Tanja Pyykönen

Laskutuslisäosa projektinhallintaohjelmiston ja tilitoimistopalvelun välille

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

30.4.2024

Tiivistelmä

Tekijä:	Tanja Pyykönen
Otsikko:	Laskutuslisäosa projektinhallintaohjelmiston ja tilitoimisto-palvelun välille
Sivumäärä:	60 sivua + 1 liite
Aika:	30.4.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Simo Silander Toimitusjohtaja Pete Tahvanainen

---

Insinöörityön tarkoituksena oli tehostaa Systencess Oy:n laskutusprosessia kehittämällä laskutuslisäosa yrityksen Redmine-projektinhallintaohjelmiston ja Talenom-tilitoimistopalvelun välille. Tavoitteena oli luoda laskutuslisäosa ja sen automatisointi, joka generoisi myyntilaskut Redmine-projektinhallintaohjelmiston asiakasprojektien tiedoista ja lähettäisi ne Talenom-tilitoimistopalvelulle kerran kuukaudessa.

Systencess Oy:n toiveena oli parantaa laskutusprosessin tehokkuutta ja tarkkuutta korvaamalla myyntilaskujen manuaalinen luonti laskutuslisäosalla. Laskutuslisäosan integroimisella yrityksen laskutusprosessiin pyrittiin minimoimaan inhimillisten virheiden riskejä, jotka saattaisivat syntyä myyntilaskujen manuaalisen luonnin aikana.

Insinöörityössä keskityttiin laskutuslisäosan ja sen automatisoinnin kehitysprosessiin, joka sisälsi suunnittelun, käytetyt teknologiat, toteutuksen, testauksen ja käyttöönoton. Laskutuslisäosan toiminnallisuudet sisältävät myyntilaskujen generoinnin Redmine-projektinhallintaohjelmiston asiakasprojekteista, myyntilaskujen validoinnin XML-skeeman avulla ja myyntilaskujen lähetyksen Talenom-tilitoimistopalvelulle.

Laskutuslisäosan ajastettu suoritus toteutettiin Microsoft Azure -pilvipalvelualustan Azure-funktion ja Microsoft Power Automate -työnkulun avulla. Laskutettavia asiakasprojekteja käsittelevä Azure-funktio käynnistettiin jokaisen kuukauden viimeisenä päivänä Power Automate -työnkulun avulla.

Laskutuslisäosa ja sen automatisointi kehitettiin ja integroitiin onnistuneesti osaksi yrityksen laskutusprosessia. Insinöörityön tuloksena saatiin toimiva ja automatisoitu laskutuslisäosa, joka laskuttaa yrityksen asiakkaat ajastetusti kerran kuukaudessa.

Avainsanat: laskutuslisäosa, myyntilasku, laskutusprosessi, automatisointi, Redmine, Talenom, Java, Microsoft Azure, Microsoft Power Automate

---

Abstract

Author:	Tanja Pyykönen
Title:	Invoicing Add-on between Project Management Software and Accounting Service
Number of Pages:	60 pages + 1 appendix
Date:	30 April 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Simo Silander, Senior Lecturer Pete Tahvanainen, CEO

The purpose of the study was to enhance Systencess Oy's invoicing process by developing an invoicing add-on between the company's Redmine project management software and Talenom accounting service. The goal was to create an invoicing add-on and its automation, which would generate sales invoices from the customer project information in the Redmine project management software and to send them to the Talenom accounting service once a month.

Systencess Oy's wish was to improve the efficiency and accuracy of the invoicing process by replacing the manual creation of sales invoices with the invoicing add-on. The integration of the invoicing add-on into the company's invoicing process aimed to minimize the risks of human error that could occur during the manual creation of sales invoices.

The thesis is focused on the development process of the invoice add-on and its automation, which includes its planning, technologies used, implementation, testing, and deployment. The functionalities of the invoicing add-on included generating sales invoices from customer projects in the Redmine project management software, validating sales invoices with XML schema, and sending sales invoices to Talenom accounting service.

The scheduled execution of the invoicing add-on was implemented using Microsoft Azure cloud platform's Azure function and the Microsoft Power Automate workflow. The Azure function responsible for handling invoiceable customer projects was triggered on the last day of each month using the Power Automate workflow.

The invoicing add-on and its automation was successfully developed and integrated into the company's invoicing process. The result of the project was a functional and automated invoicing add-on, which invoices the company's customers once a month on a scheduled basis.

Keywords: invoicing add-on, sales invoice, invoicing process, automation, Redmine, Talenom, Java, Microsoft Azure, Microsoft Power Automate

Sisällys

Lyhenteet

1 Johdanto 1

2 Nykytilanne ja tavoitteet 2

2.1 Nykyinen laskutusprosessi 2
2.2 Suunniteltu laskutusprosessi ja tavoitteet 4

3 Määrittely ja suunnittelu 6

3.1 Käyttötapausmalli 6
3.2 Käyttötapaus 1: Hae tiedot Redmine-projektinhallintaohjelmistosta 8
3.3 Käyttötapaus 2: Generoi myyntilaskut 9
3.4 Käyttötapaus 3: Validoi myyntilaskut 10
3.5 Käyttötapaus 4: Lähetä myyntilaskut Talenom-tilitoimistopalvelulle 10
3.6 Yrityksen ohjeistukset ja toiveet 11

4 Käytetyt teknologiat ja työkalut 11

4.1 Java 12
4.2 Apache Maven 14
4.3 Redmine 15
4.4 Talenom 18
4.5 Microsoft Azure 20
4.6 Microsoft Power Automate 22
4.7 Kehitysprosessin hallinta 24

5 Toteutus 27

5.1 Kehitysympäristön asennus ja konfigurointi 27
5.2 Myyntilaskujen tietojen haku Redmine-projektinhallintaohjelmistosta 29
5.3 Myyntilaskujen generointi 34
5.4 Myyntilaskujen validointi 36
5.5 Myyntilaskujen lähetys Talenom-tilitoimistopalvelulle 37
5.6 Azure-funktion luonti 39
5.7 Power Automate -työnkulun luonti 42

6 Testaus 44

6.1 Toiminnallisuuksien testaus 44
6.1.1 Yksikkötestaus 45
6.1.2 Manuaalinen testaus 46
6.2 Automatisoinnin testaus 49

7 Käyttöönotto 52

8 Yhteenveto 54

Lähteet 56

Liitteet

Liite 1: Myyntilaskujen elementit

Lyhenteet

API: *Application Programming Interface*. Ohjelmointirajapinta eli rajapinta, joka mahdollistaa erilaisten järjestelmien vuorovaikutuksen keskenään.

CRUD: *Create, Read, Update, Delete.* Tapahtumamalli, jota käytetään rajapinnoissa ja tietokannoissa esimerkiksi tietojen hakemiseen tai tallentamiseen.

DOM: *Document Object Model.* XML-dokumenttien esitysmalli, joka mahdollistaa XML-dokumenttien rakenteen ja sisällön käsittelyn.

HTTP: *Hypertext Transfer Protocol*. Protokolla, joka mahdollistaa tiedon siirtämisen esimerkiksi web-selainten ja -palvelinten välillä.

JAXP: *Java API for XML Processing.* Java.xml-pakettiin kuuluva rajapinta XML-tietojen käsittelyyn.

JSON: *Javascript Object Notation*. Formaatti, jota käytetään tietojen siirtämiseen ja tallentamiseen.

POM: *Project Object Model*. Apache Maven -projektinhallintakehyksen hyödyntämä XML-muotoinen tiedosto, joka hoitaa projektinhallinnan keskitetysti.

REST: *Representation State Transfer*. Ohjelmointirajapintojen arkkitehtuurimalli, joka hyödyntää HTTP-protokollaa.

SDK: *Software Development Kit*. Ohjelmistokehityspaketti, joka sisältää työkaluja kehittäjille, kuten kirjastoja ja dokumentaatioita.

SOAP: *Simple Object Access Protocol.* XML-pohjainen protokolla tietojen vaihtoon, jota yleensä käytetään HTTP:n kanssa hajautetuissa ympäristöissä.

UML: *Unified Modeling Language*. Standardoitu mallinnuskieli ohjelmistojärjestelmien visuaaliseen suunnitteluun, dokumentointiin ja kommunikaatioon.

URI: *Uniform Resource Identifier*. Yksilöivä tunniste, joka määrittää internetissä olevan resurssin sijainnin tai nimen.

UTF: *Unicode Transformation Format.* Merkkijonojen koodaukseen käytettävä merkistökoodaus.

W3C: *World Wide Web Consortium.* Organisaatio, joka tarjoaa verkkostandardeja ja ohjeita.

WXS: *W3C XML Schema.* W3C-organisaation ylläpitämä XML-skeemakieli, joka tarjoaa oliopohjaisen tavan määritellä XML-skeemoja.

XML: *Extensible Markup Language*. Merkintäkieli, joka soveltuu rakenteellisten tietojen tallentamiseen ja siirtämiseen järjestelmien välillä.

1 Johdanto

Tässä insinöörityössä käsitellään laskutusprosessin automatisointia ja sen toteuttamista laskutuslisäosan avulla Systencess Oy:n sisäisenä kehitysprojektina. Insinöörityön tilasi Systencess Oy, joka toivoi ratkaisua laskutusprosessin tehostamiseen ja automatisointiin. Insinöörityön tavoitteena oli kehittää laskutuslisäosa yrityksen Redmine-projektinhallintaohjelmiston ja Talenom-tilitoimistopalvelun välille. Lisäksi tavoitteena oli kehittää laskutuslisäosan automatisointi, joka suorittaisi laskutuslisäosan toiminnallisuudet automaattisesti ja ajastetusti kerran kuukaudessa.

Laskutuslisäosan keskeisenä tavoitteena oli automatisoida myyntilaskujen luonti Redmine-projektinhallintaohjelmistossa olevien asiakasprojektien tietojen perusteella ja korvata yrityksen nykyinen manuaalinen prosessi, jossa myyntilaskujen tiedot kopioidaan Redmine-projektinhallintaohjelmistosta myyntilaskuille. Laskutuslisäosan integroiminen osaksi yrityksen laskutusprosessia voi parhaimmillaan vähentää myyntilaskujen tietojen manuaalisen kopioinnin aiheuttamia virheitä ja nopeuttaa laskutusprosessia.

Insinöörityö rakentuu laskutuslisäosan ja sen automatisoinnin suunnittelusta, toteutuksesta, testauksesta ja käyttöönotosta. Aluksi käsitellään yrityksen nykyistä laskutusprosessia ja suunniteltua laskutusprosessia, jossa toteutettava laskutuslisäosa on integroitu osaksi yrityksen laskutusprosessia. Tämän jälkeen käydään läpi, miten laskutuslisäosa määriteltiin ja suunniteltiin kehitystyötä varten. Lisäksi käsitellään kehitystyössä käytetyt teknologiat ja työkalut sekä laskutuslisäosan ja sen automatisoinnin kehitysprosessin hallinta. Tämän jälkeen siirrytään laskutuslisäosan ja sen automatisoinnin toteutukseen, jossa kehittäminen kuvataan suunniteltujen vaatimusten mukaisesti. Toteutuksen jälkeen keskitytään laskutuslisäosan ja sen automatisoinnin testaukseen erilaisissa olosuhteissa. Lopuksi käsitellään laskutuslisäosan käyttöönottoa.

Insinöörityössä ei syvennytä laskutuslisäosan ohjelmakoodien yksityiskohtiin yrityksen toiveiden ja kilpailukyvyn säilyttämisen vuoksi. Sen sijaan

laskutuslisäosan ohjelmakoodia käsitellään erilaisten esimerkkien, yksinkertaistettujen versioiden ja havainnollistavien kaavioiden avulla.

2 Nykytilanne ja tavoitteet

Tässä luvussa käsitellään Systencess Oy:n nykyistä laskutusprosessia ja toteutettavan laskutuslisäosan tavoitteita. Aluksi esitellään yrityksen nykyinen laskutusprosessi yksityiskohtaisesti, mukaan lukien siihen liittyvät työkalut ja manuaaliset vaiheet. Samalla tarkastellaan nykyisen laskutusprosessin haasteita ja virhealttiutta. Lopuksi tarkastellaan suunniteltua laskutusprosessia laskutuslisäosan avulla, sekä insinöörityöhön liittyviä tavoitteita ja haasteita.

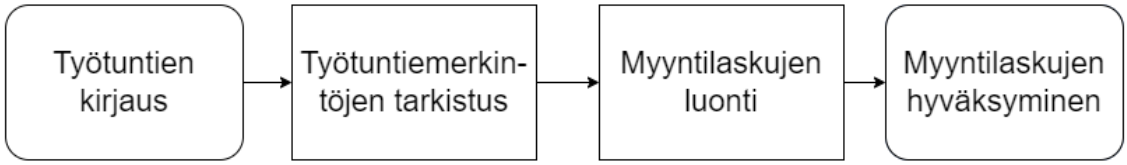
2.1 Nykyinen laskutusprosessi

Systencess Oy on osakeyhtiö, joka toimii markkinoinnin automatisoinnin, ohjelmistokehityksen ja analytiikan parissa [1; 2]. Yrityksen asiakkaiden laskutus tehdään asiakasprojektikohtaisesti kerran kuukaudessa. Yrityksessä käytetään Redmine-projektinhallintaohjelmistoa asiakasprojektien hallintaan ja Talenom-tilitoimistopalvelua asiakkaiden laskutukseen ja kirjanpitoon.

Kuvassa 1 havainnollistetaan yrityksen nykyistä laskutusprosessia laskutuksesta ja taloushallinosta vastaavan henkilön, eli laskuttajan näkökulmasta. Yrityksessä työskentelevä henkilö aloittaa laskutusprosessin kirjaamalla asiakasprojekteihin käytetyt työtunnit Redmine-projektinhallintaohjelmistoon. Työtunnit pyritään kirjaamaan automaattisesti käyttämällä Toggl Track -ajanseurantaohjelmistoa. Redmine Toggl -laajennuksen avulla ajanseurantaohjelmiston työtuntimerkinnät synkronoituvat automaattisesti Redmine-projektinhallintaohjelmistoon [3]. Työtuntimerkinnät voidaan myös vaihtoehtoisesti kirjata Redmine-projektinhallintaohjelmistoon manuaalisesti [4].

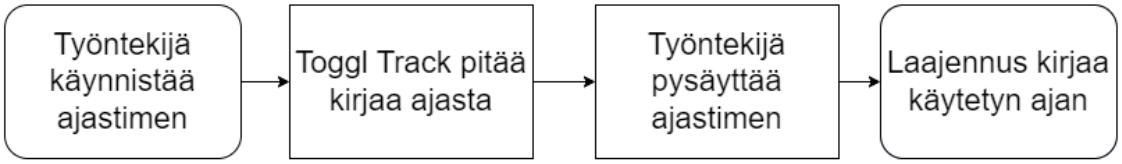
Laskuttaja tarkistaa työtuntimerkinnät ennen myyntilaskujen manuaalista luontia, jotta voidaan olla varmoja siitä, että Toggl Track -ajanseurantaohjelmisto on synkronoinut ne oikein Redmine-projektinhallintaohjelmistoon. Laskuttaja luo

myyntilaskut kopioimalla laskutukseen tarvittavat tiedot Redmine-projektinhallintaohjelmiston asiakasprojekteista. Lopuksi laskuttaja tarkistaa ja hyväksyy myyntilaskut.



Kuva 1. Systencess Oy:n asiakkaiden laskutusprosessi laskuttajan näkökulmasta.

Kuvassa 2 havainnollistetaan, kuinka Toggl Track -ajanseurantaohjelmisto ja Redmine Toggl -laajennus automatisoivat työtuntien kirjausta. Työntekijä käynnistää Toggl Track -ajanseurantaohjelmiston ajastimen aloittaessaan tai jatkaessaan asiakasprojektin työstämistä. Toggl Track -ajanseurantaohjelmisto pitää automaattisesti kirjaa työhön käytetystä ajasta. Työskentelyn jälkeen työntekijä pysäyttää ajastimen, minkä jälkeen Redmine Toggl -laajennus kirjaa työtuntimerkinnät Redmine-projektinhallintaohjelmistoon [3; 5].



Kuva 2. Työtuntien kirjausten automatisointi Toggl Track -ajanseurantaohjelmiston ja Redmine Toggl -laajennuksen avulla [3; 5].

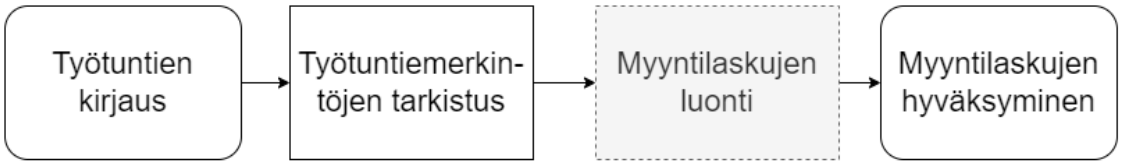
Nykyisessä laskutusprosessissa tietojen kopioiminen ja syöttäminen manuaalisesti myyntilaskuille ovat alttiita operaatioita inhimillisille virheille. Lisäksi ne vievät laskuttajalta aikaa ja resursseja, mikä voi aiheuttaa tehokkuuden heikentymistä ja hidastaa prosesseja. [6, s. 215.]

Työtuntien kirjausten automatisoinnilla yrityksessä on pyritty vähentämään mahdollisia virheitä, jotka voivat tapahtua työtuntien kirjauksessa. Myyntilaskuja

luodessa työtuntimerkinnät kuitenkin syötetään myyntilaskuille manuaalisesti. Vaikka työtuntien kirjauksessa tai työtuntimerkintöjen tarkistuksessa ei ilmenisi virheellisiä tietoja, voi myyntilaskujen manuaalisen luonnin aikana tapahtua virheitä. Tämä voi johtaa siihen, että alkuperäisesti virheettömät arvot siirtyvät virheellisinä myyntilaskuille.

2.2 Suunniteltu laskutusprosessi ja tavoitteet

Systencess Oy:n toiveena oli tehostaa ja automatisoida nykyistä laskutusprosessia vähentämällä myyntilaskujen luomiseen vaadittavaa aikaa ja minimoida virheellisten tietojen syöttö myyntilaskuille. Kuvassa 3 esitetään yrityksen laskutusprosessi laskuttajan näkökulmasta, kun tässä insinöörityössä toteutettava laskutuslisäosa on integroitu osaksi yrityksen laskutusprosessia. Työtuntien kirjaus, työtuntimerkintöjen tarkistus ja myyntilaskujen hyväksyminen suoritetaan samalla tavalla kuin nykyisessä laskutusprosessissa, kuvassa 1. Myyntilaskujen manuaalinen luonti tullaan korvaamaan laskutuslisäosan avulla.



Kuva 3. Systencess Oy:n asiakkaiden laskutusprosessi laskuttajan näkökulmasta, kun laskutuslisäosa on integroitu osaksi yrityksen laskutusprosessia.

Tarkka ja virheetön laskutus on tärkeä toiminto yrityksille. Virheelliset myyntilaskut voivat aiheuttaa maksujen viivästymistä ja heikentää yrityksen ammattimaisuutta ja mainetta. On siis tärkeää, että virheellisten myyntilaskujen määrä pyritään minimoimaan. Laskutuslisäosan ja sen automatisoinnin avulla voidaan parhaassa tapauksessa tehostaa yrityksen laskutusprosessia sekä vähentää virheellisten tietojen syöttämistä myyntilaskuille [6, s. 215; 7].

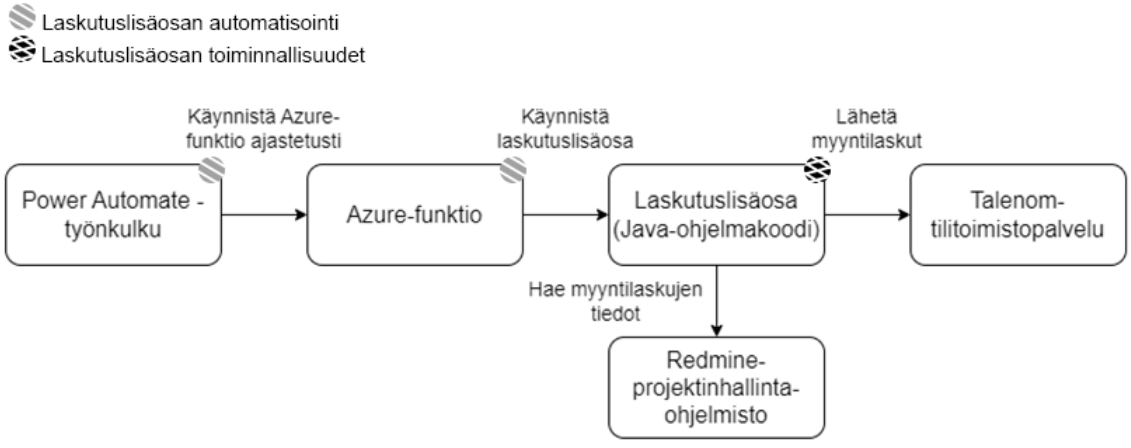
Insinöörityössä ei keskitytä erilaisten mittareiden, kuten resurssien säästöjen mittaamiseen tai niiden analyysiin laskutuslisäosan käyttöönoton jälkeen. Sen

sijaan insinöörityössä keskitytään laskutuslisäosan ja sen automatisoinnin suunnitteluun, kehitykseen, testaukseen ja käyttöönottoon.

Kuten luvussa 1 Johdanto mainittiin, insinöörityön tavoitteena on toteuttaa laskutuslisäosa ja sen automatisoitu ja ajastettu suoritus. Laskutuslisäosan ja sen automatisoinnin tavoitteena on generoida myyntilaskut Systencess Oy:n Redmine-projektinhallintaohjelmiston asiakasprojekteista ja lähettää myyntilaskut Talenom-tilitoimistopalvelulle laskutusta varten kerran kuukaudessa automaattisesti ja ajastetusti.

Kuva 4 havainnollistaa, mitä tässä insinöörityössä toteutettiin ja mistä osista laskutuslisäosa ja sen automatisointi koostuu. Laskutuslisäosan automatisointi toteutettiin Power Automate -työnkulun ja Azure-funktion avulla. Power Automate -työnkulku käynnistää Azure-funktion ajastetusti kerran kuukaudessa ja Azure-funktio käynnistää laskutuslisäosan ja sen toiminallisuudet.

Itse laskutuslisäosa koostuu Java-ohjelmakoodista, joka sisältää laskutuslisäosan toiminnallisuudet. Toiminnallisuudet sisältävät myyntilaskujen generoinnin Redmine-projektinhallintaohjelmiston asiakasprojekteista haetuilla tiedoilla, myyntilaskujen validoinnin XML-skeemalla ja niiden lähetyksen Talenom-tilitoimistopalvelulle.



Kuva 4. Laskutuslisäosan toiminnallisuuksien ja laskutuslisäosan automatisoinnin toimintalogiikka.

Laskutuslisäosan kehittäminen ja sen automatisointi olivat teknisesti haastavia monesta syystä. Ensinnäkin laskutuslisäosa ja sen automatisointi sisälsivät useita ei järjestelmiä, mikä toi mukanaan kokonaisuuden hallinnan ja integroinnin haasteita. Järjestelmien yhteensopivuuden varmistaminen ja niiden saumaton toiminta olivatkin keskeinen vaikeus.

Laskutuslisäosa vaati tarkkaa suunnittelua ja hallintaa, jotta kaikki osa-alueet saatiin toimimaan yhteen. Pienikin virhe tai epäyhteensopivuus saattoi vaikuttaa koko järjestelmään, joka korosti tarvetta huolelliseen suunnitteluun ja virheiden hallintaan.

Myyntilaskujen tietojen hakuun ja generointiin kehitetyt algoritmit tuottivat myös haasteita, sillä ne vaativat syvällistä ymmärrystä laskutusprosessista sekä algoritmien suunnittelusta ja toteutuksesta. Algoritmien tuli olla tehokkaita ja tarkkoja, jotta laskutuslisäosa voisi automatisoida myyntilaskujen luonnin luotettavasti ja virheettömästi. Lisäksi laskutuslisäosan automatisointi loi haasteita ennestään tuntemattomien ympäristöjen vuoksi.

3 Määrittely ja suunnittelu

Tässä luvussa käsitellään laskutuslisäosan määrittelyä ja suunnittelua. Aluksi tarkastellaan laskutuslisäoan keskeisiä toimintoja käyttötapausmallin ja neljän käyttötapauksen avulla. Lopuksi tarkastellaan Systencess Oy:n ohjeistuksia ja toiveita laskutuslisäosan kehityksen suhteen.

3.1 Käyttötapausmalli

Systencess Oy määritteli korkean tason vaatimuksen laskutuslisäosalle ja sen automatisoinnille, mikä havainnollisti, miten laskutuslisäosan tulisi toimia. Korkean tason vaatimuksena oli, että laskutuslisäosa generoisi ja lähettäisi myyntilaskut automaattisesti ja ajastetusti Talenom-tilitoimistopalveluun kerran kuukaudessa.

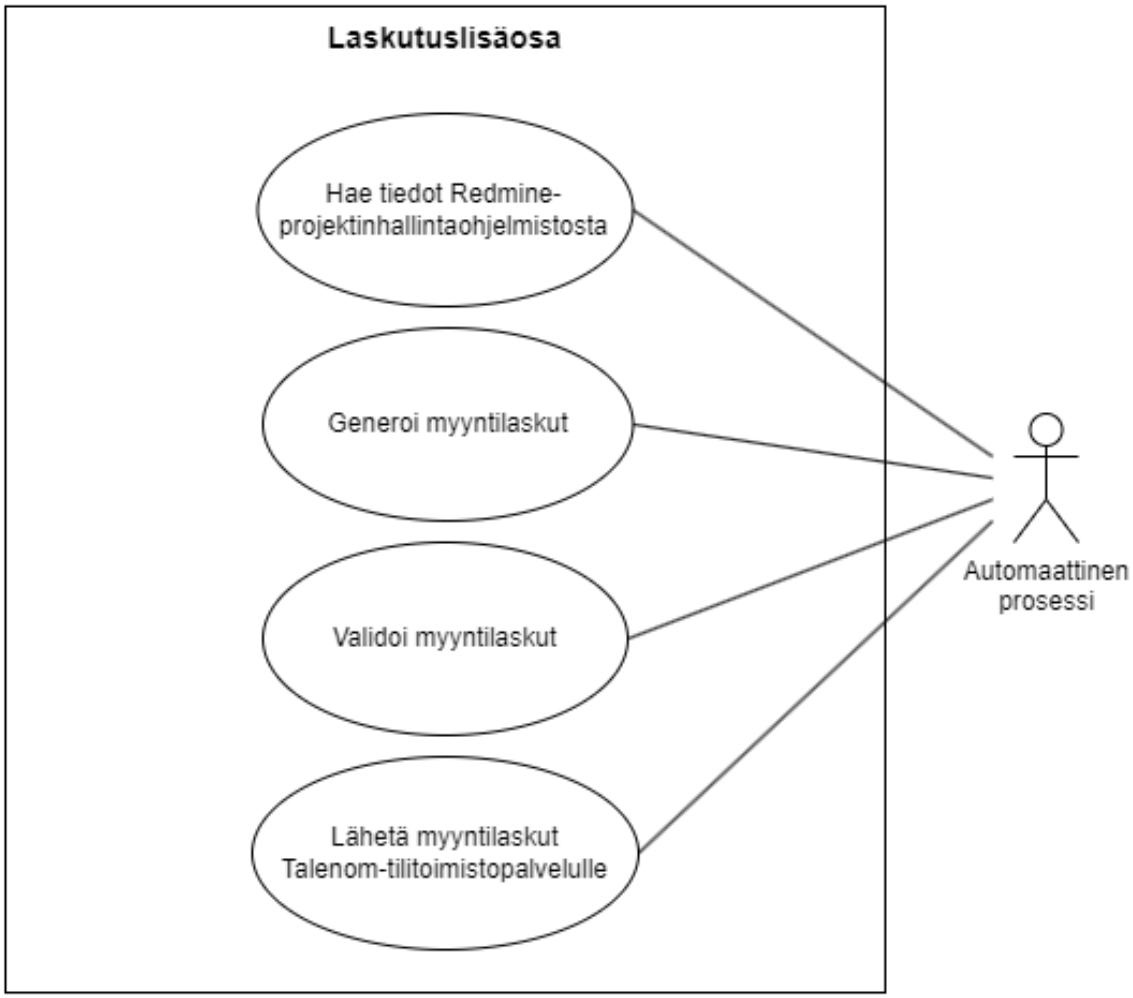
Jotta laskutuslisäosan vaadittavat toiminnallisuudet ja tarkempi kokonaiskuva saatiin selvitettyä, suunnittelu alkoi määrittelemällä laskutuslisäosan keskeiset päätoiminnot. Nämä päätoiminnot kuvattiin käyttötapauksina ja visualisoitiin UML-käyttötapauskaaviota (Unified Modeling Language) hyödyntäen [8; 9]. Standardoitua UML-mallinnuskieltä käytetään ohjelmistojärjestelmien visuaaliseen suunnitteluun, dokumentointiin ja kommunikaatioon [9].

Käyttötapauskaavio on yksi UML-mallinnuskielen käyttäytymiskaavioista, joka keskittyy kuvaamaan järjestelmien toiminnallisuutta, mutta ei tarjoa yksityiskohtia siitä, miten nämä toiminnot toteutetaan. Hyödyntämällä käyttötapauskaaviota laskutuslisäosan kehityksen varhaisessa vaiheessa saatiin parempi yleiskuva siitä, mitä toiminnallisuuksia laskutuslisäosan tulisi sisältää. [8.]

Kuvassa 5 on esitetty käyttötapauskaavio laskutuslisäosan toiminnallisuuksista. Se sisältää neljä käyttötapausta, jotka määrittelevät laskutuslisäosan tärkeimmät päätoiminnallisuudet ja vaatimukset. Käyttötapaukset ovat:

- tiedon haku Redmine-projektinhallintaohjelmistosta
- myyntilaskujen generointi
- myyntilaskujen validointi
- myyntilaskujen lähettäminen Talenom-tilitoimistopalvelulle.

Käyttötapauskaavio sisältää toimijan nimeltä automaattinen prosessi, joka kuvaa laskutuslisäosan sisäistä komponenttia. Tämä automaattinen prosessi on vastuussa laskutukseen liittyvien tehtävien suorittamisesta ilman käyttäjän manuaalista väliintuloa.



Kuva 5. Käyttötapauskaavio laskutuslisäosan toiminnallisista piirteistä.

Käyttötapauskaavion pohjalta luotiin käyttötapaukset, jotka kuvaavat tarkemmin laskutuslisäosan toiminnallisuuksia. Seuraavaksi tarkastellaankin laskutuslisäosan jokaista käyttötapausta yksityiskohtaisemmin.

3.2 Käyttötapaus 1: Hae tiedot Redmine-projektinhallintaohjelmistosta

Automaattinen prosessi aloittaa laskutuksen hakemalla tarvittavat tiedot myyntilaskujen generointia varten asiakasprojekteista Redmine-projektinhallintaohjelmiston API:n (Application Programming Interface) eli rajapinnan avulla [11]. Automaattisen prosessin käynnistyminen vaatii, että Redmine-projektinhallintaohjelmiston rajapinnan tunnistetiedot, kuten URI-osoite (Uniform Resource

Identifier) ja API-avain ovat oikein [12]. URI-osoitetta ja API-avainta tarvitaan, jotta API-kutsuja voidaan käyttää myyntilaskujen tietojen hakemiseen [11; 13].

Automaattinen prosessi tarkastaa jokaisen asiakasprojektin kohdalta, onko automattisen laskutuksen tila päällä. Seuraavaksi automaattinen prosessi tarkistaa, onko asiakasprojekteihin kirjattu työtuntimerkintöjä laskutettavan kuukauden aikana. Molemmat tarkistukset on tärkeä tehdä, jotta laskutuslisäosa ottaa huomioon vain ne asiakasprojektit, jotka halutaan laskuttaa automaattisesti ja joita on työstetty laskutettavan kuukauden aikana.

Mikäli asiakasprojekti on merkitty automaattisesti laskutettavaksi ja siihen on kirjattu työtuntimerkintöjä laskutettavan kuukauden aikana, automaattinen prosessi kerää asiakasprojektista tarvittavat tiedot laskutusta varten. Tämän jälkeen siirrytään automaattisesti seuraavaan vaiheeseen laskutusprosessissa, jossa myyntilaskut generoidaan haettujen tietojen perusteella.

3.3 Käyttötapaus 2: Generoi myyntilaskut

Automaattinen prosessi ryhtyy generoimaan myyntilaskuja, kun laskutettavien asiakasprojektien tiedot on haettu Redmine-projektinhallintaohjelmiston rajapinnan avulla onnistuneesti. Automaattinen prosessi käsittelee yhden asiakasprojektin kerrallaan luoden myyntilaskun perustuen asiakasprojektin tietoihin, asiakastietoihin ja kirjattuihin työtuntimerkintöihin.

Tämän jälkeen automaattinen prosessi muuntaa myyntilaskun XML-muotoiseksi (Extensible Markup Language) tiedostoksi ja tallentaa sen myöhempää käyttöä varten [14]. Myyntilaskut tulee esittää XML-muodossa, jotta ne voidaan myöhemmässä vaiheessa validoida ja lähettää Talenom-tilitoimistopalvelulle. Kun XML-muotoiset myyntilaskutiedostot on generoitu ja tallennettu, siirrytään kolmanteen vaiheeseen laskutusprosessissa, jossa XML-muotoiset myyntilaskutiedostot validoidaan XML-skeeman avulla. XML-skeema kuvaa rajoituksia, jotka koskevat tietyn XML-tiedoston rakennetta [15].

3.4 Käyttötapaus 3: Validoi myyntilaskut

Myyntilaskujen validointi XML-skeeman avulla on olennainen osa laskutusprosessia, sillä se auttaa minimoimaan virheellisiä myyntilaskuja, jotka voivat aiheuttaa ongelmia niiden käsittelyssä tai laskutuksessa [15]. Automaattinen prosessi aloittaa myyntilaskujen validoinnin lataamalla XML-skeeman ja XML-muotoiset myyntilaskutiedostot, mikäli ne ovat saatavilla. Tämän jälkeen se suorittaa XML-muotoisten myyntilaskutiedostojen validoinnin XML-skeeman avulla, mikä varmistaa niiden virheettömyyden.

Onnistuneen validoinnin jälkeen kaikkien myyntilaskujen odotetaan olevan määritellyn XML-skeeman mukaisia. Tämän jälkeen siirrytään neljänteen ja viimeiseen vaiheeseen, jossa automaattinen prosessi lähettää validoidut myyntilaskut Talenom-tilitoimistopalvelulle.

3.5 Käyttötapaus 4: Lähetä myyntilaskut Talenom-tilitoimistopalvelulle

Ennen kuin myyntilaskut voidaan lähettää Talenom-tilitoimistopalvelulle, automaattinen prosessi varmistaa, että ne ovat läpäisseet validointiprosessin onnistuneesti. Kaikki onnistuneesti validoidut myyntilaskut lähetetään Talenom-tilitoimistopalvelulle myyntilaskurajapinnan avulla laskutusta ja kirjanpitoa varten. Jotta myyntilaskut voidaan lähettää myyntilaskurajapinnan avulla, sen tunnistetiedot on oltava oikein.

Kun kaikki neljä käyttötapausta suoriutuvat onnistuneesti, myyntilaskut etenevät manuaaliseen tarkistusvaiheeseen. Tässä vaiheessa laskuttaja tarkistaa myyntilaskut ja hyväksyy ne, mikäli ne ovat oikein, tai tekee tarvittavat muokkaukset virheitä sisältäviin myyntilaskuihin. Hyväksynnän jälkeen Talenom-tilitoimistopalvelu vastaa myyntilaskujen automaattisesta lähettämisestä asiakkaille ja arkistoinnista kirjanpitoa varten.

3.6 Yrityksen ohjeistukset ja toiveet

Systencess Oy antoi yksityiskohtaisempia ohjeita ja toiveita laskutuslisäosalle ja sen automatisoinnille, jotka tuli ottaa huomioon teknologioiden ja työkalujen valinnassa, sekä laskutuslisäosan toiminnallisuuksien ja sen automatisoinnin suunnittelussa ja toteutuksessa. Teknologioiden ja työkalujen valinnassa noudatettiin pääosin yrityksen käytössä olevien ympäristöjen vaatimuksia. Teknologioiden ja työkalujen valinnasta kerrotaan yksityiskohtaisemmin seuraavassa luvussa.

Suunnitteluvaiheessa oli tärkeää ottaa huomioon laskutuslisäosan joustavuus ja ylläpidettävyys. Vaikka yrityksen nykyinen tarve oli automatisoida laskutus yhdestä Redmine-projektinhallintaohjelmiston ympäristöstä yhteen Talenom-tilitoimistopalvelun ympäristöön, suunnittelussa otettiin huomioon mahdollisuus laajentaa laskutuslisäosan käyttöä useammalle ympäristölle. Ottamalla joustavuus ja ylläpidettävyys huomioon suunnitteluvaiheessa, mahdollistettiin laskutuslisäosan jatkokehitysmahdollisuudet ja skaalautuvuus yrityksen tulevaisuuden tarpeisiin.

Käytännössä joustavuutta ja ylläpidettävyyttä saatiin lisättyä toteutusvaiheessa siten, että esimerkiksi rajapintojen URI-osoitteita ja muita arvoja ei kovakoodattu Java-ohjelmakoodiin [16]. Arvot syötettiin Microsoft Power Automate -pilviautomaatioympäristön työnkulkuun. Näin ollen arvot ovat helposti konfiguroitavissa ja vaihdettavissa tarpeen tullen. Kuten luvussa 2.2 Suunniteltu laskutusprosessi ja tavoitteet mainittiin, Power Automate -työnkulku vastasi laskutuslisäosan Azure-funktion automaattisesti ja ajastetusta käynnistämisestä.

4 Käytetyt teknologiat ja työkalut

Tässä luvussa esitellään laskutuslisäosan ja sen automatisoinnin kehityksessä käytetyt teknologiat ja työkalut. Lisäksi käydään läpi kehitysprosessin hallintaa, jossa käsitellään laskutuslisäosassa ja sen automatisoinnissa käytettyä ohjelmistokehitysmallia ja versionhallintaa. Laskutuslisäosan ja sen automatisoinnin

kehityksessä käytettiin runsaasti erilaisia teknologioita ja työkaluja. Suurin osa käytetyistä teknologioista ja työkaluista olivat valmiiksi Systencess Oy:n käytössä, jonka vuoksi erillistä suurempaa valintaa, miksi kyseistä teknologiaa tai työkalua käytettiin, ei näiden kohdalta tehty.

4.1 Java

Systencess Oy suositteli Java-ohjelmointikielen valitsemista laskutuslisäosan ohjelmointikieleksi. Vaikka Javan valinta ei ollut ehdoton vaatimus, sitä suositeltiin sen vankan ekosysteemin, laajan yhteisön tuen ja yrityskäytön vuoksi. Ennen ohjelmointikielen lopullista valintaa, tutkittiin ja varmistettiin Javan soveltuvuus laskutuslisäosan kehitykseen ja tarpeisiin. Soveltuvuus varmistettiin tutustumalla ohjelmointikielten suosiokyselyihin, Javan tekniseen dokumentaation sekä sen kirjastoihin.

Java on ollut käytössä yli kahden vuosikymmenen ajan, ja tämä pitkäaikainen käyttöhistoria on mahdollistanut Java-yhteisön kehittää laaja valikoima kirjastoja, työkaluja ja kehitysympäristöjä, jotka tukevat Java-sovellusten kehitystä erilaisissa käyttötarkoituksissa. Oracle, Javan pääkehittäjä ja ylläpitäjä, tarjoaa kattavan dokumentaation ja virallisen tuen, mikä vahvistaa Javan asemaa vakaana ja luetettavana ohjelmointikielenä. [17.]

Stack Overflow Developer Survey 2023 -kysely paljastaa suosituimmat ohjelmointikielet ohjelmistokehittäjien keskuudessa. Kyselyn mukaan JavaScript, Python, TypeScript ja Java olivat yleisimpiä käytettyjä ohjelmointikieliä. Vaikka Javan suosio oli hieman alhaisempi kolmeen muuhun suosituimpaan verrattuna, se pysyi vahvana suosikkina ohjelmistokehittäjien joukossa. [18.]

The State of Developer Ecosystem 2023 -tutkimus tarjoaa katsauksen ohjelmistokehityksen tilaan. Tässä tutkimuksessa Java oli myös yksi suosituimmista ohjelmistokielistä, rinnallaan JavaScript, Python ja TypeScript. [19.] Vaikka raportoidut suosiot vaihtelivat kyselyn ja tutkimuksen välillä, molemmat lähteet vahvistivat Javan vankan aseman suosituimpien ohjelmointikielten joukossa.

Systencess Oy:n suosituksen ja edellä mainittujen seikkojen perusteella päätettiin Java valita laskutuslisäosan ohjelmointikieleksi. Laskutuslisäosan logiikka ja toiminnallisuudet toteutettiin Java-ohjelmointikielellä, käyttäen Java 11 -versiota. Java-ohjelmakoodin kirjoittamiseen käytettiin Visual Studio Code -kehitysympäristöä.

Javan ekosysteemi tarjoaa laajan valikoiman ulkopuolisia kirjastoja, sekä monipuolisen standardikirjaston [20; 21]. Laskutuslisäosan kehityksessä hyödynnettiin Javan standardikirjastoa sekä ulkopuolisten kehittäjien luomia kirjastoja, jotka tarjoavat valmiita ratkaisuja erilaisiin ohjelmointitehtäviin [20; 22]. Laskutuslisäosan toteutuksessa käytettiin seuraavia pakkauksia Javan standardikirjastosta:

- Java.util-pakkausta käytettiin laajalti tietorakenteiden käsittelyyn, ajankäsittelyyn sekä valinnaisten arvojen käsittelyyn.
- Java.io-pakkausta hyödynnettiin tiedostojen hallintaan, esimerkiksi tiedostojen lukeminen ja kirjoittaminen.
- Java.text-pakkausta käytettiin pääasiassa tekstin ja numeroiden muotoiluun ja tulkitaan.
- Java.xml-pakkausta hyödynnettiin XML-dokumenttien käsittelyssä.

Ulkoisia kirjastoja ja niiden pakkauksia käytettiin laskutuslisäosassa täydentämään aukkoja, joita Javan standardikirjasto ei kattanut. Laskutuslisäosan toteutuksessa hyödynnettiin seuraavia ulkoisia kirjastoja:

- JSON-java-kirjastoa (Javascript Object Notation), toiselta nimeltä org.json, käytettiin JSON-tietojen jäsentämiseen ja luomiseen [23].
- JUnit-testauskehystä hyödynnettiin yksikkötestien toteuttamiseen ja suorittamiseen.
- Apache HttpClient -kirjastoa käytettiin HTTP-pyyntöjen (Hypertext Transfer Protocol) luomiseen ja vastausten käsittelyyn [24].
- Apache POI -kirjastoa hyödynnettiin Microsoft Excel -asiakirjojen käsittelyyn.
- Azure Functions SDK -ohjelmistokehityspakettia (Software Development Kit) käytettiin Azure-resurssien hallintaan ja käyttöön Java-ohjelmakoodissa [25].

4.2 Apache Maven

Laskutuslisäosan toteutuksessa käytettiin avoimen lähdekoodin Apache Maven -projektinhallintakehystä. Maven-projektinhallintakehystä hyödynnettiin laskutuslisäosan Maven-projektin riippuvuuksien ja ulkoisten kirjastojen hallintaan, sekä projektirakenteen standardoimiseen ja automatisointiin.

Laskutuslisäosan Maven-projektin luomiseen hyödynnettiin Maven-arkkityyppejä. Nämä arkkityypit toimivat projektipohjina, joiden avulla projektirakennetta ja peruskonfiguraatioita ei tarvitse määritellä itse alusta alkaen. Esimerkiksi maven-archetype-simple-arkkityypillä saadaan generoitua yksinkertainen Maven-projekti. [26.]

Maven-projektinhallintakehys hyödyntää XML-muotoista POM-tiedostoa (Project Object Model), joka toimii Maven-projektin keskeisenä konfiguraatiotiedostona. POM-tiedosto sisältää tärkeitä tietoja Maven-projektista, kuten sen metatiedot, paketointiasetukset ja riippuvuudet. POM-tiedostolla mahdollistetaan Maven-projektin rakenteen ja asetusten hallinta keskitetysti yhdessä tiedostossa. [27; 28.]

Maven-projektinhallintakehyksen avulla helpotettiin laskutuslisäosan ulkoisten kirjastojen käyttöönottoa. Ulkoiset kirjastot kirjattiin POM-tiedostoon, jonka jälkeen Maven-projektinhallintakehys latasi ne laskutuslisäosan Maven-projektin käyttöön. Tämän avulla pystyttiin välttämään ulkoisten kirjastojen manuaalinen asennus. [27; 28.]

Esimerkkikoodi 1 kuvastaa, kuinka Azure Functions SDK -ohjelmistokehityspaketti määriteltiin laskutuslisäosan Maven-projektin POM-tiedostoon. Ohjelmistokehityspaketin riippuvuus (dependency) kirjattiin POM-tiedoston riippuvuudet-osioon (dependencies). Riippuvuuden ryhmätunniste (groupId), artefaktitunniste (artifactId) ja versio (version) auttavat Mavenia tunnistamaan ja lataamaan oikeat kirjastoversiot Maven-projektin käyttöön. [29; 27.]

```
<dependencies>
     <dependency>
          <groupId>com.microsoft.azure.functions</groupId>
          <artifactId>azure-functions-java-library</artifactId>
          <version>3.0.0</version>
     </dependency>
</dependencies>
```

Esimerkkikoodi 1. Azure Functions SDK -ohjelmistokehityspaketin riippuvuuden määrittely Apache Maven -projektinhallintakehyksen POM-tiedostossa [29].

Sonatypen hallinnoima Maven-keskuskirjasto (Maven Central Repository) tarjosi kätevän tavan etsiä ja ladata ulkoisia kirjastoja laskutuslisäosan Maven-projektin käyttöön. Keskuskirjasto tarjoaa tietoa kirjastoista sekä esimerkkikoodin 1 mukaisia valmiita koodilohkoja, jotka voidaan helposti kopioida Maven-projektin POM-tiedostoon. [21.]

4.3 Redmine

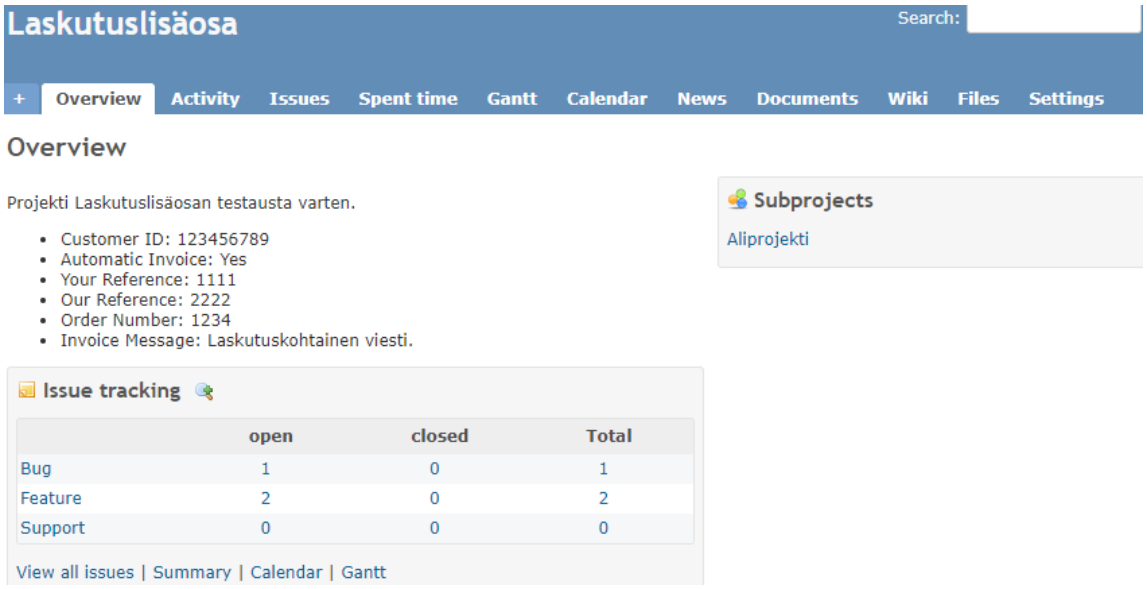
Systencess Oy hyödyntää Redmine-projektinhallintaohjelmistoa sisäisessä kehityksessä, asiakasprojektien hallinnassa ja laskutuksessa. Kuten luvussa 1 Johdanto mainittiin, laskutuslisäosan tavoitteena oli korvata myyntilaskujen manuaalinen luonti generoimalla myyntilaskut automaattisesti Redmine-projektinhallintaohjelmiston asiakasprojekteista.

Redmine-projektinhallintaohjelmisto on avoimen lähdekoodin Ruby on Rails -verkkosovellus, joka tarjoaa useita avainominaisuuksia projektinhallintaan, kuten tapahtumienseurannan, ajanseurantaominaisuudet, wikisivustot ja raportoinnin [30]. Kuvassa 6 nähdään Redmine-projektinhallintaohjelmiston esimerkkiasiakasprojektin aloitussivu. Aloitussivu sisältää yleiskatsauksen (overview), joka sisältää asiakasprojektin kuvauksen sekä sen mukautetut kentät (custom fields). Esimerkkiasiakasprojektissa määriteltyjä mukautettuja kenttiä hyödynnettiin myös laskutuslisäosassa. Esimerkkiasiakasprojektin ja laskutuslisäosan mukautetut kentät sisältävät

- asiakasnumeron (customer id)
- automaattisen laskutuksen tilan (automatic invoice)

- viitteenne (your reference)
- viitteemme (our reference)
- tilausnumeron (order number)
- laskutuskohtaisen viestin (invoice message).

Mukautettujen kenttien avulla voitiin antaa laskutuslisäosassa generoitaville myyntilaskuille lisätietoja, joita Redmine-projektinhallintaohjelmisto ei oletuksena tarjonnut. Redmine-projektinhallintaohjelmiston asiakasprojekteille voidaan määritellä tapahtumia (issue), joita voidaan hallinnoida tapahtumienseurannasta (issue tracking) [30]. Tapahtumat voivat olla esimerkiksi asiakasprojektin kehitystyöhön liittyviä toiminnallisuuksia tai ongelmatilanteiden tukipyyntöjä. Itse tapahtumille luodaan aikamerkintöjä, joita voidaan seurata ajanseurannassa (time tracking).

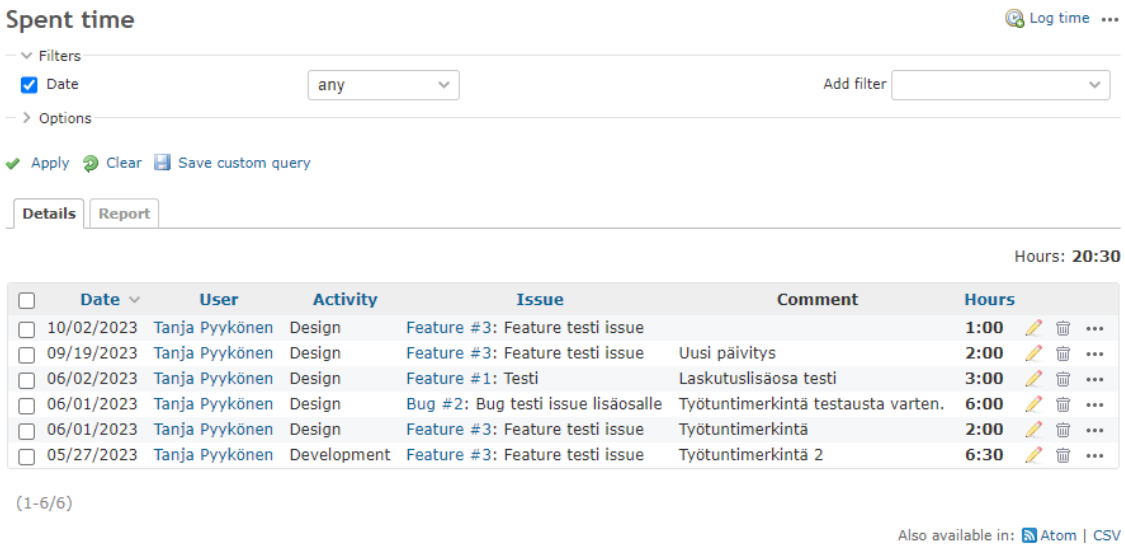


Kuva 6. Redmine-projektinhallintaohjelmiston esimerkkiasiakasprojektin aloitussivu.

Laskutuslisäosa generoi jokaisesta asiakasprojektista oman myyntilaskunsa. Asiakasprojektin tapahtumat toimivat generoidun myyntilaskun laskuriveinä. Tapahtumien aikamerkintöjen tuntimäärän ja mukautetun kentän avulla määritellään laskurivin laskutettava summa. Systencess Oy käyttää tapahtumilleen

mukautettua kenttää nimeltä sovittu hinta, jolla määritellään asiakasprojektin tuntihinta.

Ajanseurannan yksityiskohdista voidaan seurata ja hallinnoida asiakasprojektien tapahtumien aikamerkintöjä [4]. Kuvassa 7 nähdään esimerkkiasiakasprojektin tapahtumiin tehdyt aikamerkinnät. Aikamerkintä sisältää muun muassa päivämäärän, jolloin se on tehty, ja tapahtuman, jolle kyseinen aikamerkintä on kirjattu. Tunnit kertovat tapahtumaan käytetyn ajan tunteina.



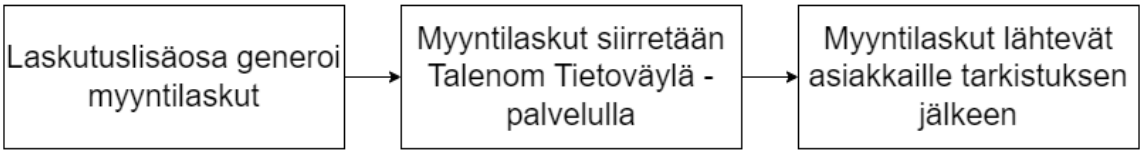
Kuva 7. Redmine-projektinhallintaohjelmiston esimerkkiasiakasprojektin tapahtumiin tehdyt aikamerkinnät.

Myyntilaskujen generointiin tarvittavat tiedot haettiin asiakasprojekteista Redmine-projektinhallintaohjelmiston REST-rajapinnan (Representation State Transfer) avulla, joka tarjoaa CRUD-toiminnot rajapinnan resursseille. Rajapinnan projektit- ja tapahtumat-resursseja hyödynnettiin asiakasprojektien, tapahtumien ja mukautettujen kenttien hakuun. Tapahtumien aikamerkinnät haettiin aikamerkinnät-resurssin avulla, ja tiedostot-resurssia hyödynnettiin asiakasrekisterin hakemiseen, joka sisältää laskutettavien asiakkaiden tietoja. [31; 32; 33.]

4.4 Talenom

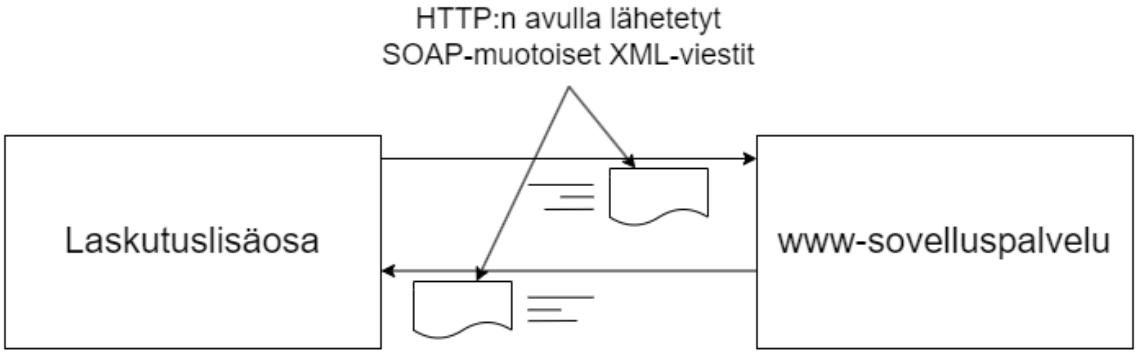
Talenom-tilitoimistopalvelu tarjoaa taloushallinnon palveluita ja kirjanpitoa yrityksille [34]. Talenom-tilitoimistopalvelu oli Systencess Oy:n valmiiksi käytössä oleva palvelu, jota yritys käyttää asiakkaiden laskuttamiseen ja kirjanpitoon. Redmine-projektinhallintaohjelmiston asiakasprojekteihin käytetyt työtunnit laskutetaan käyttämällä Talenom-tilitoimistopalvelun myyntilaskutusjärjestelmää.

Kuvassa 8 havainnollistetaan, kuinka laskutuslisäosa hyödynsi Talenom Tietoväylä -palvelua generoitujen myyntilaskujen lähetykseen Talenom-tilitoimistopalvelulle. Talenom Tietoväylä -palvelu mahdollisti laskutuslisäosan ja Talenom-tilitoimistopalvelun keskustelun sähköisesti. Generoidut myyntilaskut siirrettiin Talenom-tilitoimistopalvelulle Talenom Tietoväylä -palvelun www-sovelluspalvelun myyntilaskurajapinnan avulla. Myyntilaskujen siirron jälkeen, kuten luvussa 3.5 Käyttötapaus 4: Lähetä myyntilaskut Talenom-tilitoimistopalvelulle mainittiin, laskuttaja tarkistaa myyntilaskut manuaalisesti. Tämän jälkeen myyntilaskut lähtevät asiakkaille automaattisesti myyntilaskutusjärjestelmästä. [34; 35.]



Kuva 8. Myyntilaskujen lähetys myyntilaskutusjärjestelmään Talenom Tietoväylä -palvelun avulla [35].

Laskutuslisäosa lähettää generoidut myyntilaskut XML-viesteinä Talenom-tilitoimistopalvelun www-sovelluspalvelulle HTTP-protokollan avulla kuvan 9 esittämällä tavalla. Myyntilaskun sisältämä XML-viesti muotoillaan SOAP-syntaksin (Simple Object Access Protocol) mukaisesti. Myyntilaskujen vastaanottamisen jälkeen www-sovelluspalvelu lähettää vastauksen laskutuslisäosalle. [36.]



Kuva 9. Kommunikointi laskutuslisäosan ja Talenom-tilitoimistopalvelun www-sovelluspalvelun välillä [36].

Laskutuslisäosa käyttää SOAP-syntaksin 1.1 versiota myyntilaskujen lähettämiseen Talenom-tilitoimistopalvelulle. Esimerkkikoodissa 2 kuvataan laskutuslisäosan lähettämä HTTP POST -pyyntö. Pyyntö sisältää SOAP-viestin, jossa määritellään uuden myyntilaskun lisääminen Talenom-tilitoimistopalvelun www-sovelluspalveluun. SOAP-viestissä lähetetään Talenom-tilitoimistopalvelun käyttäjätunnus (userName), salasana (userPw) ja XML-muodossa oleva myyntilasku (xmlInvoice).

```
POST /InvoiceService.asmx HTTP/1.1
Host: verkkopalvelu3.talenom.fi
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://www.talenom.fi/NewInvoice"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://sche-
mas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <NewInvoice xmlns="http://www.talenom.fi/">
      <userName>string</userName>
      <userPw>string</userPw>
      <xmlInvoice>xml</xmlInvoice>
    </NewInvoice>
  </soap:Body>
</soap:Envelope>
```

Esimerkkikoodi 2. Talenom-tilitoimistopalvelun myyntilaskurajapinnan HTTP POST -pyyntö myyntilaskujen lähettämiseen [37].

Myyntilaskuilla olevien tietojen tulee noudattaa Talenom-tilitoimistopalvelun tuottamaa XML-skeemaa, joka määrittelee esimerkiksi myyntilaskujen pakolliset

tiedot, niiden järjestyksen ja merkkirajoitukset. Myyntilasku voi sisältää useita elementtejä, jotka kuvaavat sen tietoja, mutta sen on sisällettävä vähintään pakolliset elementit. Laskutuslisäosa käytti myyntilaskuille sekä pakollisia että vapaaehtoisia elementtejä. Käytettyjä elementtejä oli runsaasti ja laskutuslisäosan käyttämät elementit ovat nähtävissä liitteen 1 taulukosta. [35.]

4.5 Microsoft Azure

Jotta laskutuslisäosaa voitiin suorittaa automaattisesti, tarvittiin jokin alusta tai palvelu, joka kykenisi suorittamaan laskutuslisäosan Java-ohjelmakoodia pyynnöstä. Tähän tarkoitukseen käytettiin Microsoft Azure -pilvipalvelualustaa ja sen tarjoamaa Azure-funktiopalvelun (Azure Function App) Azure-funktiota (Azure Function) [25]. Azure-pilvipalvelualusta oli Systencess Oy:n valmiiksi käytössä oleva ympäristö.

Microsoft Azure -pilvipalvelualusta tarjoaa laajan valikoiman pilvipalveluita, kuten laskentaa, tietokantoja ja kehitystyökaluja. Azure-pilvipalvelualusta mahdollistaa sovellusten kehittämisen, suorittamisen ja hallinnoinnin pilvessä ilman, että käyttäjän tarvitsee huolehtia fyysisestä infrastruktuurista. [25.]

Esimerkkikoodissa 3 esitetään yksinkertaistettu versio laskutuslisäosan Azure-funktiosta, joka käynnistää laskutuslisäosan toiminnallisuudet ja prosessoi laskutettavat asiakasprojektit yksi kerrallaan. HttpTrigger -annotaatiolla määritellään Azure-funktion käyttäytyminen HTTP POST -pyyntöön. Tämä antaa ohjeet Azure-funktiolla siitä, miten sen tulee käynnistä ja toimia. Laskutettavia asiakasprojekteja käsittelevä Azure-funktio hakee pyynnön kyselyparametrit (query), joihin määriteltiin muun muassa käytettävien rajapintojen tunnistetiedot. Tämän jälkeen Azure-funktio generoi, validoi ja lähettää myyntilaskut yksi kerrallaan toistorakenteessa. Lopuksi esimerkkikoodi luo HTTP-vastauksen, jonka se lähettää HTTP-pyynnön tekijälle. [38; 39.]

```
@FunctionName("processProjects")
public HttpResponseMessage run(
     @HttpTrigger(name = "req", methods = {
          HttpMethod.POST}, authLevel = AuthorizationLevel.FUNCTION)
          HttpRequestMessage<Optional<String>> request,
          final ExecutionContext context) {
               String redmineApiKey = request.getQueryParameters()
                    .get("redmineApiKey");
               // More query parameters

               List<Project> projects = redmineApi
                    .getProjects(redmineBaseUrl, redmineApiKey);
               for (Projects project : projects) {
                    // Methods for generating, validating, and sending
                    invoices
               }

               return request.createResponseBuilder(HttpStatus.OK)
                    .body("Projects processed successfully.")
                    .build();
          }
```

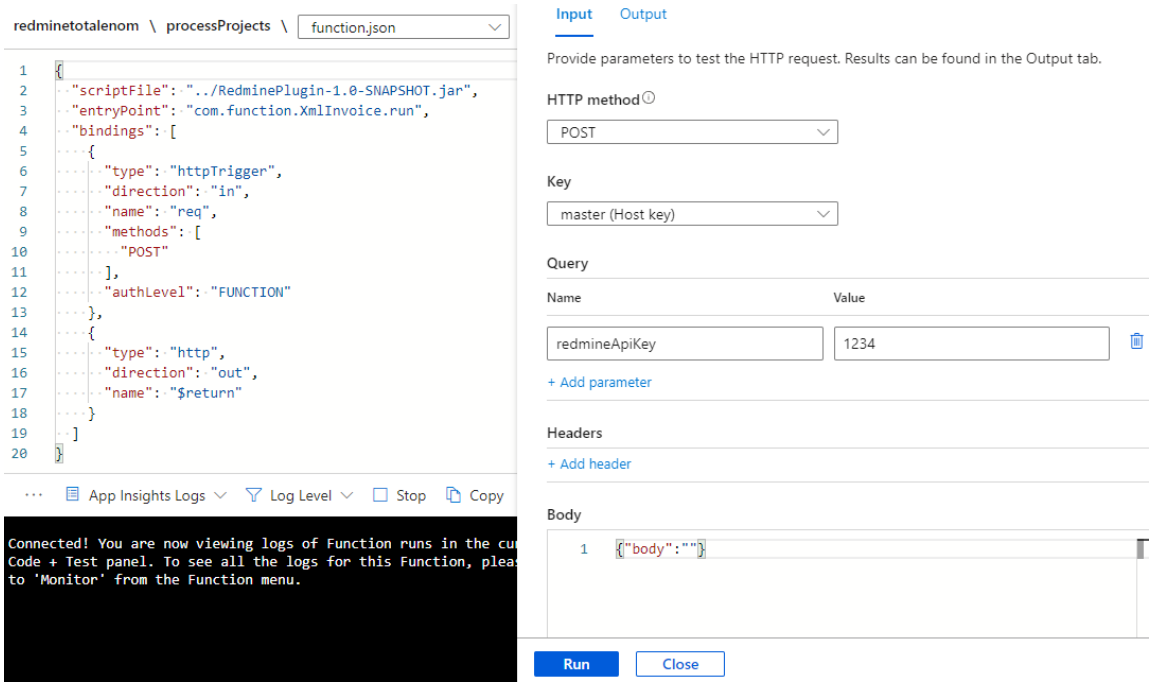
Esimerkkikoodi 3. Laskutettavia asiakasprojekteja käsittelevän Azure-funktion määrittely Java-ohjelmointikielellä [38].

Kun Azure-funktio julkaistaan Azure-funktiopalveluun, joka isännöi ja hallinnoi Azure-funktioita, se voidaan käynnistää esimerkiksi Azure-portaalin (Azure Portal) kautta. Azure-portaali tarjoaa verkkopohjaisen hallintatyökalun Azure-palveluiden hallintaan ja valvontaan. Laskutuslisäosan Azure-funktiota suoritettiin laskutuslisäosan kehitysvaiheessa ja testauksessa Azure-portaalin kautta. [38; 40; 41.]

Kuvassa 10 nähdään Azure-portaalin Code + Test -paneeli, jossa laskutuslisäosan Azure-funktiota suoritettiin kehitystyön ja testauksen aikana. Laskutuslisäosan Azure-funktion function.json -konfiguraatiotiedosto sisältää tietoja siitä, miten Azure-funktio reagoi saamiinsa syötteisiin ja millaisia tulosteita se tuottaa. Tämä konfiguraatiotiedosto generoitui automaattisesti Azure-funktion luonnin yhteydessä. [42; 43.]

Syötevälilehdellä (input) määritellään Azure-funktion HTTP-metodi, avain valtuutusta varten ja kyselyparametrit. Esimerkin vuoksi syötevälilehden kyselyparametreihin on kirjattu vain yksi parametri nimeltä redmineApiKey. Todellisuudessa tähän sijoitettiin muun muassa Redmine-projektinhallintaohjelmiston ja Talenom-tilitoimistopalvelun rajapintojen tunnistetiedot. Azure-funktion

suorittamisen jälkeen HTTP-pyynnön vastausta voidaan tarkastella tulostusvälilehdeltä (output).



Kuva 10. Laskutuslisäosan Azure-funktion käynnistäminen Azure-portaalin Code + Test -paneelista.

Kuten luvussa 1 Johdanto mainittiin, tämän insinöörityön yksi tavoitteista oli, että laskutuslisäosan toiminnallisuudet tulisi suorittaa automaattisesti ja ajastetusti kerran kuukaudessa. Tähän tarkoitukseen laskutuslisäosan Azure-funktion manuaalinen suorittaminen Azure-portaalissa ei ollut sopiva tapa. Azure-funktiota tulisi olla mahdollista suorittaa automaattisesti kerran kuukaudessa.

4.6 Microsoft Power Automate

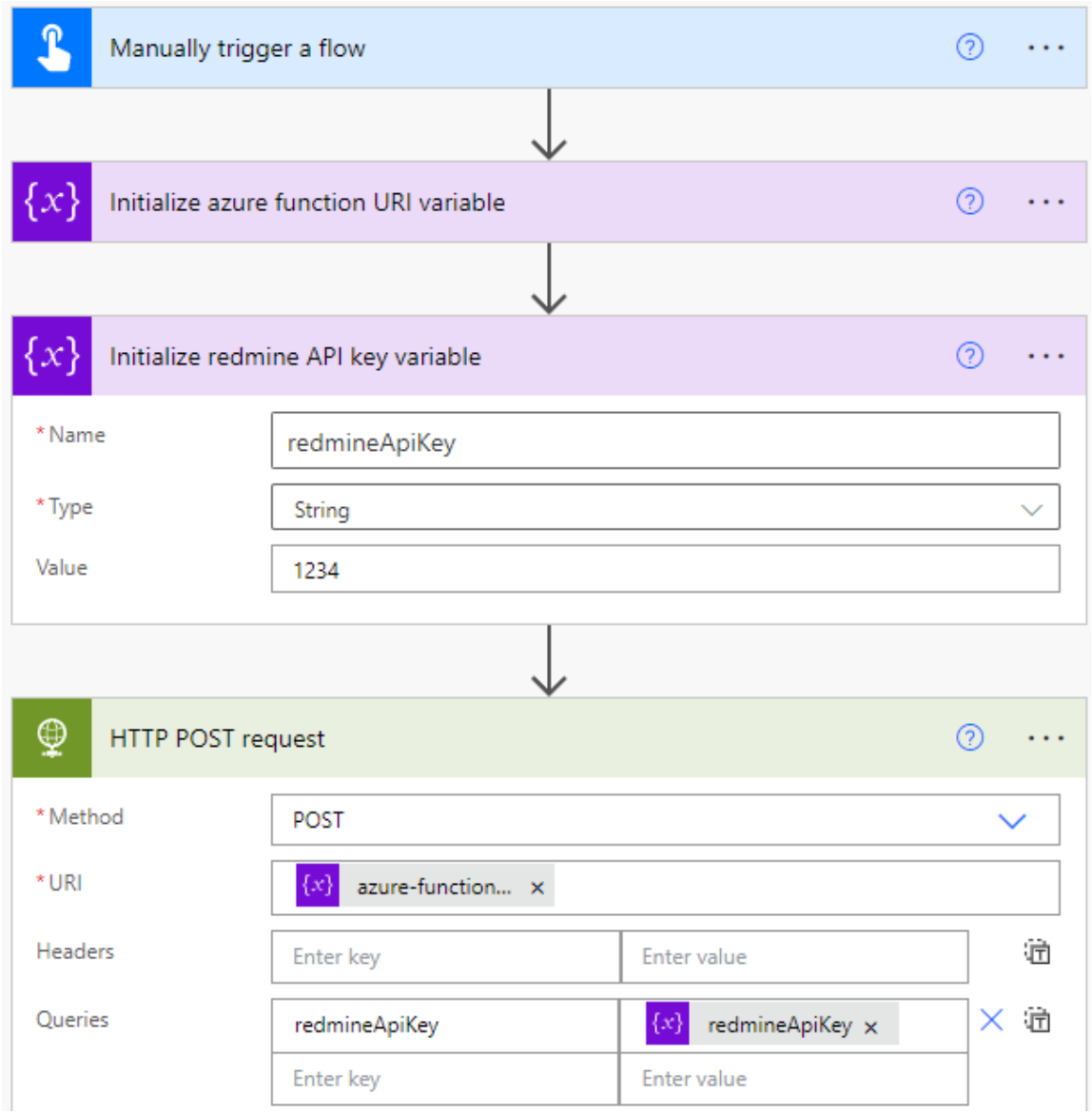
Jotta laskutuslisäosalla voisi laskuttaa yrityksen asiakkaat automaattisesti ja ajastetusti kerran kuussa, tarvittiin automaatioympäristö, joka automatisoisi laskutuslisäosan Azure-funktion käynnistyksen. Automaatioympäristön tulisi lähettää HTTP-pyyntö laskutuslisäosan Azure-funktion URI-osoitteeseen ajastetusti kerran kuukaudessa kyselyparametrien kera. Kuten luvussa 2.2 Suunniteltu laskutusprosessi ja tavoitteet mainittiin, Power Automate -työnkulkua käytettiin

Azure-funktio kanssa laskutuslisäosan automatisointiin. Microsoft Power Automate -pilviautomaatioympäristö oli Systencess Oy:n valmiiksi käytössä oleva ympäristö.

Power Automate on pilviautomaatioympäristö, joka integroituu kätevästi muihin Microsoft-sovelluksiin, kuten Microsoft Azure -pilvipalvelualustaan. Power Automate -työnkulun (workflow) avulla voidaan automatisoida erilaisia tehtäviä ja prosesseja. [44; 45.]

Power Automate -työnkulku muodostuu erityyppisistä askeleista (steps) ja ne voivat olla tyypiltään esimerkiksi laukaisimia (triggers), tapahtumia (actions) tai ehtoja (conditions). Itse Power Automate -työnkulut voivat olla tyypiltään erimerkiksi manuaalisesti painikkeella käynnistettäviä tai ajastettuja. [45.]

Kuva 11 esittää yksinkertaisen manuaalisesti käynnistettävän Power Automate -esimerkkityönkulun. Ensimmäisessä askeleessa määritellään manuaalinen laukaisin, joka käynnistää esimerkkityönkulun. Seuraavissa askeleissa alustetaan Azure-funktion URI-osoitteen ja Redmine-projektinhallintaohjelmiston API-avaimen muuttujat. HTTP-pyynnön askeleessa lähetetään HTTP POST -pyyntö URI-osoitteeseen, joka saadaan Azure-funktion URI-osoitteen muuttujasta. Lopuksi HTTP-pyynnön askeleen kyselyparametreihin asetetaan Redminen-projektihallintaohjelmiston API-avaimen muuttujan määrittelemä arvo.



Kuva 11. Microsoft Power Automate -pilviautomaatioympäristön manuaalinen esimerkkityönkulku.

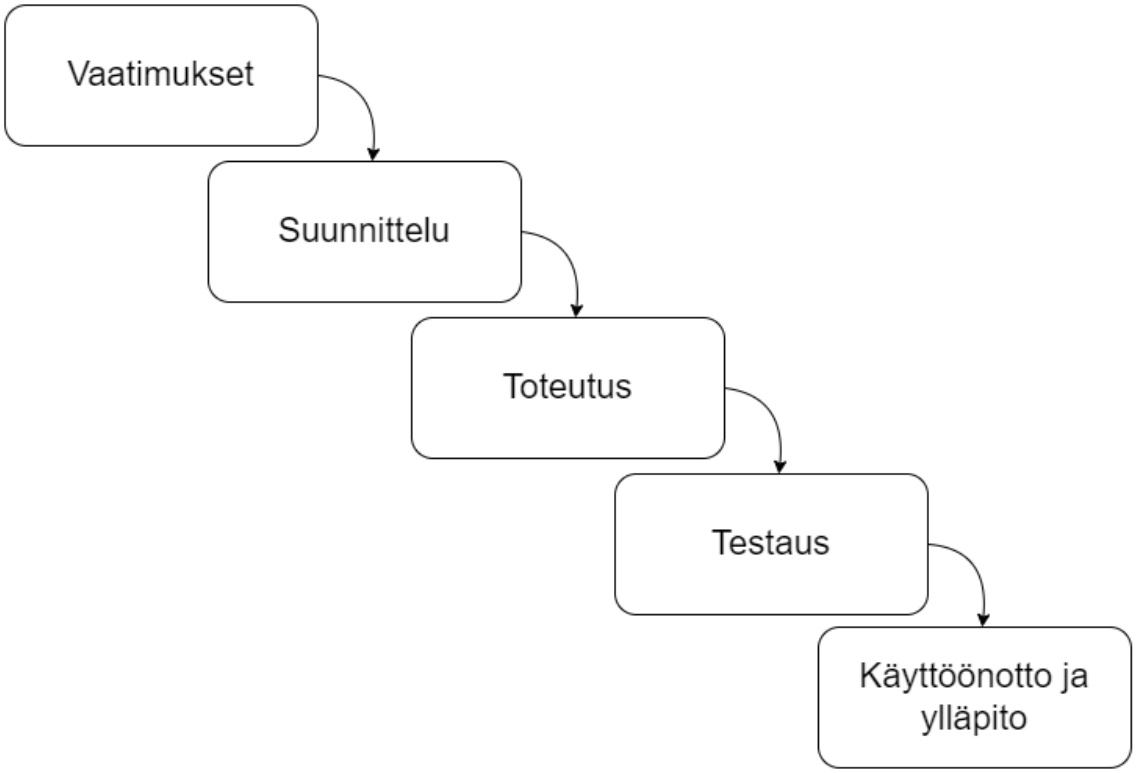
Laskutuslisäosan kehityksessä sovellettiin kuvan 11 mukaista Power Automate -työnkulkua. Laskutuslisäosan lopullisesta, ajastetusta työnkulusta, kerrotaan tarkemmin luvussa 5.7 Power Automate -työnkulun luonti.

4.7 Kehitysprosessin hallinta

Laskutuslisäosaan ja sen automatisointiin sovellettiin vesiputous-ohjelmistokehitysmallia. Vesiputousmalli on perinteinen ohjelmistokehitysmalli, jossa kehitys

etenee lineaarisesti eteenpäin vaihe vaiheelta. Jokaisen vesiputousmallin vaiheen tulee olla toteutettu ennen kuin seuraavaan vaiheeseen voidaan siirtyä. Vesiputousmalli valittiin laskutuslisäosan ja sen automatisoinnin ohjelmistokehitysmalliksi, koska niiden vaatimukset olivat selkeät ja tiedossa etukäteen, mikä sopi hyvin vesiputousmallin vaiheistettuun luonteeseen. [46.]

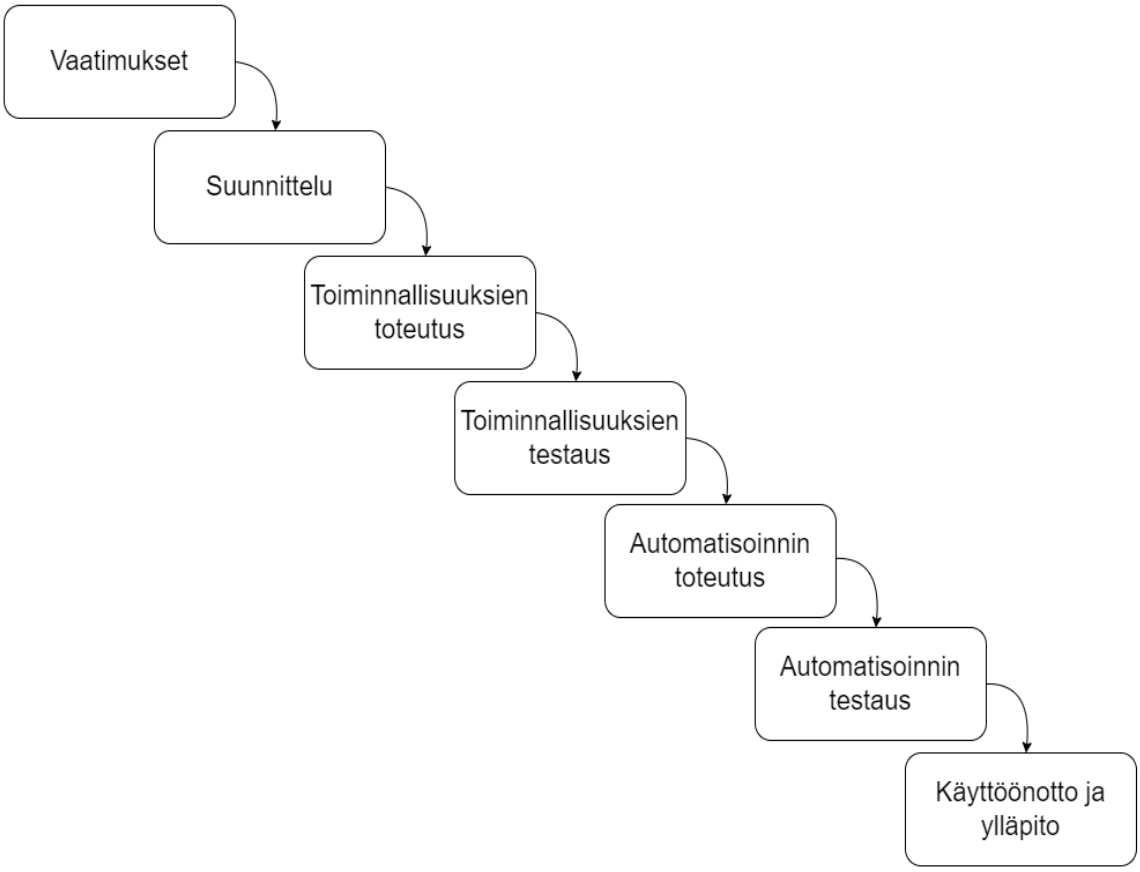
Kuvassa 12 havainnollistetaan perinteisen vesiputousmallin vaiheet. Ensimmäisessä vaiheessa määritellään toteutettavan ohjelmiston vaatimukset. Vaatimusten määrittelyn jälkeen siirrytään ohjelmiston suunnitteluun, jossa vaatimukset muunnetaan konkreettisiksi suunnitelmiksi ohjelmiston toteutusta varten. Tämän jälkeen ohjelmisto toteutetaan suunnitelmien pohjalta. Testausvaiheessa ohjelmiston eri osat kootaan yhteen ja niiden toimivuus varmistetaan. Viimeisessä vaiheessa ohjelmisto otetaan käyttöön, ja samalla mahdolliset käyttöönoton jälkeen ilmenneet virheet käsitellään. [46.]



Kuva 12. Vesiputous-ohjelmistokehitysmallin vaiheet [46].

Kuvassa 13 havainnollistetaan laskutuslisäosan ja sen automatisoinnin soveltaman vesiputousmallin vaiheita. Laskutuslisäosan toiminnallisuuksien ja automatisoinnin vaatimukset ja suunnitelmat toteutettiin kehityksen alkuvaiheessa. Tämän jälkeen ne toteutettiin ja testattiin vaiheittain. Ensin toteutettiin laskutuslisäosan toiminnallisuudet, jonka jälkeen ne testattiin yksikkötestien ja manuaalisten testien avulla.

Kun laskutuslisäosan toiminnallisuudet oli toteutettu ja testattu, siirryttiin automatisoinnin toteutukseen ja testaukseen. Automatisointia varten luotiin Azure-funktio ja Power Automate -työnkulku, jonka jälkeen automatisointia testattiin manuaalisilla testeillä. Lopuksi laskutuslisäosa integroitiin osaksi Systencess Oy:n laskutusprosessia ja käyttöönoton jälkeen ilmenneet virheet käsiteltiin.



Kuva 13. Laskutuslisäosan ja sen automatisoinnin soveltaman vesiputousmallin vaiheet [46].

Laskutuslisäosan Java-ohjelmakoodin versionhallintaan käytettiin Git-versionhallintaa ja Gerrit Code Review -työkalua. Gerrit on Git-versionhallinnan päälle rakennettu työkalu, jonka avulla laskutuslisäosan Java-ohjelmakoodia voitiin tarkastaa, ennen kuin se liitettiin koodikantaan. Gerrit tarjoaa verkkopohjaisen käyttöliittymän koodin tarkastelulle, arvioinnille ja kommentoinnille. [47.]

Kun koodikantaan tehdään muutoksia, niistä tehdään muutosehdotus Gerritiin. Tämän jälkeen muutosehdotukselle määritellään arvioija, joka tarkistaa ja antaa arvion koodista. Arviointi annetaan arviointimerkinnöillä ja niitä käytetään ilmaisemaan muutosehdotuksen hyväksymistä tai hylkäämistä. Arvioija voi antaa koodimuutoksella arviointimerkinnän +2 ja -2 väliltä. Koodimuutoksella on oltava vähintään yksi +2-arviointimerkintä, jotta koodimuutos voidaan hyväksyä ja liittää koodikantaan. [47.]

5 Toteutus

Tässä luvussa tarkastellaan laskutuslisäosan ja sen automatisoinnin toteutusta vaiheittain. Aluksi käsitellään laskutuslisäosan kehitysympäristön asennusta ja konfigurointia. Tämän jälkeen keskitytään laskutuslisäosan toiminnallisuuksien toteutusvaiheeseen, joka koostuu myyntilaskujen tietojen hausta, generoinnista, validoinnista ja lähetyksestä. Lopuksi käsitellään laskutuslisäosan automatisoinnin toteutus, joka pitää sisällään laskutettavia asiakasprojekteja käsittelevän Azure-funktion ja Power Automate -työnkulun luonnin.

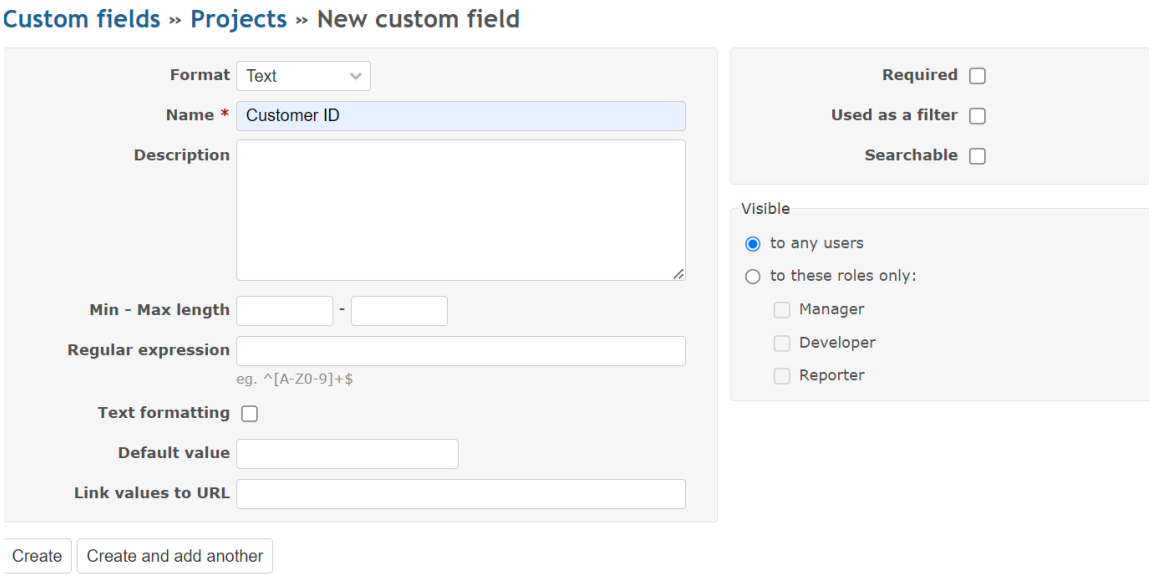
5.1 Kehitysympäristön asennus ja konfigurointi

Laskutuslisäosan ja sen automatisoinnin kehitysvaiheessa ei suoraan käytetty Systencess Oy:n Redmine-projektinhallintaohjelmiston ja Talenom-tilitoimistopalvelun ympäristöjä. Sen sijaan hyödynnettiin testiympäristöjä, jotka mahdollistivat laskutuslisäosan toiminnallisuuksien ja automatisoinnin kehityksen ja testauksen ilman, että yrityksen käytössä olevat ympäristöt olisivat vaarassa häiriintyä tai vaurioitua. [48.]

Talenom-tilitoimistopalvelu tarjosi testiympäristön heidän järjestelmiinsä, joten erillistä asennusta tai konfigurointia ei tarvinnut tehdä. Redmine-projektinhallintaohjelmisto täytyi kuitenkin asentaa ja konfiguroida paikallisesti laskutuslisäosan kehitystyötä ja testausta varten.

Redmine-projektinhallintaohjelmisto voidaan asentaa manuaalisesti tai käyttämällä kolmannen osapuolen tarjoamia Redmine-paketteja. Manuaalisen asennuksen sijaan, paikallinen Redmine-ympäristö saatiin käyttöön Bitnami Redmine Stack -paketin avulla. Paikallista Redmine-ympäristöä ajatettiin Bitnami virtuaalikonekuvan ja VirtualBox -virtuaalikoneen avulla. [49.]

Paikallinen Redmine-ympäristö täytyi vielä kuitenkin konfiguroida laskutuslisäosan tarpeisiin. Siihen luotiin mukautetut kentät, joita Systencess Oy käyttää asiakasprojekteilleen ja tapahtumilleen omassa Redmine-ympäristössään. Näistä mukautetuista kentistä kerrottiin luvussa 4.3 Redmine. Kuvassa 14 esitetään, kuinka asiakasnumeron mukautettu kenttä luotiin asiakasprojekteille paikallisessa Redmine-ympäristössä. Asiakasnumeron muodoksi (format) määriteltiin teksti (text) ja mukautettu kenttä määriteltiin näkyväksi kaikille käyttäjille.



Kuva 14. Mukautetun kentän, asiakasnumeron, luonti paikallisessa Redmine-ympäristössä.

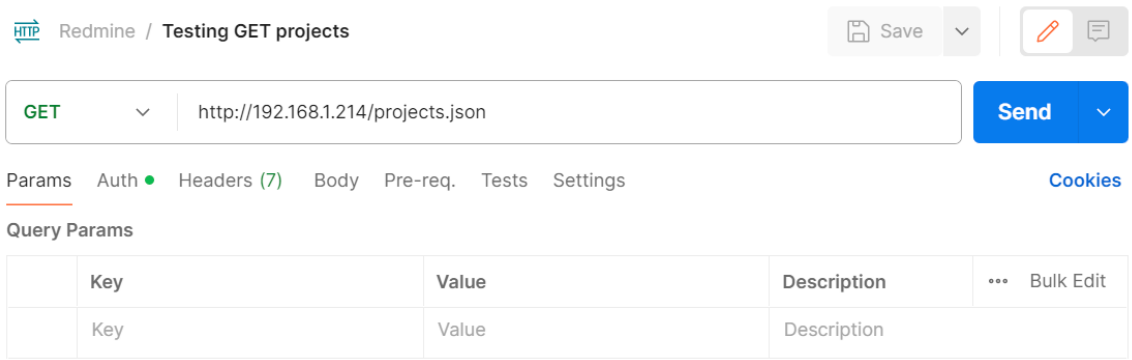
Paikalliseen Redmine-ympäristöön luotiin myös testiprojekteja, jotka vastaisivat yrityksen käytössä olevan Redmine-projektinhallintaohjelmiston asiakasprojekteja. Näille projekteille luotiin testitapahtumia ja aikamerkintöjä, jotta paikallisesta Redmine-ympäristöstä saatiin haettua tietoja myyntilaskujen generointia varten.

Ennen varsinaista kehitystyön aloittamista laskutuslisäosan toiminnallisuuksien Maven-projekti luotiin hyödyntämällä Maven-arkkityyppiä. Tämän jälkeen POM-tiedostoon konfiguroitiin Maven-projektin tiedot ja ulkoiset kirjastot.

5.2 Myyntilaskujen tietojen haku Redmine-projektinhallintaohjelmistosta

Ennen laskutuslisäosan toiminnallisuuksien toteutuksen aloittamista testattiin Redmine-projektinhallintaohjelmiston rajapinnan resursseja Postman-sovelluksen avulla. Postman-sovelluksen avulla testattiin asiakasprojektien, tapahtumien, aikamerkintöjen ja tiedostojen hakua, eli kaikkia laskutuslisäosassa käytettäviä Redmine-projektinhallintaohjelmiston rajapinnan resursseja.

Kuvassa 15 nähdään HTTP GET -pyynnön lähetys paikallisen Redmine-ympäristön projektit-resurssiin Postman-sovelluksen avulla. HTTP-pyynnön tyypiksi valittiin GET-pyyntö, jonka viereen syötettiin paikallisen Redmine-ympäristön projektit-resurssin URI-osoite. Pyynnön lähettämisen jälkeen Postman-sovellus antoi rajapinnan tuottaman vastauksen JSON-muodossa.



Kuva 15. HTTP GET -pyynnön lähetys paikallisen Redmine-ympäristön projektit-resurssiin Postman-sovelluksen avulla.

Esimerkkikoodi 4 sisältää yksinkertaistetun JSON-vastauksen paikallisen Redmine-ympäristön projektit-resurssiin tehdystä HTTP GET -pyynnöstä. Esimerkkikoodi sisältää vain yhden asiakasprojektin tilan säästämisen vuoksi. Vastauksessa saatiin tietoja asiakasprojekteista, kuten niiden yksilöivät tunnisteet (id), nimet (name) ja kuvaukset (description). Vastauksessa saatiin myös asiakasprojekteille määritetyt mukautetut kentät, joista vain yksi on sisälletty esimerkkikoodiin havainnollistamisen vuoksi.


```
{
     "projects": [
          {
               "id": 2,
               "name": "Laskutuslisäosa",
               "identifier": "laskutuslisaosa",
               "description": "Projekti laskutuslisäosan testaukseen",
               "status": 1,
               "is_public": false,
               "inherit_members": false,
               "custom_fields": [
                    {
                         "id": 4,
                         "name": "Customer ID",
                         "value": "123456789"
                    }
               ],
               "created_on": "2023-06-01T13:03:22Z",
               "updated_on": "2024-06-12T12:19:29Z"
          }
     ],
     "total_count": 3,
     "offset": 0,
     "limit": 25
}
```


Esimerkkikoodi 4. Yksinkertaistettu Postman-sovelluksen tuottama JSON-muotoinen vastaus paikallisen Redmine-ympäristön projektit-resurssin HTTP GET -pyynnöstä.

Redmine-projektinhallintaohjelmiston rajapinnan resurssien testauksen jälkeen aloitettiin laskutuslisäosan toiminnallisuuksien ohjelmointi. Ohjelmointi aloitettiin myyntilaskujen tietojen haulla Redmine-projektinhallintaohjelmiston asiakasprojekteista.

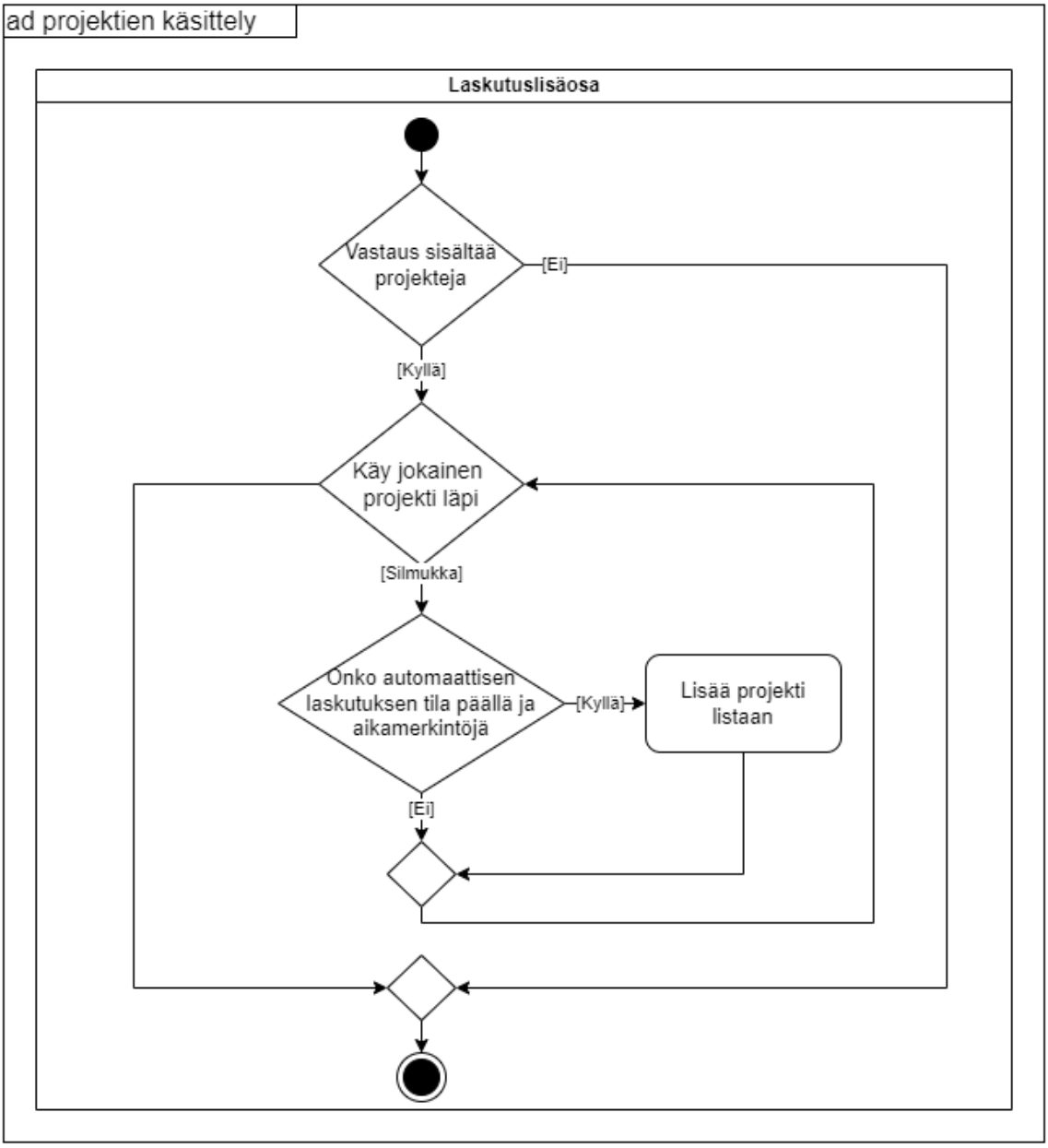
Tiedot haettiin HTTP GET -pyynnöllä Apache HttpClient -kirjastoa hyödyntäen, joka on havainnollistettu esimerkkikoodin 5 Java-metodissa. Esimerkkikoodin

Java-metodissa alustetaan HTTP-asiakasohjelma ja luodaan HTTP GET -pyyntö parametrina annettuun URI-osoitteeseen. Tämän jälkeen pyyntö lähetetään, ja lopuksi Java-metodi palauttaa HTTP GET -pyynnön tuottaman vastauksen sisällön merkkijonona.

```
private String httpGetRequest(String uri) thrown Exception {
     try (ClosableHttpClient httpClient = HttpClients
          .createDefault()) {
          HttpGet httpGet = new HttpGet(uri);
          HttpResponse response = httpClient.execute(httpGet);
          HttpEntity entity = response.getEntity();
          return EntityUtils.toString(entity);
     } catch (Exception e) {
           throw new Exception("Error occurred with the request: "
              + e.getMessage());
     }
}
```

Esimerkkikoodi 5. Java-metodi HTTP GET -pyynnön lähettämisestä parametrina annettuun URI-osoitteeseen.

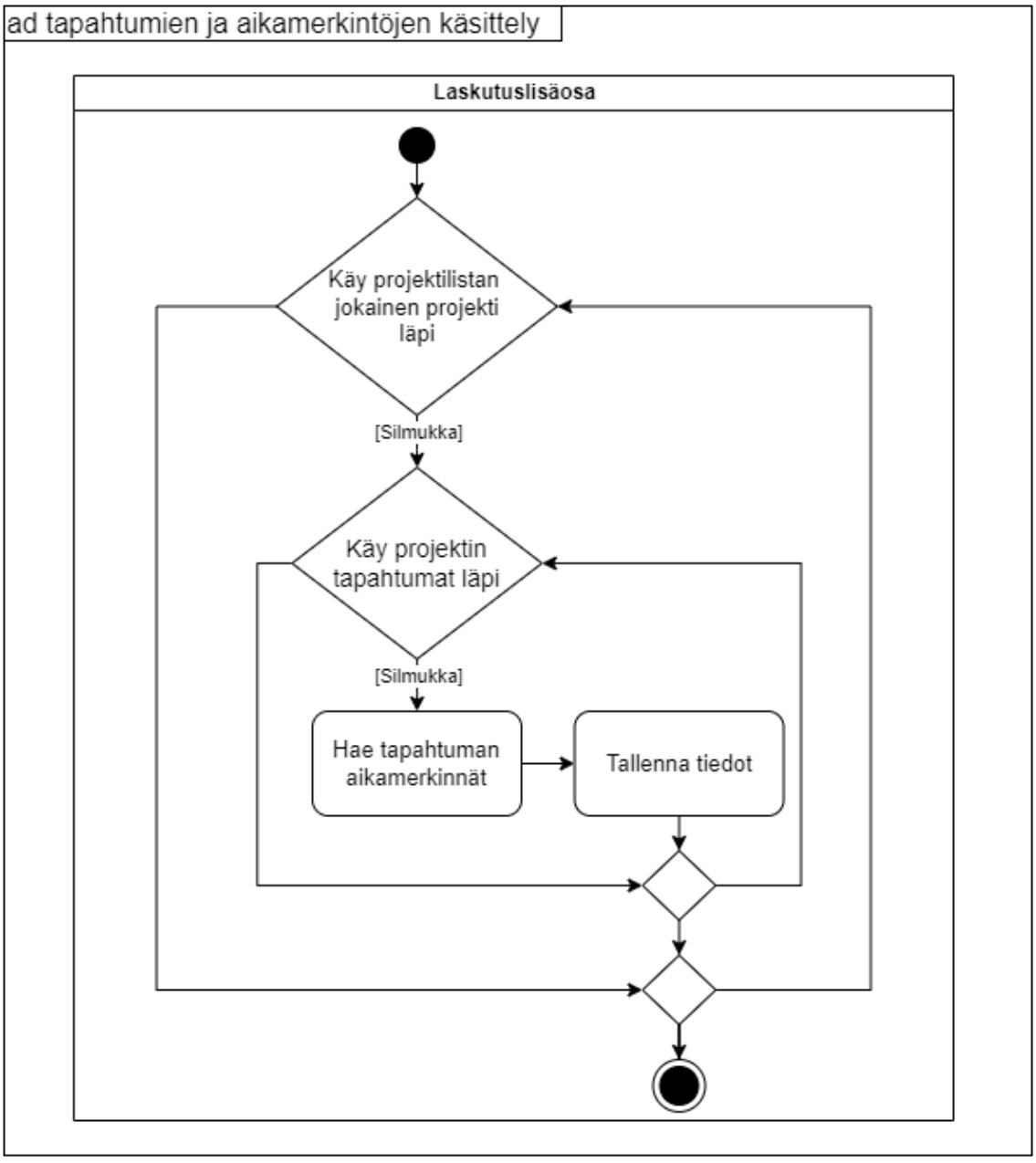
Kuvassa 16 havainnollistetaan aktiviteettikaavion avulla asiakasprojektien käsittelyä, kun niiden tiedot on haettu HTTP GET -pyynnöllä Redmine-projektinhallintaohjelmiston rajapinnan avulla esimerkkikoodin 5 mukaisella tavalla. Mikäli rajapinnan tuottama vastaus sisältää asiakasprojekteja, laskutuslisäosa tarkistaa asiakasprojektin mukautetun kentän avulla sen automaattisen laskutuksen tila. Samalla se tarkistaa, onko kyseisen asiakasprojektin tapahtumiin tehty laskutettavan kuukauden aikana aikamerkintöjä. Mikäli molemmat ehdot täyttyvät, laskutuslisäosa ottaa asiakasprojektin tiedot talteen laskutusta varten lisäämällä sen asiakasprojekteja sisältämään listaan.



Kuva 16. Aktiviteettikaavio asiakasprojektien käsittelystä.

Asiakasprojektien käsittelyn jälkeen laskutuslisäosa haki laskutettavien asiakasprojektien tapahtumat ja niiden aikamerkinnät HTTP GET -pyynnöllä esimerkkikoodin 5 mukaisella tavalla. Kuvassa 17 esitetään tapahtumien ja aikamerkintöjen käsittely aktiviteettikaavion avulla. Laskutuslisäosa aloittaa tapahtumien ja aikamerkintöjen käsittelyn käymällä asiakasprojekteja sisältävä lista läpi. Se käy asiakasprojektin tapahtumat läpi toistorakenteessa ja hakee niiden aikamerkinnät. Tapahtumien aikamerkinnöistä haetaan vain ne aikamerkinnät, jotka on

tehty laskutettavan kuukauden aikana. Lopuksi laskutuslisäosa tallentaa tarvittavat tiedot tapahtumista ja niiden aikamerkinnöistä myyntilaskujen generointia varten.



Kuva 17. Aktiviteettikaavio tapahtumien ja aikamerkintöjen käsittelystä.

Asiakkaiden tietoja sisältävä Excel-muotoinen asiakasrekisteri haettiin Redmine-projektinhallintaohjelmiston rajapinnan tiedostot-resurssilla.

Asiakasrekisterin haun jälkeen, se muunnettiin Apache POI -kirjaston avulla XSSFWorkbook-olioksi, jotta sen tietoja voitiin käsitellä.

Esimerkkikoodi 6 kuvaa yksinkertaistetun Java-metodin, jossa Apache POI -kirjaston avulla luodaan uusi XSSFWorkbook-olio, joka edustaa Redmine-projektinhallintaohjelmistosta haettua Excel-muotoista asiakasrekisteriä. Asiakasrekisterin tiedot luetaan HTTP-vastauksesta ja vastaus tallennettaan byte-taulukkoon. Tämän jälkeen byte-taulukko muunnetaan InputStream-olioksi, jonka avulla luodaan uusi XSSFWorkbook-olio. Lopuksi XSSFWorkbook-olio palautetaan myöhempää käsittelyä varten.

```
private Workbook createWorkbook() {
     byte[] excelBytes = EntityUtils.toByteArray(response.getEntity());
     InputStream excelInputStream =
          new ByteArrayInputStream(excelBytes);
     return new XSSFWorkbook(excelInputStream);
}
```

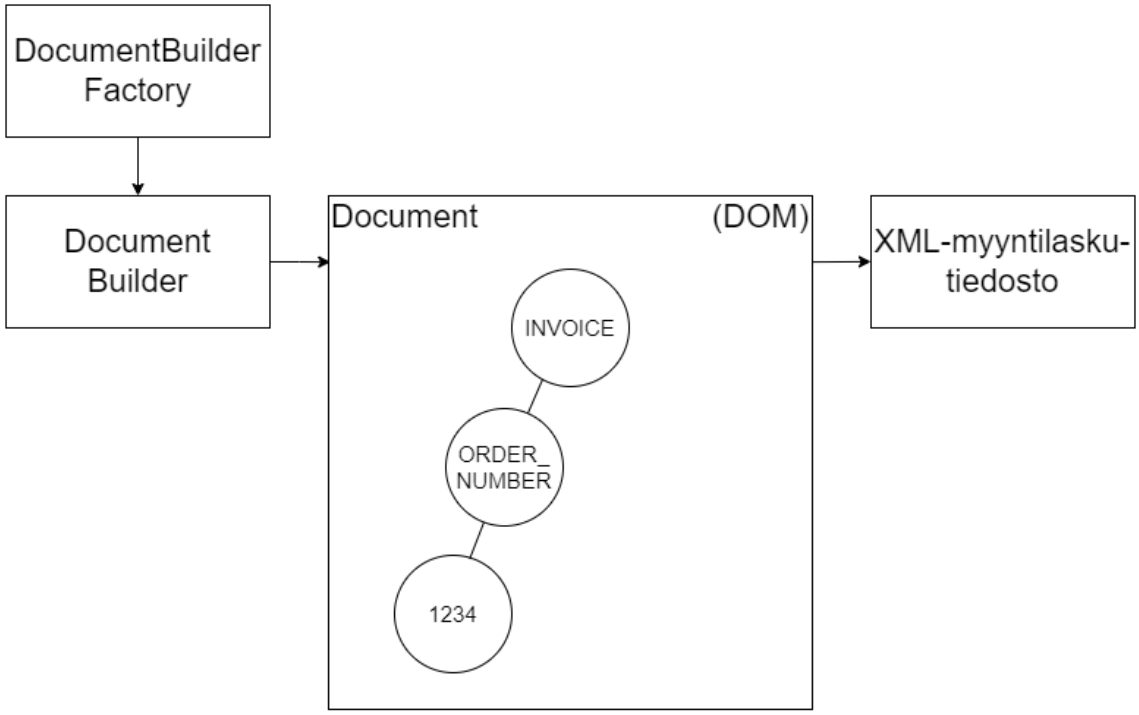
Esimerkkikoodi 6. Yksinkertaistettu Java-ohjelmakoodi Apache POI -kirjaston XSSFWorkbook-olion luonnista HTTP:n vastauksesta.

Kun asiakasrekisteri oli muunnettu XSSFWorkbook-olioksi, etsittiin asiakasprojektin asiakasnumeron avulla kyseinen asiakas XSSFWorkbook-oliosta. Asiakasrekisteristä tallennettiin asiakkaan tiedot myyntilaskujen generointia varten. Myyntilaskujen tiedot asiakasprojekteista, tapahtumista, aikamerkinnöistä ja asiakkaan tiedoista tallennettiin laskutuslisäosan Java-ohjelmakoodissa niitä vastaaviin Java-olioihin myyntilaskujen generointia varten.

5.3 Myyntilaskujen generointi

Laskutuslisäosassa myyntilaskut generoitiin hyödyntämällä Javan standardikirjaston java.xml-pakkausta, jota käytettiin XML-muotoisten myyntilaskutiedostojen luontiin, elementtien lisäämiseen ja niiden täyttämiseen Redmine-projektinhallintaohjelmistosta haetuilla tiedoilla. Myyntilaskujen generointiin käytettiin kyseisen pakkauksen JAXP-rajapintaa (Java API for XML Processing) ja JAXP-rajapinnan hyödyntämää DOM-standardia (Document Object Model) [51].

Kuvassa 18 havainnollistetaan, kuinka laskutuslisäosan XML-muotoiset myyntilaskutiedostot rakentuivat JAXP-rajapinnan ja DOM-standardin avulla. JAXP-rajapinnan DocumentBuilderFactory- ja DocumentBuilder -luokkien avulla luodaan uusi XML-dokumentti (document-olio). Tämän jälkeen luotu XML-dokumentti rakennetaan DOM-puulla erilaisten solmujen avulla. DOM-puu muodostuu juurisolmusta, elementtisolmuista ja niiden arvoista, tekstisolmuista. Kuvaan on merkitty havainnollistamisen vuoksi vain yksi myyntilaskun elementtisolmuista, ORDER_NUMBER. Todellisuudessa laskutuslisäosan INVOICE-juurisolmu sisältää liitteessä 1 esitetyt elementit. Lopuksi document-olio kirjoitetaan XML-tiedostoon, joka edustaa XML-muotoista myyntilaskua. [52.]



Kuva 18. XML-muotoisten myyntilaskutiedostojen generointi JAXP-rajapinnan ja DOM-standardin avulla [52].

Esimerkkikoodi 7 kuvaa yksinkertaistetun esimerkin siitä, kuinka laskutuslisäosa generoi myyntilaskut Java-ohjelmakoodissa. Esimerkkikoodin Java-metodi luo uuden XML-dokumentin, joka kuvastaa yhden asiakasprojektin myyntilaskua. Myyntilaskulle lisätään Talenom-tilitoimistopalvelun myyntilaskurajapinnan edellyttämä INVOICE-juurielementti. Juurielementille luodaan ORDER_NUMBER-

alielementti, jonka arvo haetaan projektioliosta. Lopuksi Java-metodi palauttaa generoidun XML-dokumentin. Tämän jälkeen, erillisessä Java-metodissa, XML-dokumentti kirjoitetaan tiedostoon, joka edustaa lopullista XML-muotoista myyntilaskua.

```
private Document createInvoiceDoc() {
     Document doc = docBuilder.newDocument();
     Element root = doc.createElement("INVOICE");
     doc.appendChild(root);

     Element orderNo = doc.createElement("ORDER_NUMBER");
     orderNo.appendChild(doc.createTextNode(project.getOrderNumber()));
     root.appendChild(orderNo);

     return doc;
}
```

Esimerkkikoodi 7. Yksinkertaistettu Java-ohjelmakoodi XML-muotoisen myyntilaskutiedoston generoinnista JAXP-rajapinnan ja DOM-standardin avulla.

Myyntilaskujen generoinnin jälkeen laskutuslisäosa validoi myyntilaskut XML-skeeman avulla. Talenom-tilitoimistopalvelu tarjosi validointia varten valmiin XML-skeeman, jota hyödynnettiin laskutuslisäosan generoimien XML-muotoisten myyntilaskujen validoinnissa.

5.4 Myyntilaskujen validointi

Kuten luvussa 4.4 Talenom mainittiin, myyntilaskujen elementit täytyi luoda oikeassa järjestyksessä XML-muotoiseen myyntilaskutiedostoon. Kun laskutuslisäosa oli generoinut XML-muotoiset myyntilaskutiedostot, ne validoitiin XML-skeeman avulla. Validoinnilla varmistettiin, että myyntilaskujen elementit olivat oikeassa järjestyksessä, luodut elementit olivat oikean tyyppisiä ja elementit täyttivät niille asetetut pituusrajoitukset.

Esimerkkikoodissa 8 esitellään yksinkertaistettu tapa validoida XML-dokumentteja Javan standardikirjaston java-xml-pakkauksen avulla. SteamSource-olioiden avulla luodaan validoitava XML-tiedosto ja validointiin käytettävä XML-skeema. SchemaFactory-oliolla mahdollistetaan XML-skeeman käsittely. SchemaFactory-olion luonnin yhteydessä asetetaan URI-osoite, joka varmistaa, että

se tukee W3C-organisaation (World Wide Web Consortium) WXS-skeemaa (W3C XML Schema). W3C-organisaation ylläpitämä WXS-skeemakieli tarjoaa oliopohjaisen tavan määritellä XML-skeemoja. Schema-olion avulla määritellään sallitut rakenteet ja elementit XML-tiedostolle XML-skeeman avulla. Tämän avulla luodaan Validator-olio, joka valvoo XML-dokumentin validointia. [53; 54.]

```
private void validateXml(String xmlFile) {
     Source xmlDoc = new StreamSource(new File(xmlFile));
     SchemaFactory factory = SchemaFactory
          .newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
     Source schemaFile = new StreamSource(new File("mySchema.xsd"));
     Schema schema = factory.newSchema(schemaFile);
     Validator validator = schema.newValidator();
     validator.validate(xmlDoc);
}
```

Esimerkkikoodi 8. Java-ohjelmakoodi XML-muotoisen dokumentin validoinnista XML-skeeman avulla [53].

Laskutuslisäosan XML-muotoiset myyntilaskutiedostot validoitiin soveltamalla esimerkkikoodin 8 esittelemää validointitapaa. Onnistuneen validointiprosessin jälkeen laskutuslisäosa lähetti myyntilaskut Talenom-tilitoimistopalvelulle.

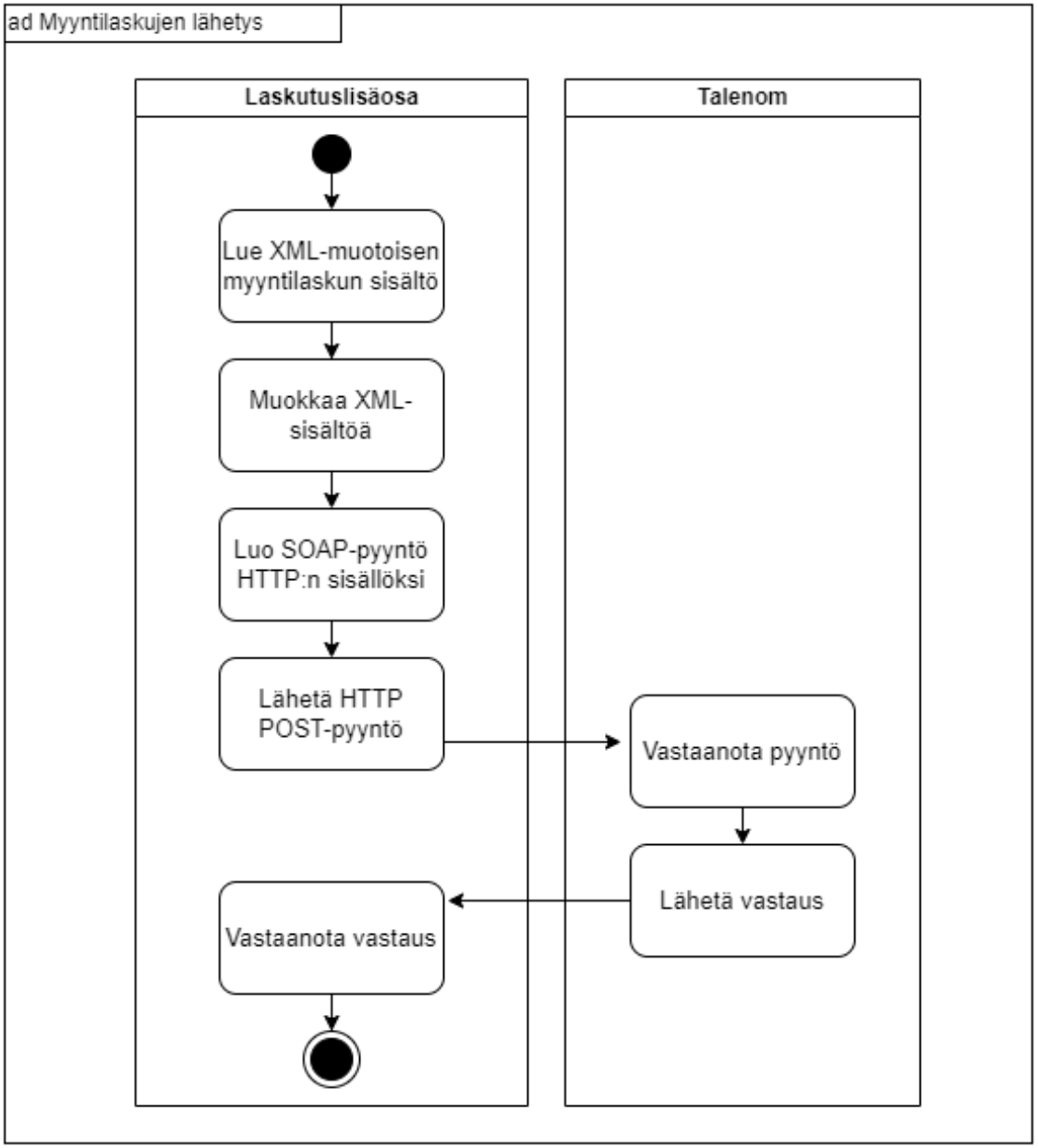
5.5 Myyntilaskujen lähetys Talenom-tilitoimistopalvelulle

Kuten luvussa 4.4 Talenom mainittiin, generoidut myyntilaskut lähetettiin Talenom-tilitoimistopalvelulle esimerkkikoodin 2 SOAP-pyynnöllä, HTTP POST -menetelmän avulla. Kuvassa 19 esitetään myyntilaskujen lähetysprosessi Talenom-tilitoimistopalvelulle aktiviteettikaavion avulla.

Laskutuslisäosa aloitti generoidun ja validoidun myyntilaskun lähetyksen lukemalla XML-muotoisen myyntilaskutiedoston sisällön. Tämän jälkeen myyntilaskun XML-sisältöä muokattiin, jotta sen INVOICE-juurielementille saatiin määriteltyä Talenom-tilitoimistopalvelun myyntilaskurajapinnan vaatima tyhjä nimiavaruus [35].

Myyntilaskun XML-sisällön muokkauksen jälkeen luotiin myyntilaskun sisältävä SOAP-pyyntö (esimerkkikoodi 2) HTTP:n sisällöksi. Tämän jälkeen myyntilasku

lähetettiin HTTP POST -pyynnöllä Apache HttpClient -kirjaston ja Talenom-tilitoimistopalvelun myyntilaskurajapinnan avulla. Lopuksi Talenom-tilitoimistopalvelun myyntilaskurajapinta vastaanotti pyynnön ja lähetti vastauksen laskutuslisäosalle.



Kuva 19. Aktiviteettikaavio myyntilaskujen lähetyksestä Talenom-tilitoimistopalvelulle.

Kuten edellä mainittiin, Talenom-tilitoimistopalvelun myyntilaskurajapinta edellytti, että myyntilaskun INVOICE-juurielementille määritellään tyhjä nimiavaruus. Java.xml-pakkaus sisältää QName-luokan, jonka avulla voidaan hallita XML-dokumenttien nimiavaruuksia. Ongelmia ilmeni laskutuslisäosan XML-muotoisten myyntilaskun tyhjän nimiavaruuden määritellyssä, sillä QName-oliota ei voitu suoraan määritellä käyttämään tyhjää nimiavaruutta. Tämän vuoksi laskutuslisäosan täytyi muokata myyntilaskun XML-sisältöä, jotta vaadittu tyhjä nimiavaruus saataisiin määriteltyä. [35; 55.]

Esimerkkikoodissa 9 esitetään, kuinka tyhjä nimiavaruus saatiin määriteltyä myyntilaskun INVOICE-juurielementille Java-ohjelmointikielen replace-metodin avulla. Replace-metodin avulla etsittiin myyntilaskun XML-sisällöstä INVOICE-juurielementti, jonka jälkeen sille lisättiin xmlns=”” -attribuutti. Muokatulla INVOICE-juurielementillä saatiin luotua Talenom-tilitoimistopalvelun myyntilaskurajapinnan vaatima tyhjä nimiavaruus myyntilaskuille.

```
String modifiedInvoice = invoice
     .replace("<INVOICE>", "<INVOICE xmlns=\"\">");
```

Esimerkkikoodi 9. Java-ohjelmakoodi tyhjän nimiavaruuden määrittämisestä myyntilaskun INVOICE-juurielementille replace-metodin avulla.

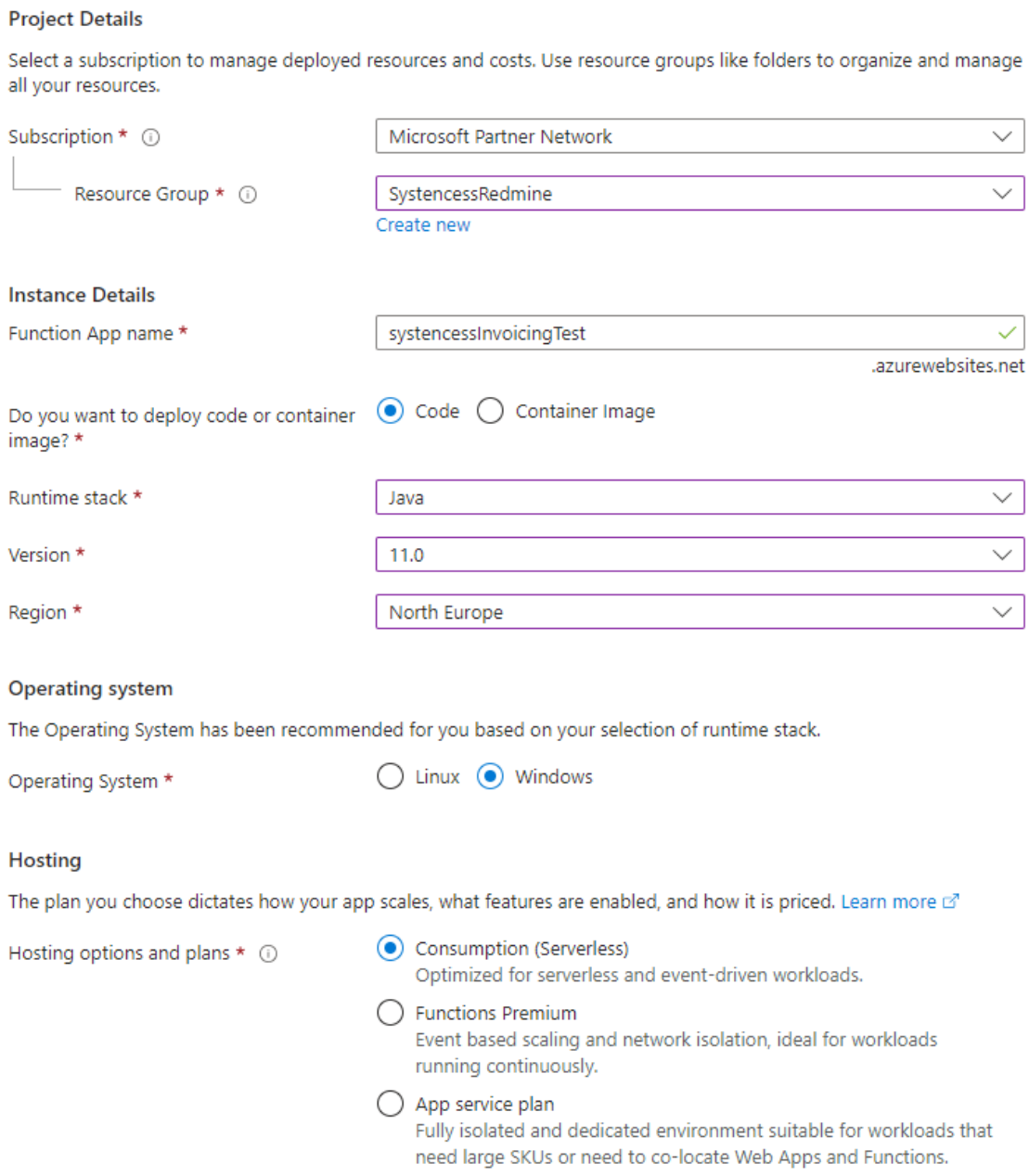
Tässä vaiheessa laskutuslisäosan toiminnallisuuksien kehittäminen saatiin valmiiksi. Tämän jälkeen siirryttiin laskutuslisäosan automatisointiin, joka koostui Azure-funktion ja Power Automate -työnkulun luonnista.

5.6 Azure-funktion luonti

Luvussa 4.5 Microsoft Azure mainittiin, että Azure-funktiopalvelua käytetään Azure-funktioiden isännöintiin ja hallintaan. Jotta laskutuslisäosan käynnistävä Azure-funktio voitiin luoda, tarvitsi ensin luoda Azure-funktiopalvelu. Kuvassa 20 nähdään, kuinka uusi Azure-funktiopalvelu luotiin Azure-portaalissa.

Azure-funktiopalvelulle määriteltiin tilaus ja resurssiryhmä, johon Azure-funktiopalvelu haluttiin luoda. Tämän jälkeen sille annettiin instanssin tiedot, kuten

nimi, julkaisutapa, julkaisutavan tiedot, käyttöjärjestelmä ja isäntäpalvelu. Isäntäpalvelulla määriteltiin, mitä ominaisuuksia laskutuslisäosan automatisointi hyödyntää ja kuinka se hinnoitellaan. Azure-funktiopalvelu määriteltiin käyttämään Consumption-isäntäpalvelua, joka tarjoaa automaattisen skaalauksen. Hinnoittelu tapahtuu Azure-funktion käytön mukaan. [56.]



Kuva 20. Azure-funktiopalvelun luonti Azure-portaalissa.

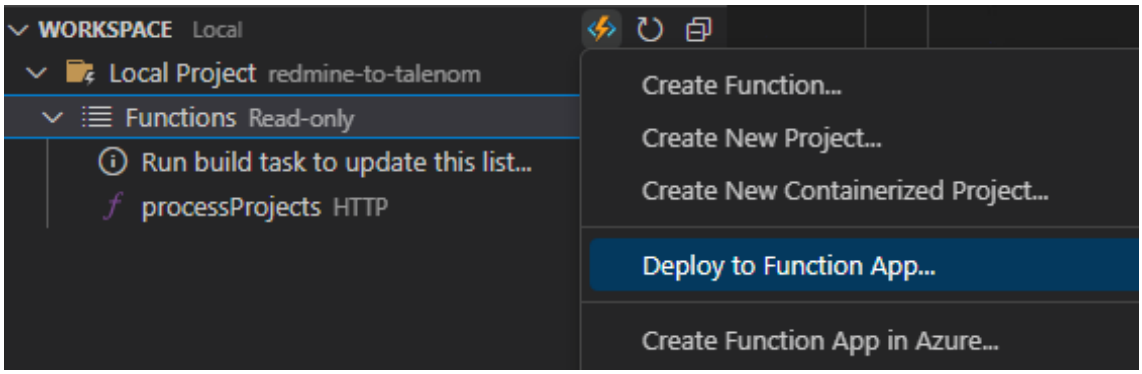
Kuten aikaisemmin mainittiin, laskutuslisäosan toiminnallisuudet ohjelmoitiin ensin Maven-projektiin. Ennen automatisoinnin toteutusta luotiin uusi Azure-projekti Maven-projektinhallintakehyksen avulla, jotta laskutuslisäosan Java-metodista, joka käsittelee laskutettavia asiakasprojekteja, voitaisiin luoda luvussa 4.5 Microsoft Azure esitetyn esimerkkikoodin 3 mukainen Azure-funktio. Azure-projektin luonnissa hyödynnettiin myös Maven-arkkityyppiä.

Esimerkkikoodissa 10 nähdään Bash-komento, jolla käynnistettiin Azure Functions -arkkityypin generointityökalu. Komennon suorittaminen edellytti, että Azure-komentotulkki on asennettuna.

```
mvn archetype:generate -DarchetypeGroupId=com.microsoft.azure
-DarchetypeArtifactId=azure-functions-archetype -DjavaVersion=11
```

Esimerkkikoodi 10. Bash-komento, joka käynnistää Azure Function -arkkityypin generointityökalun [57].

Azure-projektin luonnin jälkeen laskutuslisäosan toiminnallisuuksien Maven-projektin luokat ja tiedostot kopioitiin aiemmin luotuun Azure-projektiin, jonka jälkeen laskutettavia asiakasprojekteja käsittelevästä Java-metodista luotiin Azure-funktio. Tämän jälkeen Azure-funktio julkaistiin laskutuslisäosan Azure-funktiopalveluun. Julkaisuun hyödynnettiin Azure Functions for Visual Studio Code -laajennusta. Kuva 21 havainnollistaa, kuinka Azure-funktio julkaistiin Azure-funktiopalveluun Visual Studio Code -kehitysympäristöstä.



Kuva 21. Azure-funktion julkaisu Azure-funktiopalveluun Azure Functions for Visual Studio Code -laajennuksen avulla Visual Studio Code -kehitysympäristössä.

Laskutuslisäosan Azure-funktion julkaisun yhteydessä generoituivat sen tunnistetiedot, joiden avulla voitiin lähettää HTTP POST -pyyntö Azure-funktio URI-osoitteeseen. Näitä tunnistetietoja hyödynnettiin käynnistämään Azure-funktio automaattisesti ja ajastetusti Power Automate -työnkulun avulla.

5.7 Power Automate -työnkulun luonti

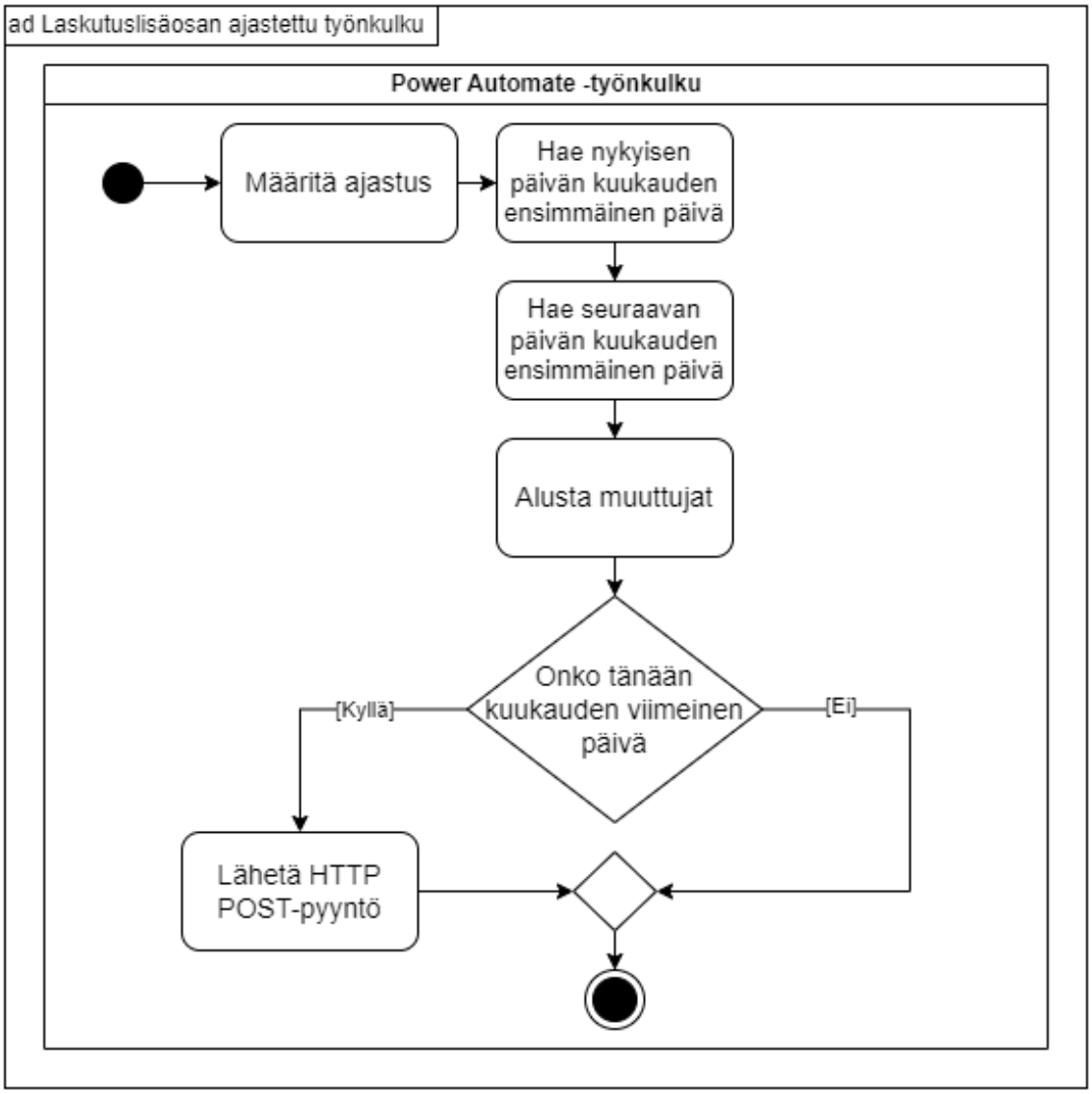
Kuten luvussa 4.6 Microsoft Power Automate mainittiin, laskutuslisäosa ja sen automatisointi hyödynsivät kehityksensä aikana manuaalista, painikkeella käynnistettävää Power Automate -työnkulkua. Lopullinen, automaattinen ja ajastettu työnkulku luotiin, kun laskutuslisäosan toiminnallisuudet ja Azure-funktio oli toteutettu ja testattu.

Laskutuslisäosan toiminnallisuudet haluttiin käynnistää ajastetusti Azure-funktion avulla jokaisen kuukauden viimeinen päivä. Power Automate ei kuitenkaan suoraan tarjonnut mahdollisuutta tähän [58]. Tämän vuoksi luotiin ajastettu Power Automate -työnkulku, joka käynnistyy kerran päivässä tarkastaen, onko nykyinen päivä kuukauden viimeinen päivä.

Kuvassa 22 havainnollistetaan aktiviteettikaavion avulla Azure-funktion ajastettu käynnistys Power Automate -työnkulun avulla. Työnkulku aloittaa toimintansa asettelemalla toistuvuuslaukaisimen, jossa se määritellään suoritettavaksi toistuvasti jokainen päivä kello 18:30. Toistuvuuslaukaisimen määrittelyn jälkeen työnkulku hakee nykyisestä päivästä kuukauden ensimmäisen päivän hyödyntämällä Power Automate -lausefunktioita. Samalla periaatteella työnkulku hakee seuraavasta päivästä kuukauden ensimmäisen päivän.

Esimerkiksi ajamalla työnkulku 30.4.2024, palautuvat päivämäärät 1.4.2024 ja 1.5.2024. Työnkulku vertailee näitä kahta päivämäärää toisiinsa määrittääkseen, onko tämä päivä, jolloin työnkulku ajetaan, kuukauden viimeinen päivä. Mikäli päivä kuukauden viimeinen päivä, työnkulku lähettää HTTP POST -pyynnön laskutuslisäosan Azure-funktioon, joka käynnistää laskutuslisäosan

toiminnallisuudet. Mikäli kyseessä ei ole kuukauden viimeinen päivä, työnkulku suorittaa toimintansa loppuun tekemättä mitään.



Kuva 22. Aktiviteettikaavio Azure-funktion ajastetusta käynnistämisestä Power Automate -työnkulun avulla.

Power Automate -työnkulun kuukauden ensimmäisten päivien hakemiseen käytettiin Microsoftin kehittämän Workflow Definition Language -kielen tarjoamia lausefunktioita. Nykyisen ja seuraavan päivän kuukauden ensimmäiset päivät saatiin startOfMonth-lausefunktion avulla, joka palauttaa kuukauden ensimmäisen päivän aikaleiman. [59.]

Päivämäärät haettiin esimerkkikoodin 11 esittämillä lausefunktioilla. Asettamalla utcNow-lausefunktion palauttama nykyisen päivän aikaleima startOfMonth-lausefunktiolle saatiin nykyisen päivän kuukauden ensimmäinen päivä. Asettamalla addDays-lausefunktion palauttama aikaleima startOfMonth-lausefunktiolla, saatiin seuraavan päivän kuukauden ensimmäinen päivä. AddDays-lausefunktiossa lisätään yksi päivä nykyiseen päivään, jolloin saada seuraavan päivän aikaleima. [59.]

```
startOfMonth(utcNow())
startOfMonth(addDays(utcNow(),1))
```

Esimerkkikoodi 11. Workflow Definition Language -kielellä määritellyt lausefunktiot nykyisen ja seuraavan päivän kuukauden ensimmäisen päivän hakuun.

Automaattisen ja ajastetun Power Automate -työnkulun toteutuksen jälkeen laskutuslisäosan toiminnallisuudet ja sen automatisointi saatiin toteutettua. Tämän jälkeen toiminnallisuudet ja automatisointi testattiin yksikkötestien ja manuaalisten testien avulla.

6 Testaus

Tässä luvussa käsitellään laskutuslisäosan ja sen automatisoinnin testausta. Testauksen käsittely alkaa käymällä läpi laskutuslisäosan toiminnallisuuksien yksikkötestausta JUnit-testauskehyksen avulla. Tämän jälkeen käsitellään laskutuslisäosan toiminnallisuuksien manuaalinen testaus ja lopuksi perehdytään laskutuslisäosan automatisoinnin testaukseen.

6.1 Toiminnallisuuksien testaus

Laskutuslisäosan toiminnallisuuksia testattiin sekä yksikkötesteillä että manuaalisilla testeillä. Yksikkötesteissä keskityttiin tarkastelemaan laskutuslisäosan laskennallista toimintaa, kun taas manuaalisissa testeissä testattiin laskutuslisäosan käyttäytymistä ja Azure-funktion toimintaa. Laskutuslisäosassa käytettäviä rajapintoja testattiin myös manuaalisesti Postman-sovelluksen avulla

kehitystyön alussa, josta kerrottiin luvussa 5.2 Myyntilaskujen tietojen haku Redmine-projektinhallintaohjelmistosta.

6.1.1 Yksikkötestaus

Laskutuslisäosan toiminnallisuuksia testattiin yksikkötesteillä JUnit-testauskehyksen avulla, mikä tarjoaa työkaluja yksikkötestien kirjoittamiseen ja suorittamiseen [60]. JUnit-testauskehys määriteltiin laskutuslisäosan POM-tiedoston riippuvuuksiin Maven keskuskirjaston tarjoamien valmiiden koodilohkojen avulla. Tämä toteutettiin samalla tavalla kuin luvun 4.2 Apache Maven esimerkkikoodissa 1, jossa määriteltiin Azure Functions SDK -ohjelmistokehityspaketin käyttöönotto.

Laskutuslisäosan yksikköteisteillä testattiin myyntilaskujen aikamerkintöjen, päivämäärien ja eräpäivien laskemista. Myyntilaskujen eräpäivät laskettiin laskutuslisäosan Java-ohjelmakoodissa lisäämällä myyntilaskun päivämäärään maksuehto. Myyntilaskun päivämäärä määritellään laskutuslisäosassa laskutettavan kuukauden viimeiseksi päiväksi.

Esimerkkikoodin 12 parametrisoidussa yksikkötestissä testataan eräpäivien laskemista. Parametrisoitu yksikkötesti mahdollistaa yksikkötestin suorittamisen useita kertoja eri argumenteilla. ValueSource-annotaatioon avulla jokainen siihen määritelty päivämäärä suoritetaan erillisenä testinä. Esimerkkikoodi sisältää vain muutaman päivämäärän havainnollistamisen vuoksi.

Yksikkötestissä luodaan kalenteri-instanssi, joka asetetaan syötetyn päivämäärän kuukauden viimeiseksi päiväksi. Kalenteri-instanssiin lisätään tämän jälkeen maksuehto (dnet). Tämän jälkeen lasketaan odotettu eräpäivä ja kutsutaan eräpäiviä laskevaa Java-metodia annetulla päivämäärällä. Lopuksi testi vertaa odotettua eräpäivää ja laskettua eräpäivää toisiinsa varmistaakseen, että eräpäiviä laskeva Java-metodi toimii odotetusti erilaisilla syötteillä. [60.]

```
@ParameterizedTest
@ValueSource(strings = { "2024-04-01", "2022-04-15", "2022-04-30" })
public void testCalculateDueDate(String date) {
     Project project = new Project();
     SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
     Date inputDate = dateFormat.parse(date);

     Calendar calendar = Calendar.getInstance();
     calendar.setTime(inputDate);
     calendar.set(Calendar.DAY_OF_MONTH, calendar
          .getActualMaximum(Calendar.DAY_OF_MONTH));
     calendar.add(Calendar.DAY_OF_MONTH, project.getDnet());

     Date expectedDueDate = calendar.getTime();
     Date calculatedDueDate = project.calculateDueDate(date);

     assertEquals(expectedDueDate, calculatedDueDate);
}
```

Esimerkkikoodi 12. JUnit-testauskehyksen parametrisoitu yksikkötesti myyntilaskujen eräpäivien laskemiseen.

Yksikkötestauksen jälkeen laskutuslisäosan toiminnallisuuksia testattiin manuaalisten testien avulla. Manuaalisten testien tarkoituksena oli varmistaa laskutuslisäosan käyttäytyminen erilaisissa tilanteissa ja testata erityisesti Azure-funktion toimintaa.

6.1.2 Manuaalinen testaus

Luvussa 4.5 Microsoft Azure mainittiin, että laskutuslisäosan Azure-funktiota suoritettiin laskutuslisäosan kehitystyön aikana Azure-portaalin Code + Test -paneelin avulla. Tätä paneelia hyödynnettiin myös laskutuslisäosan toiminnallisuuksien manuaalisessa testauksessa.

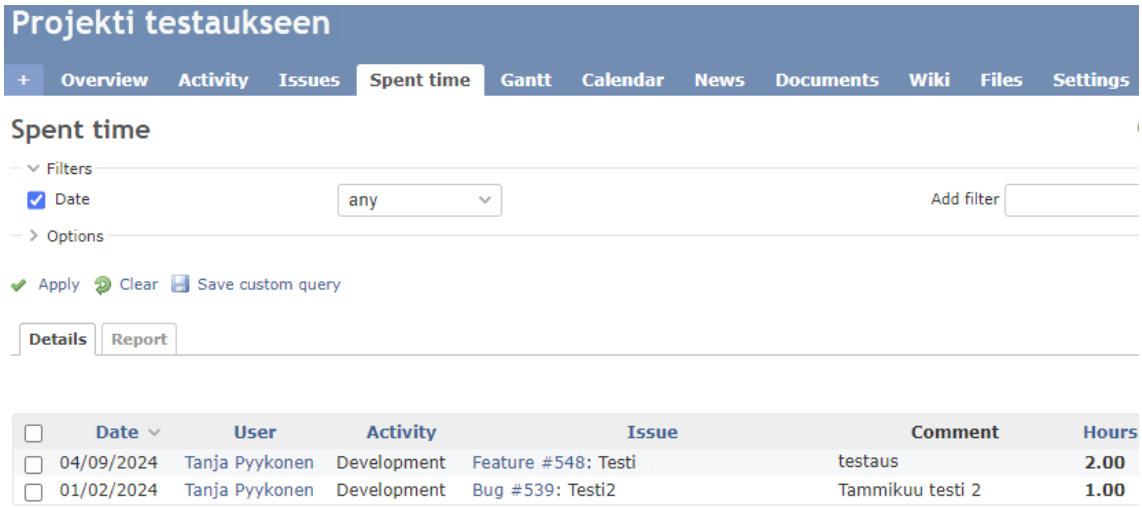
Laskutuslisäosan toiminnallisuuksia testattiin erilaisissa olosuhteissa, joissa keskityttiin sekä positiivisiin että negatiivisiin testeihin. Laskutuslisäosan toiminnallisuuksien käyttäytymistä ja Azure-funktion toimintaa testattiin muun muassa seuraavien testitapausten avulla:

- onnistunut suoritus yhdellä ja useammalla projektilla
- onnistunut suoritus yhdellä ja useammalla tapahtumalla
- onnistunut suoritus yhdellä ja useammalla aikamerkinnällä

- epäonnistunut suoritus väärillä tunnistetiedoilla
- epäonnistunut suoritus puuttuvilla asiakastiedoilla
- epäonnistunut suoritus puutteellisilla myyntilaskun tiedoilla.

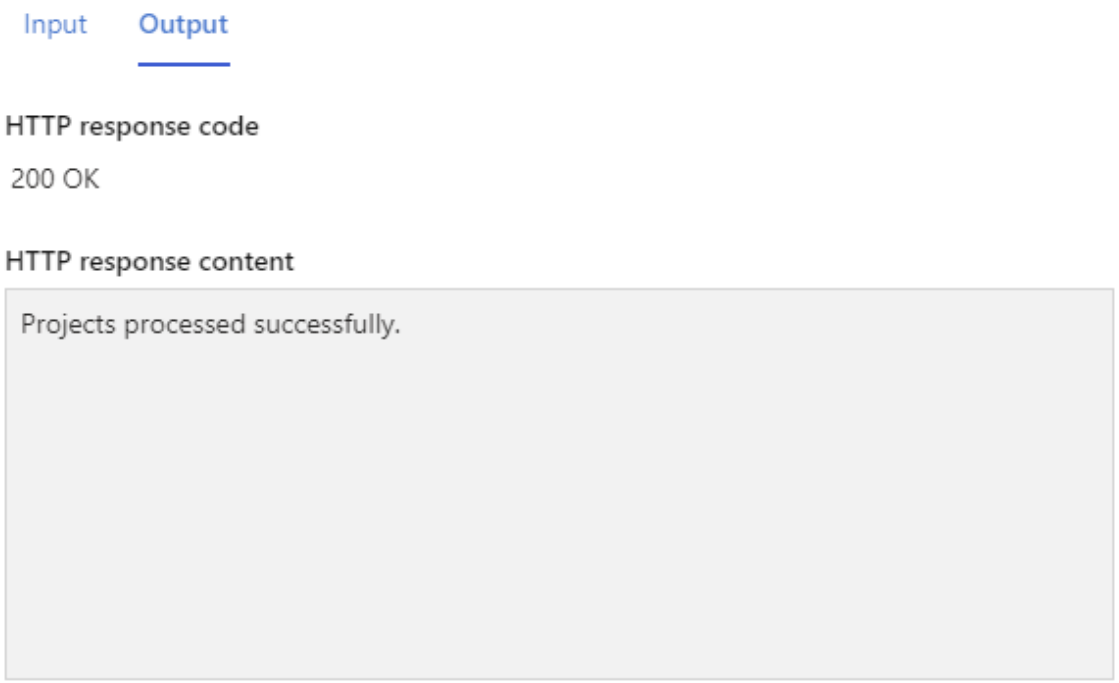
Seuraavaksi käydään läpi, kuinka onnistunut suoritus yhdellä tapahtumalla ja yhdellä aikamerkinnällä testattiin manuaalisesti. Testaus aloitettiin lisäämällä uusi tapahtuma ja aikamerkintä Redmine-projektinhallintaohjelmiston testi-asiakasprojektiin.

Kuvassa 23 nähdään testiasiakasprojektin tapahtumiin luodut aikamerkinnät huhtikuulle ja maaliskuulle. Laskutuslisäosan tulisi sisällyttää laskutukseen vain huhtikuulle tehty aikamerkintä, sillä laskutuslisäosan tulisi ottaa laskutukseen mukaan vain ne aikamerkinnät, jotka on tehty laskutettavan kuukauden aikana.



Kuva 23. Redmine-projektinhallintaohjelmiston testiasiakasprojektin tapahtumiin tehdyt aikamerkinnät toiminnallisuuksien manuaalista testausta varten.

Laskutuslisäosa käynnistettiin lähettämällä HTTP POST -pyyntö laskutuslisäosan Azure-funktioon Code + Test -paneelista. Kuvassa 24 nähdään pyynnön tuottama onnistunut HTTP-vastauskoodi ja HTTP-vastauksen sisältö. Näiden perusteella voidaan päätellä, että laskutuslisäosa onnistui käsittelemään asiakasprojektit onnistuneesti [61].



Kuva 24. HTTP POST -pyynnön tuottama HTTP-vastauskoodi ja HTTP-vastauksen sisältö Azure-portaalin Code + Test -paneelissa.

Talenom-tilitoimistopalvelun testiympäristöstä tarkastettiin vielä, että myyntilasku oli saapunut Talenom-tilitoimistopalvelulle onnistuneesti ja sen tiedot olivat oikein. Kuvassa 25 nähdään Talenom-tilitoimistopalvelun tuottama myyntilasku laskutuslisäosan avulla. Myyntilaskua tarkastelemalla huomataan, että sen laskuriville otettiin Redmine-projektinhallintaohjelmiston testiasiakasprojektin huhtikuulle tehty aikamerkintä ja sen tapahtuma. Laskurivin tuotekohdasta nähdään laskutettavan projektin nimi sekä tapahtuman nimi, johon aikamerkintä oli tehty.

LUONNOS / LASKU

Laskun päivämäärä 30.4.2024	Laskun numero 0
Asiakasnumero 30297120	Asiakkaan ALV-tunnus FI12345678
Eräpäivä 30.5.2024	Y-tunnus
Tilauspäivä	Viitenumero 0
Toimituspäivä	Maksuehto 30 pv netto
Huomautusaika 8 pv	Viivästyskorko % 11,00
Kassa-ale % 0,00	Kassa-ale päivä
Viitteenne 111	
Viitteemme 222	
Tilausnumero 123	

Laskutusosoite:
Asiakkaan nimi
Osoitetie 1 D 36
00980 Helsinki
FINLAND

Toimitusosoite:

Tuotenumero	Tuote	Määrä	Yks.	á-hinta	Ale %	Veroton	Alv %	Verollinen
548	Projekti testaukseen Testi	2	h	120,00	0,00	240,00	24,00	297,60

	Veroton	Vero	Yhteensä
Arvonlisävero 0 %	0,00	0,00	**0,00**
Arvonlisävero 10 %	0,00	0,00	**0,00**
Arvonlisävero 14 %	0,00	0,00	**0,00**
Arvonlisävero 24 %	240,00	57,60	**297,60**
Lasku yhteensä EUR	**240,00**	**57,60**	**297,60**

Tässä laskun viesti.

Kuva 25. Talenom-tilitoimistopalvelun tuottama myyntilasku yhdellä tapahtumalla ja yhdellä aikamerkinnällä.

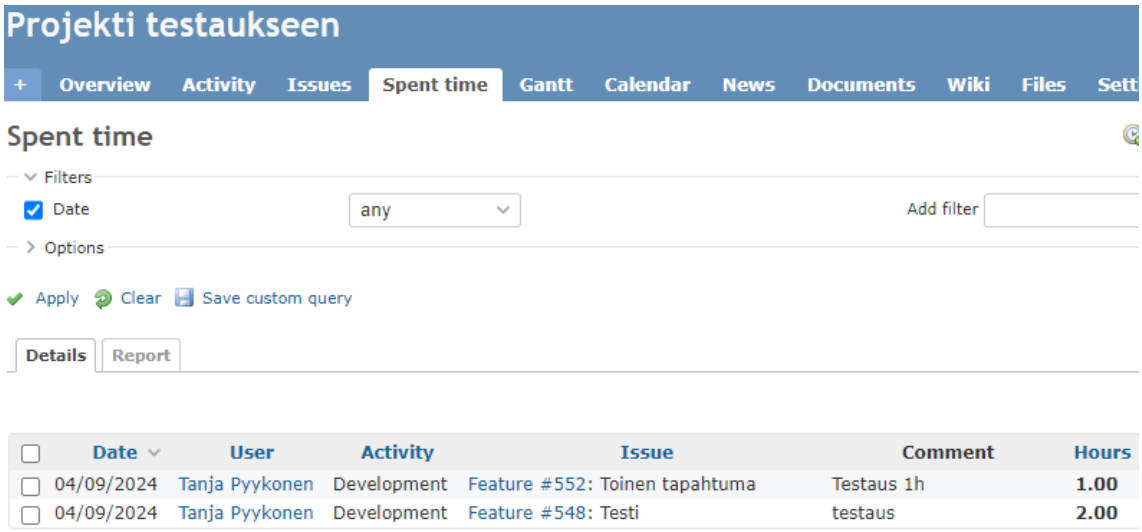
Kun laskutuslisäosan toiminnallisuudet oli saatu testattua yksikkötestien ja manuaalisten testien avulla, siirryttiin laskutuslisäosan automaattisen ja ajastetun suorituksen testaukseen. Testauksessa keskityttiin Power Automate -työnkulun testaukseen.

6.2 Automatisoinnin testaus

Laskutuslisäosan automatisoinnin ja ajastetun suorituksen testauksen tavoitteena oli testata Power Automate -työnkulun logiikkaa. Automatisoinnin testauksessa ei keskitytty enää laskutuslisäosan toiminnallisuuksien testaukseen, vaan sen automatisoinnin ja ajastuksen testaukseen erilaisilla syötteillä.

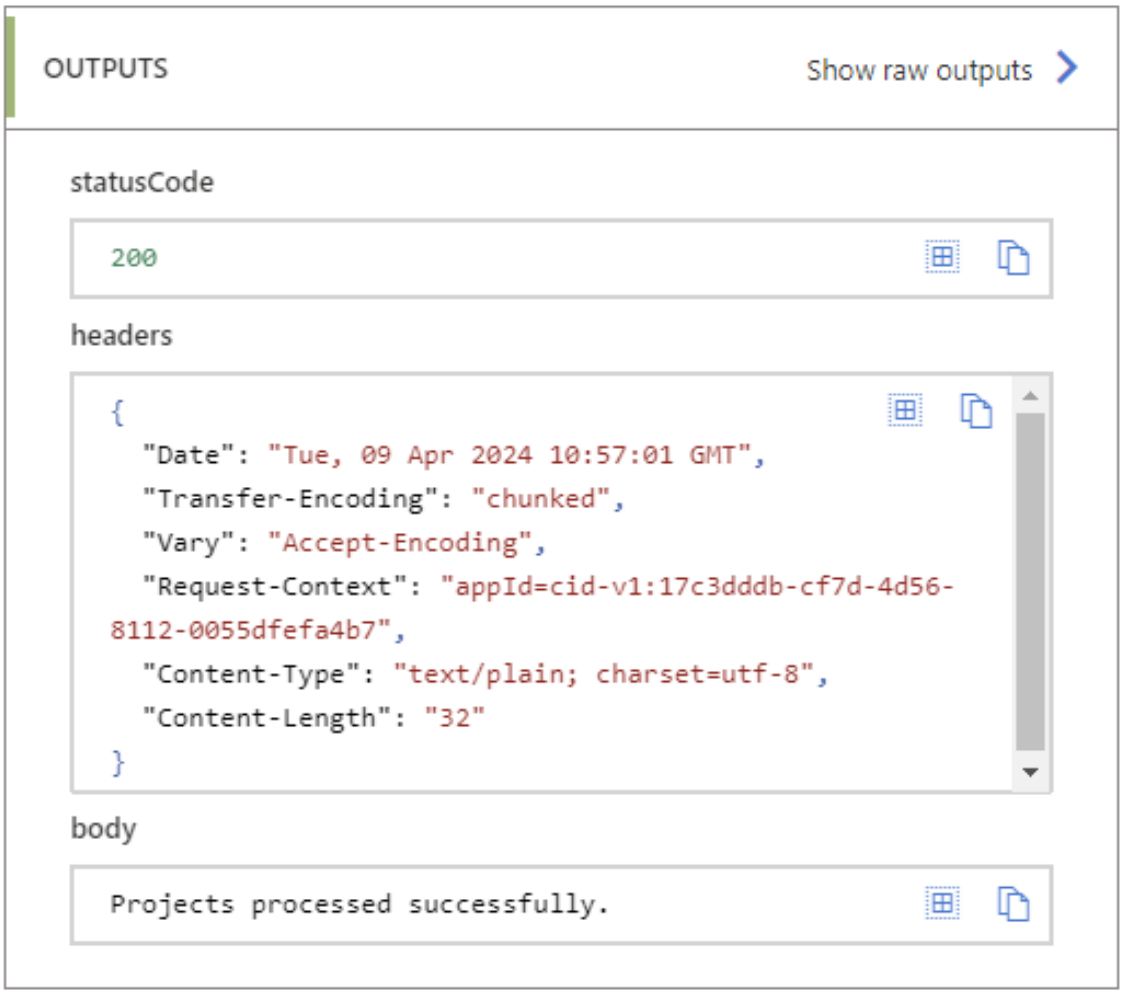
Seuraavaksi tarkastellaan, kuinka yksi laskutuslisäosan automatisoinnin manuaalisista testeistä testattiin kahdella tapahtumalla. Tämän testitapauksen tavoitteena oli varmistaa, että Power Automate -työnkulku ja sen askeleet toimivat oletetulla tavalla. Power Automate -työnkulun tulisi alustaa muuttujat ja lähettää HTTP POST -pyyntö Azure-funktioon, joka käynnistää laskutuslisäosan.

Automatisoinnin manuaalinen testaus aloitettiin lisäämällä tietoja myyntilaskuille. Kuva 26 sisältää Redmine-projektinhallintaohjelmiston testiprojektin tapahtumiin tehdyt aikamerkinnät. Testiprojekti sisältää kaksi aikamerkintää kahdelle tapahtumalle, yksi aikamerkintä kullekin tapahtumalle. Molemmat aikamerkinnät ovat tehty laskutettavan kuukauden aikana ja laskutuslisäosan tulisi ottaa nämä huomioon.



Kuva 26. Redmine-projektinhallintaohjelmiston testiasiakasprojektin tapahtumiin tehdyt aikamerkinnät automatisoinnin manuaalista testausta varten.

Automaattinen ja ajastettu Power Automate -työnkulku ajettiin tämän jälkeen ja kuvasta 27 on nähtävissä työnkulun HTTP POST -pyynnön tuottama vastaus. Vastauksesta nähdään pyynnön HTTP-vastauskoodi (statusCode), HTTP-otsikot (headers) ja HTTP-runko (body), jotka osoittavat, että laskutettavat asiakasprojektit käsiteltiin onnistuneesti [61].



Kuva 27. Laskutuslisäosan Power Automate -työnkulun HTTP POST -pyynnön tuottama vastaus.

Lopuksi Talenom-tilitoimistopalvelun testiympäristöstä tarkastettiin, että myyntilasku oli saapunut Talenom-tilitoimistopalvelulle onnistuneesti. Kuvassa 28 nähdään Talenom-tilitoimistopalvelun tuottama myyntilasku, josta nähdään Redmine-projektinhallintaohjelmiston testiasiakasprojektiin lisätyt kaksi aikamerkintää kahdelle tapahtumalle, joista kumpikin on omana laskurivinään.

LUONNOS / LASKU

Laskun päivämäärä 30.4.2024	Laskun numero 0
Asiakasnumero 30297120	Asiakkaan ALV-tunnus FI12345678
Eräpäivä 30.5.2024	Y-tunnus
Tilauspäivä	Viitenumero 0
Toimituspäivä	Maksuehto 30 pv netto
Huomautusaika 8 pv	Viivästyskorko % 11,00
Kassa-ale % 0,00	Kassa-ale päivä
Viitteenne 111	
Viitteemme 222	
Tilausnumero 123	

Laskutusosoite:
Asiakkaan nimi
Osoitetie 1 D 36
00980 Helsinki
FINLAND

Toimitusosoite:

Tuotenumero	Tuote	Määrä	Yks.	á-hinta	Ale %	Veroton	Alv %	Verollinen
552	Projekti testaukseen Toinen tapahtuma	1	h	120,00	0,00	120,00	24,00	148,80
548	Projekti testaukseen Testi	2	h	120,00	0,00	240,00	24,00	297,60

	Veroton	Vero	**Yhteensä**
Arvonlisävero 0 %	0,00	0,00	**0,00**
Arvonlisävero 10 %	0,00	0,00	**0,00**
Arvonlisävero 14 %	0,00	0,00	**0,00**
Arvonlisävero 24 %	360,00	86,40	**446,40**
Lasku yhteensä EUR	**360,00**	**86,40**	**446,40**

Tässä laskun viesti.

Kuva 28. Talenom-tilitoimistopalvelun tuottama myyntilasku kahdella tapahtumalla ja kahdella aikamerkinnällä.

Power Automate -työnkulun testauksen jälkeen laskutuslisäosan toiminnallisuuksien ja sen automatisoinnin testausvaihe saatiin päätökseen. Tämän jälkeen siirryttiin laskutuslisäosan ja sen automatisoinnin kehitysprosessin viimeiseen vaiheeseen, jossa laskutuslisäosa integroitiin osaksi Systencess Oy:n laskutusprosessia.

7 Käyttöönotto

Laskutuslisäosan käyttöönotto osaksi Systencess Oy:n nykyistä laskutusprosessia alkoi konfiguroimalla yrityksen ympäristöt laskutuslisäosaa varten.

Systencess Oy:n Redmine-ympäristöön lisättiin laskutuslisäosan tarvitsemat mukautetut kentät. Tämän jälkeen automattisen laskutuksen tila laitettiin päälle yrityksen asiakasprojekteille, jotka haluttiin laskuttaa automaattisesti laskutuslisäosan avulla. Myös Power Automate -työnkulussa käytetyt testiympäristöt vaihdettiin yrityksen käytössä oleviin ympäristöihin.

Käyttöönotto testattiin käynnistämällä Power Automate -työnkulku. Laskutuslisäosa generoi myyntilaskut yrityksen Redmine-ympäristössä olevista asiakasprojekteista ja niiden tiedoista, ja lähetti myyntilaskut yrityksen Talenom-tilitoimistopalveluun. Tässä vaiheessa ilmeni ongelma, jota kehitystyön testauksen aikana ei huomattu. Tarkastelemalla generoituja myyntilaskuja Talenom-tilitoimistopalvelusta huomattiin, että skandinaaviset merkit eivät näkyneet oikein myyntilaskuilla. Skandinaaviset merkit myyntilaskuilla oli korvattu kysymysmerkeillä, ja ongelman epäiltiin liittyvän merkistökoodaukseen.

Ennen HTTP POST -pyynnön lähetystä Talenom-tilitoimistopalvelulle skandinaaviset merkit näkyivät oikein, mutta itse vastauksessa ne olivat korvaantuneet kysymysmerkeillä. Ongelma saatiin korjattua lisäämällä UTF-8-merkistökoodaus (Unicode Transformation Format) HTTP POST -pyynnön sisältöön esimerkkikoodin 13 mukaisella tavalla. Lisäämällä UTF-8-merkistökoodaus StringEntity-oliolle, joka edustaa HTTP POST -pyynnön sisältöä merkkijonona, ongelma korjaantui ja myyntilaskujen skandinaaviset merkit näkyivät oikein myyntilaskuilla. [62; 63.]

```
HttpEntity requestEntity = new StringEntity(soapRequestBody,
     StandardCharsets.UTF_8);
```

Esimerkkikoodi 13. Laskutuslisäosan Java-ohjelmakoodi StringEntity-olion luomisesta UTF-8-merkistökoodauksella.

Jotta skandinaavisten merkkien ongelma olisi huomattu aikaisemmin, olisi laskutuslisäosan testauksessa tullut kiinnittää enemmän huomiota merkistökoodaukseen ja testata myyntilaskuja erilaisilla skandinaavisilla merkeillä. Ongelma toimi kuitenkin opettavaisena tilanteena ja hyvänä muistuttujana merkistökoodausten tärkeydestä.

Merkistökoodauksen ongelman selvittämisen jälkeen laskutuslisäosa suoritettiin uudelleen ja myyntilaskut generoituivat oikein. Myyntilaskut poistettiin vielä tässä vaiheessa manuaalisesti Systencess Oy:n Talenom-tilitoimistopalvelusta ja laskutuslisäosa suoritettiin uudelleen kuukauden viimeisenä päivänä.

Systencess Oy:n asiakkaat laskutetaan kerran kuukaudessa ja tätä insinöörityötä kirjoittaessa laskutuslisäosalla ehdittiin laskuttaa asiakkaiden projektit vain kerran automaattisesti ja ajastetusti. Laskutuslisäosa generoi ja lähetti myyntilaskut Talenom-tilitoimistopalvelulle moitteettomasti. Tämän jälkeen laskuttaja tarkisti myyntilaskujen tiedot ja hyväksyi ne, minkä jälkeen Talenom-tilitoimistopalvelu lähetti myyntilaskut asiakkaille.

8 Yhteenveto

Tämän insinöörityön tavoitteena oli kehittää Systencess Oy:n toiveiden mukaan laskutuslisäosa ja sen automatisoitu suoritus yrityksen Redmine-projektinhallintaohjelmiston ja Talenom-tilitoimistopalvelun välille. Laskutuslisäosan ja sen automatisoinnin tavoitteena oli automatisoida yrityksen asiakkaiden laskutusprosessia entisestään ja vähentää laskutuksen manuaalista työtä.

Laskutuslisäosan tulisi generoida myyntilaskut automaattisesti Redmine-projektinhallintaohjelmiston asiakasprojektien tiedoista ja laskuttaa asiakkaat automaattisesti ja ajastetusti kerran kuukaudessa. Laskutuslisäosa oli suunnattu yrityksen omiin tarpeisiin, ja se oli osana yrityksen sisäistä kehitystä. Laskutuslisäosa tarjosi hyödyn yrityksen laskutusprosessista vastaavalle henkilölle.

Laskutuslisäosan kehitystyö alkoi suunnittelemalla laskutuslisäosan ja sen automatisoinnin vaatimukset. Tämän jälkeen esiteltiin kehitystyössä käytetyt teknologiat ja työkalut, jonka jälkeen siirryttiin laskutuslisäosan toiminnallisuuksien toteutukseen. Toiminnallisuudet ohjelmointiin Java-ohjelmointikielellä, hyödyntäen Apache Maven -projektinhallintakehystä. Näiden toiminnallisuuksien avulla myyntilaskut generoitiin automaattisesti Redmine-projektinhallintaohjelmiston

asiakasprojekteista, jonka jälkeen myyntilaskut validoitiin ja lähetettiin Talenom-tilitoimistopalvelulle.

Laskutuslisäosan automaattinen ja ajastettu suoritus toteutettiin hyödyntämällä Microsoft Azure -pilvipalvelualustan Azure-funktiota ja Microsoft Power Automate -pilviautomaatioympäristön työnkulkua. Toteutusten jälkeen laskutuslisäosan toiminnallisuuksia ja sen automatisointia testattiin yksikkötestien ja manuaalisten testien avulla. Lopuksi laskutuslisäosa integroitiin osaksi yrityksen laskutusprosessia.

Laskutuslisäosan kehitystyön aikana otettiin huomioon lisäkehityksen ja laajennuksen mahdollisuus yrityksen toiveiden mukaan. Laskutuslisäosassa käytettävien ympäristöjen tunnistetiedot määriteltiin Power Automate -työnkulussa. Luomalla toinen Power Automate -työnkulku voidaan automaattinen laskutus tehdä toisesta Redmine-projektinhallintaohjelmiston ympäristöstä toiseen Talenom-tilitoimistopalvelun ympäristöön.

Haasteita laskutuslisäosan ja sen automatisoinnin kehityksessä tuotti sen suuri koko ja useat ympäristöt, jotka aiheuttivat hankaluuksia kokonaisuuden hallinnassa. Myös jälkeenpäin analysoidessa ja pohdittaessa laskutuslisäosan toteutusta, testaukseen olisi ollut hyvä kiinnittää enemmän huomiota. Vaikka laskutuslisäosaa ja sen automatisointia testattiin useilla testitapauksilla ja erilaisissa olosuhteissa, törmättiin laskutuslisäosan käyttöönotossa merkistökoodausongelmaan. Ongelma saatiin kuitenkin korjattua, mutta ongelma olisi ollut mahdollista havaita ennen käyttöönottoa laajemmilla testitapauksilla.

Laskutuslisäosa saatiin tavoitteiden mukaan onnistuneesti suunniteltua, kehitettyä, testattua ja otettua osaksi yrityksen laskutusprosessia. Yrityksen laskuttajan ei tarvinnut manuaalisesti luoda myyntilaskuja, vaan laskutuslisäosa generoi myyntilaskut automaattisesti ja lähetti ne Talenom-tilitoimistopalvelulle. Tämän jälkeen laskuttaja tarkasti myyntilaskut Talenom-tilitoimistopalvelun myyntilaskujärjestelmästä ja hyväksyi ne, jonka jälkeen ne lähtivät asiakkaille.

Lähteet

1 Systencess Oy kotisivu. Verkkoaineisto. Systencess Oy. <https://www.systencess.fi>. Luettu 16.5.2023.

2 Yritys Systencess Oy. Verkkoaineisto. Kauppalehti. <https://www.kauppalehti.fi/yritykset/yritys/systencess+oy/21259011>. Luettu 16.5.2023.

3 Redmine Toggl Plugin. Verkkoaineisto. Redmine. <https://www.redmine.org/plugins/redmine_toggl>. Luettu 17.5.2023.

4 Redmine Time Tracking. Verkkoaineisto. Redmine. <https://www.redmine.org/projects/redmine/wiki/redminetimetracking>. Luettu 19.5.2023.

5 Toggl. 2021. Toggl Track Product Demo. Video. YouTube. <https://www.youtube.com/watch?v=e_SKyiGgilg>. Katsottu 23.5.2023.

6 Biemer P. Paul; Lyberg E. Lars. 2003. Introduction to Survey Quality. Hoboken: John Wiley & Sons.

7 Varmista, että nämä asiat on ilmoitettu laskussa. Verkkoaineisto. Allianz Trade. <https://www.allianz-trade.com/fi_FI/luottovakuutuksen-tuntemus/e-kirja-opas-tehokkaaseen-laskutukseen/luku-1-varmista-etta-nama-asiat-on-ilmoitettu-laskussa.html>. Luettu 23.5.2023.

8 Alhir, Sinan Si. 2003. Learning UML. E-kirja. O'Reilly Media, Inc.

9 What is UML. Verkkoaineisto. UML. <https://www.uml.org/what-is-uml.htm>. Päivitetty 1.7.2005. Luettu 19.2.2024.

10 Kulak, Daryl & Guiney Eamonn. 2003. Use Cases: Requirements in Context, Second Edition. E-kirja. Addison-Wesley Professional.

11 What is an application programming interface (API). Verkkoaineisto. IBM. <https://www.ibm.com/topics/api>. Luettu 19.2.2024.

12 Awati, Rahul. Uniform Resource Identifier (URI). Verkkoaineisto. TechTarget. <https://www.techtarget.com/whatis/definition/URI-Uniform-Resource-Identifier>. Päivitetty 1.10.2021. Luettu 19.2.2024.

13 Why and when to use API keys. Verkkoaineisto. Google Cloud. <https://cloud.google.com/endpoints/docs/openapi/when-why-api-key>. Päivitetty 14.2.2024. Luettu 19.2.2024.

14 What is XML. Verkkoaineisto. AWS. <https://aws.amazon.com/what-is/xml/>. Luettu 19.2.2024.

15 Van der Vlist, Eric. 2002. XML Schema. E-kirja. O'Reilly Media, Inc.

16 Burd, Barry. 2021. Beginning Programming with Java For Dummies, 6th Edition. E-kirja. For Dummies.

17 JDK 11 Documentation. Verkkoaineisto. Oracle. <https://docs.oracle.com/en/java/javase/11/>. Luettu 2.3.2024.

18 2023 Developer Survey. Verkkoaineisto. Stack Overflow. <https://survey.stackoverflow.co/2023/>. Luettu 12.3.2024.

19 The State of Developer Ecosystem 2023. Verkkoaineisto. JetBrains. <https://www.jetbrains.com/lp/devecosystem-2023/>. Luettu 2.3.2024.

20 Java Platform, Standard Edition & Java Development Kit Version 11 API Specification. Verkkoaineisto. Oracle. <https://docs.oracle.com/en/java/javase/11/docs/api/index.html>. Luettu 5.3.2024.

21 Maven Central Repository. Verkkoaineisto. Sonatype. <https://central.sonatype.com/>. Luettu 5.3.2024.

22 Bloch, Joshua. 2017. Effective Java, 3rd Edition. E-kirja. Addison-Wesley Professional.

23 Introducing JSON. Verkkoaineisto. JSON.org. <https://www.json.org/json-en.html>. Luettu 5.3.2024.

24 HTTP. Verkkoaineisto. MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/HTTP>. Luettu 5.3.2024.

25 Andersson, Jonah Carrio. 2023. Learning Microsoft Azure. E-kirja. O’Reilly Media, Inc.

26 Introduction to Archetypes. Verkkoaineisto. Apache Maven Project. <https://maven.apache.org/guides/introduction/introduction-to-archetypes.html>. Luettu 6.3.2024.

27 Introduction to the POM. Verkkoaineisto. Apache Maven Project. <https://maven.apache.org/guides/introduction/introduction-to-the-pom.html>. Luettu 6.3.2024.

28 Varanasi, Balaji. 2019. Introducing Maven: A Build Tool for Today's Java Developers. E-kirja. Apress.

29 Maven Central Repository: azure-functions-java-library. Verkkoaineisto. Sonatype. <https://central.sonatype.com/artifact/com.microsoft.azure.functions/azure-functions-java-library>. Luettu 12.3.2023.

30 Redmine Wiki. 2023. Verkkoaineisto. Redmine. <https://www.redmine.org>. Luettu 12.3.2023.

31 Giretti, Anthony. 2023. Coding Clean, Reliable, and Safe REST APIs with ASP.NET Core 8: Develop Robust Minimal APIs with .NET 8. E-kirja. Apress.

32 Redmine API. Verkkoaineisto. Redmine. <https://www.redmine.org/projects/redmine/wiki/rest_api>. Luettu 14.3.2024.

33 Reis, Joe & Housley, Matt. 2022. Fundamentals of Data Engineering. E-kirja. O'Reilly Media, Inc.

34 Talenom kotisivu. Verkkoaineisto. Talenom. <https://talenom.com/fi-fi/>. Luettu 15.3.2024.

35 Web service myyntilaskurajapinnan tekninen kuvaus. Verkkoaineisto. Talenom Integraatioportaali. <https://integration.talenom.fi/developers/web-service-myyntilaskurajapinta/>. Luettu 15.3.2024.

36 Introduction to Web Service Technologies. Verkkoaineisto. Oracle. <https://docs.oracle.com/cd/E13224_01/wlw/docs103/guide/webservices/conBasicWebServiceTechnologies.html>. Luettu 15.3.2024.

37 Service NewInvoice. Verkkoaineisto. Talenom verkkopalvelu. <https://verkkopalvelu3.talenom.fi:4445/InvoiceService.asmx?op=NewInvoice>. Luettu 15.3.2024.

38 Azure Functions HTTP trigger. Verkkoaineisto. Microsoft Learn. <https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-http-webhook-trigger?tabs=python-v2%2Cisolated-process%2Cnodejs-v4%2Cfunctionsv2&pivots=programming-language-java>. Luettu 20.3.2024.

39 Quickstart: Create a Java function in Azure using Visual Studio Code. Verkkoaineisto. Microsoft Learn. <https://learn.microsoft.com/en-us/azure/azure-functions/create-first-function-vs-code-java>. Luettu 20.3.2024.

40 Microsoft Azure Portal. Verkkoaineisto. Microsoft Azure. <https://azure.microsoft.com/en-us/get-started/azure-portal>. Luettu 21.3.2024.

41 Create your first function in the Azure portal. Verkkoaineisto. Microsoft Learn. <https://learn.microsoft.com/en-us/azure/azure-functions/functions-create-function-app-portal?pivots=programming-language-java>. Luettu 21.3.2024.

42 Sawhney, Rahul & Chanumolu, Kalyan. 2023. Beginning Azure Functions: Building Scalable and Serverless Apps. E-kirja. Apress.

43 Develop Azure Functions by using Visual Studio Code. Verkkoaineisto. Microsoft Learn. <https://learn.microsoft.com/en-us/azure/azure-functions/functions-develop-vs-code?tabs=node-v3%2Cpython-v2%2Cisolated-process&pivots=programming-language-java>. Luettu 21.3.2024.

44 Microsoft Power Automate – Process Automation Platform. Verkkoaineisto. Microsoft. <https://www.microsoft.com/en-us/power-platform/products/power-automate>. Luettu 22.3.2024.

45 Guilmette, Aaron. 2022. Workflow Automation with Microsoft Power Automate – Second Edition. E-kirja. Packt Publishing.

46 Marshall, Donis & Bruno, John. 2009. Solid Code: Optimizing the Software Development Life Cycle. E-kirja. Microsoft Press.

47 Gerrit Code Review for Git. Verkkoaineisto. Gerrit. <https://gerrit-documentation.storage.googleapis.com/Documentation/3.7.2/index.html>. Luettu 25.3.2024.

48 Mohan, Gayathri. 2022. Full Stack Testing. E-kirja. O'Reilly Media, Inc.

49 Installing Redmine. Verkkoaineisto. Redmine. <https://www.redmine.org/projects/redmine/wiki/redmineinstall>. Luettu 27.3.2024

50 Bitnami package for Redmine. Verkkoaineisto. Bitnami. <https://bitnami.com/stack/redmine/virtual-machine>. Luettu 27.3.2024.

51 Java API for XML Processing (JAXP) Tutorial. Verkkoaineisto. Oracle. <https://www.oracle.com/java/technologies/jaxp-introduction.html>. Luettu 29.3.2024.

52 Introduction to JAXP. Verkkoaineisto. Oracle. <https://docs.oracle.com/javase/tutorial/jaxp/intro/dom.html>. Luettu 29.3.2024.

53 Javax.xml.validation (Java SE 11 & JDK 11). Verkkoaineisto. Oracle. <https://docs.oracle.com/en/java/javase/11/docs/api/java.xml/javax/xml/validation/package-summary.html>. Luettu 1.4.2024.

54 W3C – Making the Web work. Verkkoaineisto. W3C. <https://www.w3.org/>. Luettu 1.4.2024.

55 QName (Java SE 11 & JDK 11). Verkkoaineisto. Oracle. <https://docs.oracle.com/en/java/javase/11/docs/api/java.xml/javax/xml/namespace/QName.html>. Luettu 2.4.2024.

56 Azure Functions hosting options. Verkkoaineisto. Microsoft Learn. <https://learn.microsoft.com/en-us/azure/azure-functions/functions-scale>. Luettu 3.4.2024.

57 Quickstart: Create a Java function in Azure from the command line. Verkkoaineisto. Microsoft Learn. <https://learn.microsoft.com/en-us/azure/azure-functions/create-first-function-cli-java?tabs=windows%2Cbash%2Cazure-cli%2Cbrowser>. Luettu 3.4.2024.

58 Run flows on a schedule. Verkkoaineisto. Microsoft Learn. <https://learn.microsoft.com/en-us/power-automate/run-scheduled-tasks>. Luettu 3.4.2024.

59 Reference guide to workflow expression functions in Azure Logic Apps and Power Automate. Verkkoaineisto. Microsoft Learn. <https://learn.microsoft.com/en-us/azure/logic-apps/workflow-definition-language-functions-reference>. Luettu 5.4.2024.

60 JUnit 5 User Guide. Verkkoaineisto. JUnit. <https://junit.org/junit5/docs/current/user-guide/#overview>. Luettu 8.4.2024.

61 HTTP response status codes. Verkkoaineisto. MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>. Luettu 9.4.2024.

62 HTML Unicode (UTF-8) Reference. Verkkoaineisto. W3Schools. <https://www.w3schools.com/charsets/ref_html_utf8.asp>. Luettu 11.4.2024.

63 StringEntity. Verkkoaineisto. Apache HttpCore 4.4.16 API. <https://hc.apache.org/httpcomponents-core-4.4.x/current/httpcore/apidocs/org/apache/http/entity/StringEntity.html>. Luettu 11.4.2024.

Myyntilaskujen elementit

Taulukko 1. Elementit, joita laskutuslisäosa hyödynsi myyntilaskuille [35].

Elementti	**Pakollinen**	**Kommentti**
INVOICE	Kyllä	Juurielementti. Vaatii tyhjän nimiavaruuden määrittelyn.
CLIENT_ID	Kyllä	Systencess Oy:n asiakasnumero
INVOICE_DATE	Kyllä	Myyntilaskun päivämäärä
DUE_DATE	Kyllä	Myyntilaskun eräpäivä
ORDER_NUMBER	Ei	Tilausnumero
OUR_REFERENCE	Ei	Viitteemme
YOUR_REFERENCE	Ei	Viitteenne
DNET	Kyllä	Maksuehto
REMARKTIME	Kyllä	Huomautusaika
INVOICE_MESSAGE	Ei	Laskutuskohtainen viesti
SHIPMODE	Kyllä	Lähetystapa. Laskutuslisäosa käyttää lähetystapaa 0 (= Tulostus), jonka vuoksi lasku on käytävä hyväksymässä manuaalisesti Talenomissa.
PAYER	Kyllä	Asiakkaan tiedot
NUMBER	Kyllä	Asiakkaan asiakasnumero
NAME	Kyllä	Asiakkaan nimi
ACCOUNTS_RECEIVABLE	Kyllä	Asiakkaan saatavatilinumero
INVOICE_ADDRESS	Kyllä	Laskutusosoite
STREET_ADDRESS	Kyllä	Laskutusosoite: Katuosoite
POSTAL_CODE	Kyllä	Laskutusosoite: Postinumero
POST_OFFICE	Kyllä	Laskutusosoite: Postitoimipaikka
COUNTRY	Kyllä	Laskutusmaa. Esimerkiksi FI/SE/jne.
EINVOICEID	Ei	Asiakkaan verkkolaskutunnus

INVOICE_LANGU-AGE	Kyllä	Laskupohjan kieli. Mahdolliset arvot: fi-FI, sv-SE, en-GB ja de-DE.
ROWS		Myyntilaskun rivit
ROW		Myyntilaskun rivi. Vähintään yksi rivi per myyntilasku.
PRODUCT_NUM-BER	Kyllä	Tuotenumero
PRODUCT_NAME	Kyllä	Tuotteen nimi
PRODUCT_INFO	Kyllä	Selite. Elementti on pakollinen, mutta se voi olla tyhjä, kun ROW_TYPE on nolla.
QUANTITY	Kyllä	Määrä
UNIT	Ei	Yksikkö. Esimerkiksi kpl tai h.
PRICEPERUNIT	Kyllä	Yksikköhinta
DISCOUNT_PER-CENTAGE	Kyllä	Alennusprosentti
VAT_RATE	Kyllä	Arvonlisäveroprosentti
ROW_TYPE	Kyllä	Rivin tyyppi. Mahdolliset arvot: 0 (Tuote), 1 (Selite).
SALES_AC-COUNT_NUMBER	Kyllä	Myyntitilinumero