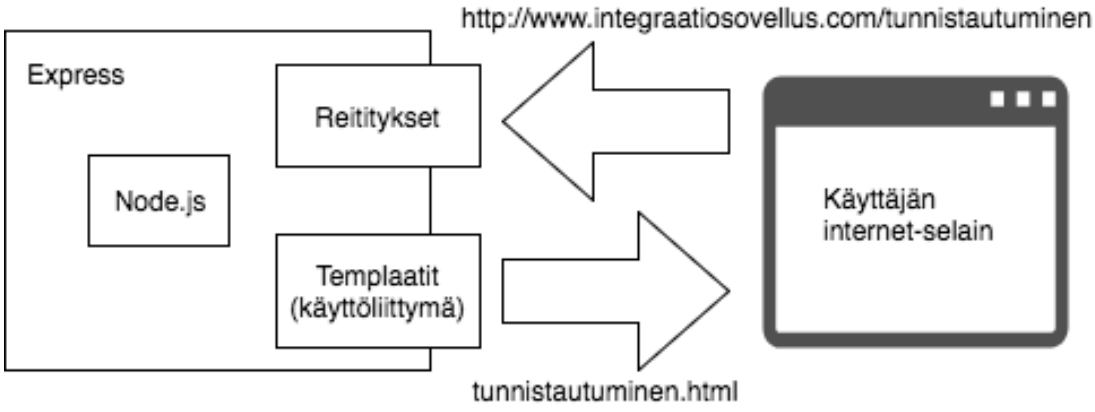
- **timers:** Callback-funktiot, jotka on asetettu setTimeout() tai setInterval()-metodeilla.
- **I/O callbacks:** Lähes kaikki callback-funktiot, paitsi timers- tai close callbacks-vaiheessa olevat, tai setImmediate()-metodilla asetetut operaatiot.
- **idle, prepare:** Node.js:n sisäiseen käyttöön.
- **poll:** I/O-operaatioiden noutaminen tai blokkaus.

- **check:** setImmediate()-metodin callback-funktioiden suorittaminen.
- **close callbacks**

4.2 Express

Express on suosittu ja monipuolinen web- ja mobiilisovellusten kehitykseen suunniteltu sovelluskehys Node.js:lle (Express n.d.). Express on minimalistinen sovelluskehys, joka antaa kehittäjälle vapauden täydentää sovellusta erilaisilla väliohjelmistoilla, jotka käsittelevät saapuvat pyynnöt. Express pohjautuu Node.js:n http-moduuliin ja tekee useiden toimintojen käyttämisestä helpompaa. Se mahdollistaa myös esimerkiksi MVC-mallin (Model View Controller) rakenteen käyttämisen sovelluksessa. Sovellus voi olla esimerkiksi REST-arkkitehtuurin rajoitteet täyttävä ohjelmointirajapinta, tai muu web-sovellus. (Mardan 2014.)

Express-sovelluskehystä käytettiin tässä tutkimuksessa tarjoamaan sovellukselle rakenne ja käyttöliittymä sekä käsittelemään saapuvat vastaukset eri järjestelmistä reititysten avulla. Reitityksellä tarkoitetaan eri osoitteisiin tulevien HTTP-pyyntöjen käsittelemistä kyseisen reitityksen määräysten mukaisesti (Kuvio 4).



Kuvio 4. Express-sovelluskehyksen toiminta yksinkertaistettuna

Express-sovellukselle määritellään mm. väliohjelmat (engl. middleware), stattiset hakemistot, parsijat, virheenkäsittelijät, ohjain-funktiot (engl. controller) ja reitit (engl. routes). Ajon aikana Express-sovellus kuuntelee HTTP-pyyntöjä ja käsittelee ne määriteltyjen reittien ja väliohjelmien mukaisesti järjestyksessä ylhäältä alaspäin. (Mardan 2014.)

Kuviossa 5 määritellään väliohjelmat, jotka pätevät määrätyille reiteille. Jokaisella reitillä on oma HTTP-metodi, kuten GET tai POST sekä ohjain-funktio. Ohjain-funktio pätee vain määriteltyyn metodiin ja hoitaa siihen vastaamisen tai hylkäyksen. Vastaus voi olla esimerkiksi web-sivun renderöinti määritellyllä templaatti-moottorilla, kuten tässätapauksessa käytetty Handlebars. Viimeisenä määritelty väliohjelma ottaa vastaan kaikki muut, kuin ennalta määrätyt reitit ja antaa virheen 404, ”sivua ei löydy”.

```
5  module.exports = function(router) {
6
7      // Index, main view
8      router.get("/", controllers.index);
9
10     // Google sign in (for more info check: https://github.com/google/google-api-nodejs-client)
11     router.get("/auth/google", controllers.auth_google);
12
13     // Google callback (Exchange auth code to access token and get profile name)
14     router.get("/auth/google/callback", controllers.auth_google_callback);
15
16     // Google sign out
17     router.get("/auth/google/signout", controllers.auth_google_signout);
18
19     // Get sheet from google and convert to JSON
20     router.post("/getsheet/", controllers.getsheet);
21
22     // Error route that receives an error message as querystring and sends it as response
23     router.get("/error", controllers.error);
24
25     // Authenticate Procountor, send invoice and then return results
26     router.post("/generate", controllers.generate)
27
28     // In case of 404
29     router.use(controllers.not_found);
30
31 }
```

Kuvio 5. Reittien määrittely

4.3 REST

Tässä tutkimuksessa kehitetty sovellus käyttää tiedon noutamiseen ja lähetykseen Googlen ja Procountorin tarjoamia REST-ohjelmointirajapintoja. REST (Representational State Transfer) termin esitti ensimmäisenä Roy Fielding, vuonna 2000 väitöskirjassaan, kuvaillessaan ja määritellessään hajautettujen hypermediajärjestelmien arkkitehtuurista tyyliä ja rajoitteita, joita noudattamalla järjestelmän voidaan katsoa noudattavan REST-periaatteita. REST on jalostettu muista tietoverkkojen arkkitehtuurisista tyyleistä, silmällä pitäen järjestelmän tarpeita ja lisäten rajoitteita, jotka muodostavat yhtenäisen rajapinnan. Sen voidaan katsoa olevan arkkitehtuuristen elementtien abstraktio, joka ei keskity komponenttien yksityiskohtiin tai syntaksiin, vaan komponenttien rooleihin ja niiden keskinäisen interaktion rajoituksiin.

Fieldingin, 2000, kuvailemat kuusi rajoitetta, jotka muodostavat REST-tyylin arkkitehtuurin ovat:

Asiakas-palvelin (Client-Server)

Asiakas-palvelin on arkkitehtuurinen tyyli, jossa selkeällä roolijaolla erotetaan toisistaan käyttäjä (asiakas), joka vastaa käyttöliittymästä sekä palvelin, joka tarjoaa palveluja. Asiakas lähettää palvelimelle pyynnön ja palvelin joko hylkää pyynnön tai käsittelee sen ja palauttaa vastauksen. Roolijaon ansiosta palvelinkomponentti pysyy yksinkertaisena ja on näin ollen paremmin skaalautuva.

Tilaton (Stateless)

Palvelin ei saa säilyttää sessioon liittyviä tietoja, kuten tietoja asiakkaan aikaisemmista pyynnöistä, vaan jokaisen asiakkaalta tulevan pyynnön tulee sisältää kaikki tarvittava informaatio, jonka palvelin tarvitsee pyynnön käsittelyyn. Tämä parantaa luotettavuutta ja skaalautuvuutta, mutta toisaalta tarkoittaa sitä, että samaa tietoa lähetetään useaan kertaan toistuvissa pyynnöissä.

Välimuisti (Cache)

Tietoverkon toiminnan tehostamiseksi jokaisen pyynnön vastaukseen sisällytetään tieto, joka ilmaisee, onko vastaus luvallista säilyttää välimuistissa. Näin voidaan säästää resursseja ja parantaa skaalautuvuutta, kun samanlaisiin pyyntöihin saadaan

suora vastaus välimuistista. Haittapuolena rajoitteessa on, että tieto voi välimuistissa joskus olla vanhentunutta.

Yhdenmukainen rajapinta (Uniform Interface)

REST painottaa yhdenmukaisen rajapinnan tärkeyttä komponenttien välillä ja sen saavuttaakseen, REST määrittelee neljä erillistä lisärajoitetta rajapinnalle:

- resurssien tunnistaminen
- resurssien manipulaatio representaatioiden kautta
- itsekuvaavat viestit
- hypermedia sovelluksen tilan moottorina.

Kaikki tieto tai data, kuten kuvat ja dokumentit voidaan katsoa olevan resursseja tai kokoelmia resursseista. Jokaisella resurssilla on oma tunnisteensa tiedon hakemista ja manipulointia varten. Resursseja manipuloidaan nk. representaatioiden kautta. Representaatio on komponenttien (asiakas ja palvelin) välillä siirtyvä kuvaus resurssin nykytilasta tai halutusta tilasta, yleensä HTTP-pyynnön mukana menevä viesti tai pyynnön vastauksen sisältönä oleva informaatio.

Itsekuvaavat viestit liittyvät tilattomuuteen. Koska asiakkaan ja palvelimen interaktiosta ei pidetä kirjaa, täytyy jokaisen uuden pyynnön olla tarpeeksi kuvaileva, jotta se voidaan ymmärtää.

Viimeisessä rajapintaan liittyvässä erityisrajoitteessa, Hypermedia sovelluksen tilan moottorina (engl. Hypermedia As The Engine Of Application State, HATEOAS), keskitytään hypermedia-linkkeihin. REST-rajapinnan resurssien formaattiin ei oteta kantaa, ne voivat olla esimerkiksi JSON tai XML-muodossa, mutta rajapinnasta palautuvien vastausten tulee sisältää resurssiin linkittyvien muiden resurssien hyperlinkit ja suhdekuvaukset näiden välillä. (Understanding HATEOAS n.d.)

Kerroksittainen järjestelmä (Layered System)

Kerroksittainen järjestelmä on hierarkinen. Jokainen taso käyttää alemman tason palveluita ja samalla tarjoaa palveluita ylemmälle kerrokselle. Koska palveluita ei tarjota usean kerroksen ylitse, säilyy riippumattomuus ja uudelleenkäytettävyys paranee. Kerroksittaisen järjestelmän haittapuolena on prosessin viive.

Ohjelmiston laajentaminen (Code-On-Demand)

Viimeinen rajoite, jota ei ole pakko käyttää, antaa mahdollisuuden ladata ja suorittaa ohjelmistokoodia esimerkiksi skriptien muodossa ja näin ollen laajentaa asiakassovelluksen toiminnallisuutta.

5 Tutkimuksen toteuttaminen

Tutkimus aloitettiin toimeksiannon ja vaatimusmäärittelyn jälkeen syyskuussa 2017. Haastattelulomake lähetettiin sähköpostilla toimeksiantajayrityksen laskutusprosessista vastaavalle henkilölle sisältäen seuraavat kysymykset:

- kuvaile laskutusprosessia ja siihen liittyviä vaiheita mahdollisimman tarkasti
- kuinka tehokas prosessi mielestäsi on (asteikolla 1–10)
- perustele arvosana.

Lisäksi tiedusteltiin:

- kuinka kauan yhden laskurivin lähettäminen kestää
- kuinka monta laskua lähetetään kuukaudessa keskimäärin.

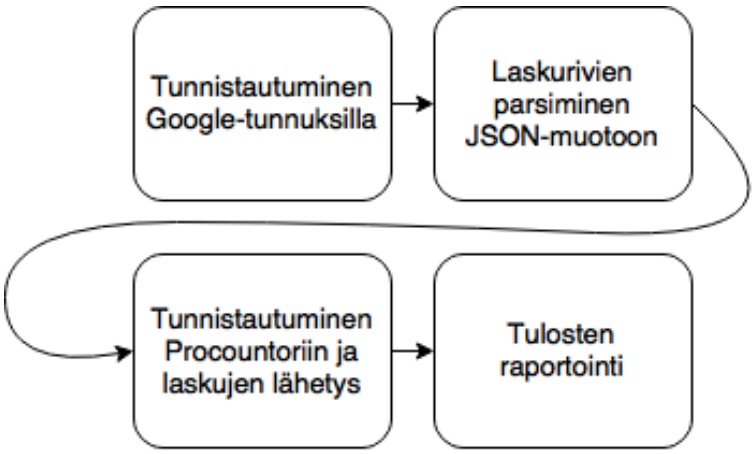
Näillä tiedoilla saatiin kattava kuva prosessista ja sen eri vaiheista sekä pystyttiin laskemaan työmäärä, joka laskujen lähettämiseen kuluu.

Yhteydenpito toimeksiantajaan

Kehitystyössä käytettiin Git-versionhallintaohjelmistoa ja GitHub-palvelua, jonne ohjelmisto varmuuskopioitiin, ja jossa ylläpidettiin päiväkirjaa sovelluksen edistymisestä. Näin toimeksiantaja pystyi seuraamaan tarkasti kehitystyön eri vaiheita.

Sovelluskehityksen aloitus

Kehitettävän sovelluksen toiminta tuli olemaan pääasiassa kahden järjestelmän ja niiden rajapintojen kanssa kommunikointia ja sovelluksen toiminta voidaan kiteyttää Kuvion 6 mukaisesti.



Kuvio 6. Sovelluksen toiminta

Ennen varsinaista ohjelmoinnin aloittamista oli Google API Consolessa perustettava projekti ja konfiguroitava tunnistautumiseen liittyvät asetukset, jotta yksityisiin Google-tilin tietoihin pääsee sovelluksesta käsiksi. Projektille aktivoitiin halutut Googlen palvelut – tässä tapauksessa Google Sheets API laskurivien noutamista varten ja Google+ API käyttäjän nimen ilmoittamista käyttöliittymässä varten. Procountorin testiohjelmointirajapintaan pyydettiin oikeus, ja tunnukset palvelun www-sivuilta löytyvällä lomakkeella.

5.1 Valtuutus (OAuth)

OAuth 2.0 on vuonna 2006 kehitetyn OAuth-valtuutusprotokollan seuraaja (OAuth 2.0 2018). Sekä Googlen Sheets API, että myös Procountorin API käyttävät valtuuttamisessa OAuth-protokollaa.

Tunnistautuminen ja valtuutus

Tunnistautumisella tai tunnistautumisprotokollalla tarkoitetaan sitä, kun sovellukselle kerrotaan, kuka käyttäjä on kyseessä ja esimerkiksi onko tämä kirjautuneena vai ei, tai muita käyttäjään liittyviä tietoja. OAuth-protokollassa käyttäjää ei tunnisteta, eikä käyttäjästä tiedetä mitään. OAuth keskittyy vain valtuutuksen tarkistamiseen, joka tehdään nk. token-koodilla. Valtuutusprotokollassa siis esimerkiksi lupa käyttää

rajapintaa voidaan antaa, vaikka käyttäjä ei olisi itse paikalla. Käyttäjän tunnistaminen on kuitenkin yleensä yhtenä osana sovelluksen valtuutusprosessia. (Richer n.d.)

OAuth-protokollassa API-pyyntöjen mukana lähetetään palvelimelle tunnuksen ja salasanan sijasta ”access token”, joka edustaa käyttäjää ja näin olleen pääsee käsiksi rajatusti yksityisiin tietoihin ohjelmointirajapinnan kautta (Spasovski, 2013). Tässä tutkimuksessa kehitetyn sovelluksen kannalta oleellista on päästä yksityisiin Google Sheets -tiedostoihin käsiksi, joissa yrityksen asiakasprojektien tuntikirjaukset sijaitsevat.

5.2 Node.js-moduulien hyödyntäminen

Node.js-alustan laaja käyttäjäyhteisö tarjoaa kattavan määrän nk. moduuleja, joita voi vapaasti käyttää omissa projekteissaan. Myös Google tarjoaa kirjastoja eri ohjelmointikielille, jotka tekevät ohjelmointirajapintojen kanssa työskentelystä helpompaa. Googlen virallisesti tukemaa Node.js Client -moduulia käytettiin Googlen palveluihin tunnistautumisessa.

Kehitetty sovellus hakee tiedon, eli laskurivit, Google Sheets -tiedostosta, jonka jälkeen se “parsitaan” Procountorin rajapinnalle sopivaan JSON-muotoon. Tämän toimenpiteen suorittaa siihen tarkoitukseen kehitetty ja npm-palvelusta löytyvä google-spreadsheet-to-json -moduuli. Kuvio 7 kuvaa pelkistettynä yhden JSON-laskurivin, joka liitetään Procountorin rajapinnalle lähetettävään HTTP POST -pyyntöön.

```
{
    "asiakkaan_nimi":    "Mulperin Karkkikauppa Oy",
    "asiakasnumero":     1,
    "laskutusosoite":    "Naavatie 20 B 15, 62900 Alajärvi",
    "summa":             500
}
```

Kuvio 7. Laskun tiedot JSON-muodossa

5.3 Tulosten raportointi

Procountorin ohjelmointirajapintaan tunnistautumisen sekä myyntilaskujen muodostamisen jälkeen viimeinen vaihe sovelluksen toiminnassa on tulosten raportointi käyttöliittymään ja Google Sheets -tiedostoon. Procountorin ohjelmointirajapinta palauttaa onnistuneen lähetyksen päätteeksi vastaus-objektin, joka sisältää laskun numeron. Tämä numero kirjataan Google Sheets -riville, kyseisen laskurivin invoice_number -kenttään.

Sovellus tarkistaa ennen laskun lähetystä, tietojen haku vaiheessa, löytyykö laskurivin invoice_number -kentästä tietoja ja jos löytyy, rivin tietoja ei lähetetä Procountorille. Käytännössä invoice_number -kentässä pidetään esimerkiksi teksti “kesken”, jos laskua ei vielä haluta laskuajossa lähettää ja pyyhitään tyhjäksi, mikäli rivi on valmis laskutettavaksi.

5.4 Kohdatut haasteet

Node.js:n asynkroninen tapa suorittaa koodia ja funktioita on tehokas, mutta vaatii esimerkiksi rajapintoihin useita kertoja pyyntöjä lähetettäessä, hieman erityispohdintaa. Molemmat käytetyt ohjelmointirajapinnat ovat suojattu rajoituksella, joka estää liian usean pyynnön tekemisen liian lyhyessä ajassa (engl. QPS – Queries Per Second). Jos laskuja lähetetään kerralla paljon, täytyy se ottaa huomioon ohjelmakoodissa, koska Node.js pyrkii lähettämään kaikki pyynnöt kerralla.

Ratkaisu ongelmaan voisi olla esimerkiksi pysäyttää Node.js-prosessi kokonaan jokaisen laskun lähettämisen jälkeen hetkellisesti. Tässä tutkimuksessa kehitettyyn sovellukseen käytettiin kuitenkin menetelmää, jossa jokaisesta lähetetystä pyynnöstä (laskusta) kirjataan aikaleima taulukkomuuttujaan ja seuraava pyyntö lähetetään vain, jos taulukosta viimeisen sekunnin aikana lähetettyjen pyyntöjen määrä ei ylitä ohjelmointirajapinnan rajoitetta.

6 Tulokset

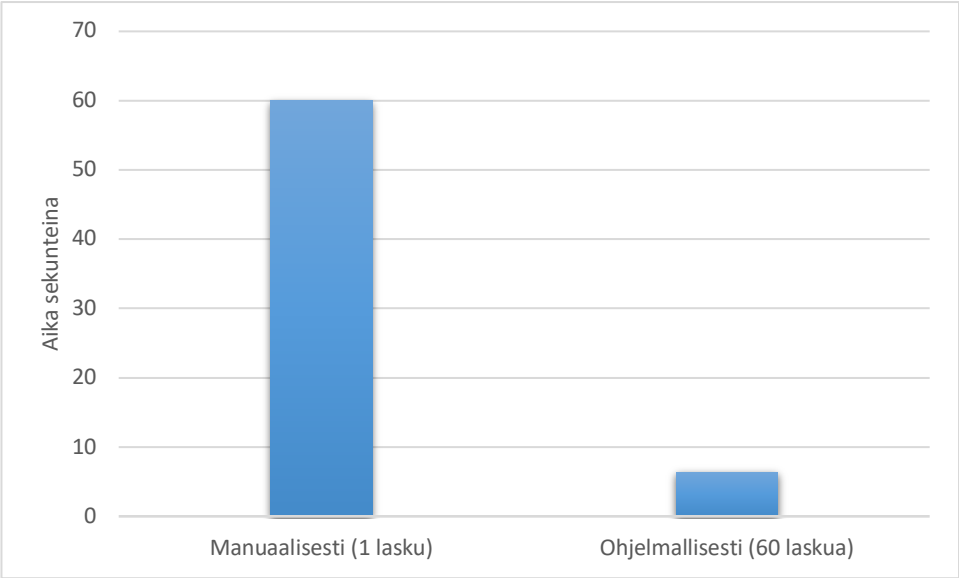
Procountorin REST-ohjelmointirajapinnan puutteellisuuden vuoksi, toimeksiantajayrityksen myyntilaskutus-prosessin täydellinen automatisointi tällä menetelmällä ei ole mahdollista. Prosessissa varmistetaan jokaisen asiakkaan kohdalla, ennen laskun luontia, että asiakkuus löytyy Procountorin asiakasrekisteristä ja koska tätä tarkistusta ei voi rajapinnasta löytyvillä toiminnoilla toistaiseksi tehdä, pysyy tuo osa prosessista manuaalisena.

Suunnittelutieteellisen tutkimuksen tuottamaa sovellusta voidaan arvioida mm. seuraavien ominaisuuksien perusteella: toimivuus, suorituskyky, luotettavuus, käytettävyys sekä soveltuvuudesta kyseiselle yritykselle. (Hevner ym. 2004).

Peilaten tutkimuskysymykseen “Kuinka paljon järjestelmäintegraatio tehostaa yrityksen laskutusprosessia?”, voidaan sanoa tutkimuksen onnistuneen hyvin: Ilman väliohjelmistoa, laskutusprosessin vaiheeseen kaksi, jossa laskun tiedot kopioidaan manuaalisesti Google Sheets -näkymästä Procountorin vastaaviin kenttiin, menee aikaa noin yksi minuutti jokaista laskuriviä kohden (riippuen tapauksesta, laskun luontiin menee 0,5–2 minuuttia).

Uudella toimintamallilla, ohjelmallisesti, eli käyttäen väliohjelmistoa, tuohon kyseiseen vaiheeseen menee 0,5–1,5 sekuntia lähetettäessä yksi laskurivi kerrallaan ja noin 6,5 sekuntia lähetettäessä 60 kappaletta laskuja kerralla (Kuvio 8). Suorituskykyä mitattiin Node.js:n omalla console.time -metodilla laskujen lähettämisen aloituksesta siihen, kunnes vastaus pyyntöön saatiin Procountorin rajapinnalta.

Toimeksiantajayrityksessä lähetetään laskuja keskimäärin 120 kappaletta kuukaudessa, jolloin sovelluksen suorittamana siihen menisi aikaa noin 13 sekuntia ja koko vuoden laskujen lähettämisen vain noin 2,6 minuuttia. Vuositasolla aikaa säästyy siis lähes 24 tuntia eli noin 3 työpäivää. Vaikka automatisaatio koskeekin vain yhtä vaihetta isosta prosessista, tuo järjestelmäintegraatio näiden tulosten valossa merkittävän parannuksen prosessiin kokonaisuutena.



Kuvio 8. Manuaalinen työ verrattuna automaatioon

Sovelluksesta saatiin tehtyä vaatimusmäärittelyiden mukainen ja siitä löytyy kaikki vaaditut ominaisuudet, joten se soveltuu hyvin toimeksiantajayrityksen tarpeisiin. Sovellusta ei vielä tutkimuksen aikana ehditty ottaa virallisesti käyttöön yrityksessä.

7 Yhteenveto ja pohdinta

Tutkimuksen tavoitteena oli tehostaa yrityksen laskutusprosessia, kehittämällä sovellus, joka automatisoisi prosessista yhden aikaa vievän, paljon toistoa vaativan vaiheen. Menetelmänä käytettiin suunnittelutiedettä, jossa ”iteratiivisen etsimisprosessin” avulla haettiin ratkaisua ongelmaan ja sen tuloksena syntyi internetselaimessa suoritettava sovellus. Sovellus toimii integraatio-väliohjelmana, välittäen tietoa kahden järjestelmän välillä ja näin sitoen löyhästi kaksi järjestelmää yhteen.

Mahdollisimman pitkälle digitalisoitu ja automatisoitu taloushallinto tehostaa yrityksen talouteen liittyviä prosesseja ja eri järjestelmien integrointi auttaa johtoa saamaan mahdollisimman ajantasaista tietoa yrityksen taloudesta, mikä on erittäin tärkeää strategisen suunnittelun kannalta (Sähköinen taloushallinto n.d). Tutkimuksen aihe oli näin ollen erittäin hyödyllinen ja tuotti tärkeää tietoa yhdenlaisesta ratkaisusta toteuttaa integraatio itse.

Väliohjelmiston avulla toteutetun järjestelmäintegraation ansiosta yritys ei ole enää välttämättä riippuvainen Google Sheets -palvelusta, vaan tieto voidaan välittää väliohjelmistolle mistä tahansa järjestelmästä. Lähdejärjestelmän muuttaminen vaatisi ainoastaan pieniä muutoksia väliohjelmiston koodiin.

Koska kehityksessä käytetyt teknologiavalinnat tehtiin toimeksiantajan toimesta, suunnittelutieteelliseen tutkimusmenetelmään sisältyvä ”etsimisprosessi” jäi käytännössä pois. Tämän voidaan katso vaikuttavan tutkimuksen luotettavuuteen, koska vertailua eri menetelmien tai teknologioiden, kuten esimerkiksi web-sovelluskehysten välillä, ei tehty.

Sovellus onnistuttiin toteuttamaan vaatimusmäärittelyiden mukaisesti ja sitä käyttämällä prosessia pystyisi testien valossa tehostamaan lähes kolmen työpäivän verran vuositasolla. Kyseessä oleva laskutusprosessi on laaja ja sovellus vaikuttaa vain yhteen osa-alueeseen, mutta tutkimus osoittaa selvästi järjestelmäintegraation tuovan suuren edun prosessin tehokkuuteen jo pienellä automatisoinnilla.

Sovelluksen toteuttaminen ja tutkimustyö oli erittäin opettavaista, ja valmisteli allekirjoittanutta hyvin työharjoitteluun sovelluskehittäjänä. Vaikka tutkimuksen kohteena olevasta prosessista pystyttiinkin tehostamaan vain yhtä osa-aluetta, antoi tutkimus hyvän katsauksen prosessin tilaan ja teknologian tuomista mahdollisuuksista taloushallinnossa.

7.1 Tutkimuksen luotettavuus

Tutkimuksen luotettavuuteen vaikuttavia asioita oli muutamia, kuten esimerkiksi se, että laskutusprosessista vastaa ainoastaan yksi henkilö. Kokemus prosessin tehokkuudesta voi siten olla subjektiivinen. Teknologiavalinnat annettiin toimeksiantajalta, minkä vuoksi tutkimuksesta jäi vaihe pois, jossa olisi tarkkaan tutkittu eri teknologiavaihtoehtojen hyötyjä ja tehty valinta sen perusteella.

Googlen Sheets -ohjelmointirajapinnan rajoituksen vuoksi ei voitu suorittaa testejä, joissa olisi lähetetty täydet 120 kappaletta laskuja yhdellä kertaa, eli se määrä jonka toimeksiantajayritys keskimäärin kuukaudessa lähettää (Googlen rajoitus on 100 pyyntöä 100 sekunnin aikana). Tutkimuksessa tehtiin testit 60 kappaleella josta laskettiin valistunut arvio.

7.2 Kehitysideat ja jatkotutkimustarpeet

Prosessin muitakin vaiheita voi tulevaisuudessa mahdollisesti automatisoida. Procountorin asiakaspalvelun mukaan asiakkuuden tarkistamiseen liittyviä ominaisuuksia on suunnitteilla ohjelmointirajapintaan (Kivinen 2017). Näin ollen myös käsin tehtävän työn asiakkuuden tarkistamiseksi voisi jättää sovelluksen hoidettavaksi.

Integraatiosovelluksen seuraavan version voisi tulevaisuudessa toteuttaa myös kokonaan pilvilaskentapalveluilla, kuten Amazon Web Services -alustalla. Laskurivien tarkistuksen voisi suorittaa esimerkiksi ajastettu Lambda-funktio, jolloin käyttöliittymän voisi poistaa kokonaan.

Lähteet

Express. N.d. Sivuston pääsivulta löytyvä kuvaus. Viitattu 20.1.2018. http://expressjs.com.

Fielding, R. 2000. Doctoral dissertation. University of California, Irvine. Architectural Styles and the Design of Network-based Software Architectures. Viitattu 14.1.2018. http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

Gackenheimer, C. 2013. Node.js Recipes: A Problem-Solution Approach. Apress.

Hevner, A., March, S., Park, J. & Ram, S. 2004. Design Science in Information Systems Research. MIS Quarterly, 28, 75-106. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.103.1725&rep=rep1&type=pdf.

Kivinen, T. 2017. Procountor Oy:n asiakaspalvelun vastaus tiedusteluun ohjelmointirajapinnan ominaisuudesta. Sähköpostiviesti 9.10.2017. Vastaanottaja M. Lakanen. Question about Partner ID.

Lahti, S. & Salminen, T. 2014. Digitaalinen taloushallinto. Helsinki: Sonoma Pro Oy.

Linthicum, S. 2003. Next Generation Application Integration: From Simple Information to Web Services. Addison Wesley.

Lång, M. 1998. Teksti metakognitiivisena artefaktina: sanataiteen ja säveltaiteen ontologiaa. Synteesi, 1998, nro 4, s. 82–94.

Mardan, A. 2014. Express.js Guide - The Comprehensive Book on Express.js. Leanpub.

Moul, B. 2010. Taking Enterprise Application Integration Into The Cloud. Viitattu 21.1.2018. http://www.zdnet.com/article/taking-enterprise-application-integration-into-the-cloud/.

Node.js. N.d. Node.js -sivuston dokumentaatio. Viitattu 18.1.2018. https://nodejs.org/en/. DOCS.

OAuth 2.0. N.d. Kuvaus palvelun kotisivuilla. Viitattu 16.11.2017. https://oauth.net/2/.

Richer, J. N.d. User Authentication with OAuth 2.0. Viitattu 27.1.2018. https://oauth.net/articles/authentication/.

Ruh, W., Maginnis, F., & Brown, W. 2001. Enterprise Application Integration: A Wiley Tech Brief. John Wiley & Sons.

Spasovski, M. 2013. OAuth 2.0 Identity and Access Management Patterns. Packt Publishing.

Sähköinen taloushallinto. N.d. Procountor International Oy:n tuottamiin tietoihin perustuva kirjoitus Suomen yrittäjien kotisivuilla. Viitattu 25.1.2018. https://www.yrittajat.fi/yrittajan-abc/taloushallinto-ja-maksut/taloushallinto/sahkoinen-taloushallinto-317818#

Taloushallinto. N.d. Ammattialan kuvaus Ammattinetin sivustolla. Viitattu 25.1.2018. http://www.ammattinetti.fi/ammattialat/detail/19/6d91ff7ac0315a8d0144dd9038a77bb1.

Tähtinen, S. 2005. Järjestelmäintegraatio : tarve, vaihtoehdot, toteutus. Jyväskylä: Talentum.

Understanding HATEOAS. N.d. Pivotal Software. Viitattu 17.1.2018. https://spring.io/understanding/HATEOAS.