TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Getter Ree 183027IABM

ARVETE SALVESTAMISE JA PÕHIANDMETE LUGEMISE TEENUSE ANALÜÜS JA ARENDUS ETTEVÕTTE SECUREBADGER OÜ NÄITEL

Magistritöö

Juhendaja: Mart Roost, MSc

Kaasjuhendaja: Karl Märka, MSc

TALLINN 2020

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Getter Ree

15.05.2020

Annotatsioon

Käesoleva magistritöö eesmärk on automatiseerida Securebadger OÜ arvete salvestamist ja põhiandmete lugemist. Töös kirjeldatakse kasutatud metoodikaid ning antakse ülevaade töö käigus kasutusele võetud tööriistadest, kaardistatakse olemasolev olukord, nõuded ja TO BE lahendus, kirjeldatakse arendusprotsessi, realisatsiooni ja testimisest ning töö lõpus toimub tulemuste analüüs.

Järjest suureneva kliendibaasi tõttu tekkis manuaalsel arvete salvestamisel ja arvelt põhiandmete kättesaamisel mitmeid probleeme. Olulisemateks probleemideks on ajakulu ja tööjõukulu kasv ning risk inimvigade tekkimiseks.

Töö tulemusena luuakse lahendus, mis muudab arvete salvestamise ja põhiandmete lugemise automaatseks protsessiks. Lahenduse realiseerimiseks kasutatakse Amazon Web Service platvormi.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 49 leheküljel, 10 peatükki, 16 joonist ja 2 tabelit.

Abstract

Analysis and implementation of an invoice storage and parsing service in Securebadger OÜ

Securebadger OÜ offers rental income factoring. The factoring service includes a free invoice management. This means that Securebadger pays all the bills related to the rental object directly to the service providers and presents the aggregated invoice to the tenant. The data required for the aggregated invoice is saved manually. Due to the growing customer base, there were several problems with manually saving invoices and obtaining basic data from the invoice. The most important problems are the increase in time and labor costs and the risk of human error.

As a result of the work, a solution is created that makes saving invoices and obtaining of basic data an automatic process. The Amazon Web Service platform is used to implement the solution.

The aim of this master's thesis is to automate the saving of invoices and obtaining of data from invoices. The work describes the methodologies used and provides an overview of used tools, maps the existing situation, requirements and TO BE process, describes the development process, gives an overview of implementation and testing, and at the end of the work the results are analyzed.

The dissertation is written in Estonian and contains 49 pages of text, 10 chapters, 16 figures and 2 tables.

Lühendite ja mõistete sõnastik

AWS	*Amazon Web Service* Pilveteenuse platvorm
AS IS	Olemasolev olukord
TO BE	Protsessi tulevikuseisund
BPMN	*Business Process Model & Notation* Äriprotsesside modelleerimiskeel
PDF	*Portable Document Format* Failiformaat
JSON	*JavaScript Object Notation* Andmevahetusvorming
MIME	*Multipurpose Internet Mail Extensions* Mitmeotstarbeline Interneti-posti laiendus
triger	Funktsiooni käivitamine
SNS	*Amazon Simple Notification Service* Teavituste teenus
SQS	*Amazon Simple Queue Service* Sõnumijärjekorra teenus
SES	*Amazon Simple Email Service* E-posti teenus
S3	*Amazon Simple Storage Service* Salvestusruum
Lambda	Funktsioon
Cloudwatch	Amazoni monitoorimise – ja haldamise teenus
NoSQL	*Not only SQL* Mitterelatsiooniline andmebaas

Slack	Töökoha kommunikatsioonivahend sõnumite, tööriistade ja failide jaoks.
API	*Application Programming Interface* Rakendusliides
TDD	*Test-driven development* Testipõhine arendus
OCR	*Optical character recognition* Optiline märgituvastus
Chalice	Pythoni mikroraamistik serverita arenduste tegemiseks

Sisukord

Jooniste nimekiri

Tabelite nimekiri

1 Sissejuhatus

Ettevõte Securebadger OÜ (edaspidi Securebadger) pakub üüritulu faktooringut. See tähendab, et kinnisvara omanik saab tulevase üüritulu kätte täielikult või osaliselt ettemaksuna, kas üürilepingu alguses või jooksvalt vajaduse tekkides. Faktooringuteenuse kasutamiseks sõlmitakse Securebadgeri ja üürileandja vahel leping [1]. Faktooringuteenuse soetamisel on kliendil võimalik kasutada arvete haldus teenust, mis on mõeldud kinnivaraomanikele, kes ei soovi aega kulutada üüripinnaga seotud arvete haldamisele.

Arvete halduse teenuse korral tasub Securebadger kõik üüriobjektiga seonduvad arved otse teenusepakkujatele ning esitavad koondarve üürnikule. See tähendab, et Securebadger maksab kõik üüripinnaga seotud kulutused: korteriühistu poolt esitatud kommunaalarved, teenusepakkujate poolt saadetud elektri- või kindlustusarved ja muud üüripinnaga seotud kulutused. Iga kuu esitatakse üürnikule koondarve, mis sisaldab üüriobjektiga seonduvaid kulutusi.

Antud töös keskendutakse koondarve koostamise jaoks vajalike andmete salvestamisele. Koondarve tarbeks vajalikud andmed salvestatakse manuaalselt. Andmete salvestamise protsessiga on esile kerkinud mitmed probleemid. Securebadgeri kliendibaasi kasvu tõttu ei jõuta enam manuaalseid toiminguid läbi viia. Tekkinud on rohkem teenusepakkujate arveid, mistõttu salvestamise protsess on muutunud ajakulukamaks ning inimvigade tekkimise risk on suurenenud. Seetõttu on kasvanud vajadus muuta arvete salvestamise ja andmete lugemise protsess automaatseks.

Lõputöö eesmärk on automatiseerida arvete salvestamine ja andmete lugemine ning salvestamine. Protsessi automatiseerimine lihtsustab ettevõtte tööd. Lahenduse realiseerimiseks kasutatakse Amazon Web Service platvormi.

1.1 Taust ja probleem

Securebadger pakub üüritulu faktooringuteenust kinnivara omanikele. Faktooringuteenuse kasutamisel on võimalus kinnivara omanikul kasutada tasuta arvete halduse teenust. Arvete haldusteenuse puhul tasub Securebadger kõik üüriobjektiga seonduvad arved otse teenusepakkujatele ja koostab koondarve üürnikule. Selleks, et edastada koondarve üürnikule peab enne arvetelt vajalikud andmed kätte saama. Hetkel saadetakse teenusepakkujate arved e-posti teel ning salvestatakse kohalikule kettale ja seejärel sisestatakse vajalikud andmed manuaalselt Excelisse. Securebadgeri kliendibaas pidevalt kasvab ning enam ei jõuta üüriobjektile kuuluvaid arveid ja andmeid manuaalselt salvestada. Arvete maht on muutunud nii suureks, et andmete salvestamine muutub aina tülikamaks ning suureneb aja- ja tööjõukulu. Mahu suurenemise tõttu kasvab järjest enam risk inimvigade tekkimiseks. Ettevõtte kasvamine muudab ka nõudmisi ja senine tööprotsess ei vasta enam vajadusele. Lisaks sellele ei ole kohalikule kettale andmete salvestamine hea praktika. Töö lihtsustamiseks ja efektiivsemaks muutmiseks on vajalik luua lahendus, mis aitab andmete salvestamise protsessi automatiseerida. Automaatsed protsessid on kuluefektiivsemad, vähendades aja- ja tööjõukulu. Magistritöö probleemi püstitusest tulenevalt on uurimisküsimused järgmised:

1. Kuidas muuta arvelt andmete lugemise ja salvestamise protsess automaatseks?
2. Millised nõuded on olulised tulevase lahenduse loomisel?

1.2 Ülesandepüstitus

Käesoleva lõputöö eesmärk on automatiseerida teenusepakkujate arvetelt andmete välja lugemine ja salvestamine. Eesmärgi saavutamiseks on olulised järgmised tegevused:

- Olemasoleva olukorra kaardistamine;
- Nõuete kogumine ja kaardistamine;
- TO BE protsessi kaardistamine;
- Arendamine ja arendusprotsessi kirjeldamine;
- Realisatsiooni kaardistamine;
- Lahenduse testimine;
- Tulemuste analüüs ja järeldused.

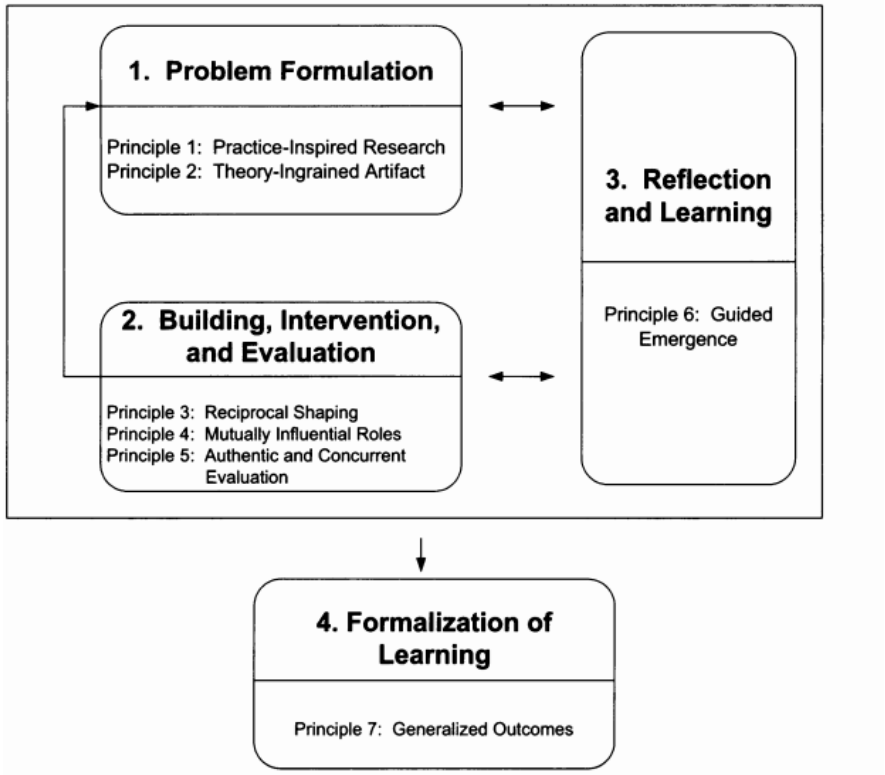
Töö eesmärgi saavutamiseks on etteantud Amazon Web Service pilveteenuste platvorm. Loodava lahenduse arendamiseks kasutatakse programmeerismikeelt Python.

1.3 Töö struktuur

Käesolev töö jaguneb kümneks peatükiks. Esimeses peatükis on kirjeldatud töö taust ja probleemid ning toimub uurimisküsimuste ja -ülesande püsitamine. Teises peatükis kirjeldatakse metoodikat ja Amazon Web Service platvormil kasutatavaid teenuseid. Kolmandas peatükis kaardistatakse hetkeolukord ning tuuakse esile arvete salvestamisega kaasnevad probleemid ning üldpildi saamiseks on koostatud ettevõtte väärtusvahetuse- ja domeenimudel. Neljandas peatükis on kirjeldatakse süsteemi funktsionaalsed ja mittefunktsionaalsed nõuded. Viiendas peatükis on kaardistatud TO BE protsess. Kuuendas peatükis kirjeldatakse arendusprotsessi, seitsmendas on kirjeldatud arenduse tulemusena valminud realisatsioon. Kaheksandas peatükis antakse ülevaade testimisest ning üheksandas peatükis toimub tulemuste analüüs ja järelduste tegemine. Ja viimases peatükis on kokkuvõte.

2 Metoodika

Käesolevas töös kasutatakse eesmärkide saavutamiseks tegevusliku disainiuuringu (inglise k. *Action Design Research*) meetodit (Joonis 1, lk 14). Tegevusliku disainiuuringu meetod kontseptualiseerib uurimisprotsessi, sisaldades lahutamatuid ja põimitud tegevusi IT lahenduse ülesehitamise, organisatsiooni sekkumise ja tulemuste hindamise ühendamiseks kooskõlastatud uurimuse saavutamise eesmärgil [2].



Joonis 1. Tegevusliku disainiuuringu meetodi etapid ja põhimõtted [2].

Tegevusliku disainiuuringu metoodika jaguneb neljaks tsüklipõhiseks tegevusplokiks, mille vahel võib vabalt navigeerida. Esimeses etapis toimub olemasoleva olukorra kaardistamine ja probleemide ning uurimisküsimuste püstitamine. Täpsustatakse metoodikat järgnevate tegevusplokkide läbiviimiseks. Teises tegevusplokis kirjeldatakse lahenduse analüüs ning toimub lahenduse realiseerimine, rakendamine ja hindamine testipõhist agiilset arendusmetoodikat järgides. Kolmas tegevusplokk on läbi põimitud esimese ja teisega ning seisneb jooksvate töötulemite pidevas peegeldamises ning

parendamises. Neljas etapp seisneb tulemuste üldistamises ning lõpliku lahenduse vormistamises.

Olemasoleva olukorra funktsionaalsete- ja mittefunktsionaalsete nõuete kogumiseks kasutatakse struktureerimata intervjuu metoodikat huvigruppide võtmeisikutega. Intervjuu on üks kõige sagedamini kasutatav meetod kvalitatiivsete andmete kogumiseks. Struktureerimata intervjuus on intervjueeritav nii küsimuste kui ka vastuste allikas. Struktureerimata intervjuu eesmärk on saada võimalikult palju teavet laialt määratletud teemal [3].

Kommunaalarvete andmete salvestamise AS IS ja TO BE protsesside modelleerimiseks kasutatakse BPMN notatsiooni. AS IS ja TO BE kaardistamiseks kasutati tööprotsesside modelleerimise vahendit Bizagi. Realiseerimise protsesside olekudiagrammide loomisel kasutati veebipõhist skeemide ja jooniste tegemise tööriista draw.io. Väärtusvahetuse- ja domeenimudeli koostamisel kasutati tarkvaralahenduste modellerimise vahendit Enterprise Architect.

Lahenduse realiseerimisel kasutatakse testipõhist arendust (inglise k *test-driven development*). TDD on tarkvaraarenduspraktika, kus automatiseeritud testid kirjutatakse enne koodi kirjutamist [4]. Arendamisel kasutati programmeerimiskeelt Python. Python on objektorienteeritud kõrgetasemeline programmeerimiskeel integreeritud dünaamilise semantikaga peamiselt veebi ja rakenduste arendamiseks [5].

2.1 Ülevaade tööriistadest

On olemas erinevaid pilveteenuse platvorme, kus antud töö eesmärki on võimalik realiseerida, kuid ülesandepüstituses on ära määratud, et kasutusele tuleb võtta Amazon Web Service platvorm, sest kogu ülejäänud ettevõtte süsteem on juba ülesehitatud AWS platvormil. Seetõttu ei analüüsita ega võrrelda omavahel erinevaid platvorme. Käesolevas peatükis kirjeldatakse Amazon Web Service teenuseid, mida töö eesmärgi saavutamiseks kasutatakse.

2.1.1 Amazon Web Service

Amazon Web Service on pilveteenuste platvorm, mis pakub palju erinevaid teenuseid. Alates infrastruktuuri tehnoloogiatest nagu arvutus, salvestus ja andmebaasid - kuni

tehnoloogiateni nagu masinõpe ja tehisintellekt [6]. Antud töö lahenduse loomisel on olulised järgmised AWS teenused:

- Amazon SES

Amazon SES on e-posti platvorm, mis pakub e-kirjade saatmise ja vastuvõtmise teenust, kasutades oma e-posti aadresse ja domeene [7].

- Amazon SQS

Amazon SQS on täielikult hallatav sõnumijärjekordade teenus, mis muudab mikroteenuste, hajutatud süsteemide ja serverita rakenduste eraldamise ja laiendamise lihtsaks. Amazon SQS liigutab andmeid hajutatud rakenduse komponentide vahel ja aitab komponente lahti ühendada [8].

- Amazon SNS

Amazon SNS on veebiteenus, mis võimaldab rakendustel, lõppkasutajatel ja seadmetel saata ja vastu võtta teateid pilvest [9].

- Amazon S3

Amazon S3 on objektide salvestusteenus, mis pakub valdkonnas juhtivat skaleeritavust, andmete saadavust, turvalisust ja jõudlust. See tähendab, et seda saab kasutada kasutada suvalise hulga andmete salvestamiseks ja kaitsmiseks mitmesugustel kasutusjuhtudel, näiteks veebisaidid, mobiilirakendused, varundamine ja taastamine, arhiivimine, ettevõtterakendused, Interneti-seadmed ja suurandmete analüüs [10].

- Amazon Lambda

AWS Lambda abil saab koodi käivitada ilma servereid haldamata. Koodi saab käivitada virtuaalselt igat tüüpi rakenduste või taustteenuste korral. Koodi saab seadistada nii, et see aktiveerub automaatselt trigerist [11]. Lambda kasutamisel saab keskenduda koodi kirjutamisele, mitte infrastruktuuri konfigureerimisele. Kuid virtuaalmasina kasutamisel tuleb tegeleda rohkem konfigureerimisega. Lambda on serverivaba, mis põhimõtteliselt tähendab, et kood jooksutatakse ilma serverit käivitamata. Virtuaalmasina puhul käivitatakse aga server.

- Amazon Textract

Amazon Textract on teenus, mis ekstraheerib skannitud dokumentidest teksti ja andmeid automaatselt. Lisaks optilise märgituvastusele (OCR) suudab Amazon Textract ka väljade sisu vormides ja tabelitesse salvestatud teave tuvastada. Amazon kasutab masinõpet dokumentide lugemiseks ning teksti ja andmete eraldamiseks ilma manuaalse pingutuse või kohandatud koodi vajaduseta [12].

- Amazon DynamoDB

Amazon DynamoDB on täielikult hallatav NoSQL andmebaasiteenus, mis tagab kiire ja prognoositava jõudluse sujuva skaleeritavuse abil. Amazon DynamoDB abil saab luua andmebaasi tabeli, kuhu saab salvestada ja välja otsida mis tahes mahus andmeid. DynamoDB jaotab tabeli andmed ja liikluse automaatselt piisava arvu serverite peale, et hallata kliendi määratud päringumahtu ja salvestatud andmete kogust, säilitades samal ajal ühtlase ja kiire jõudluse [13].

- Amazon Cloudwatch

CloudWatch kogub seire- ja tööandmeid logide, mõõdikute ja sündmuste kujul. Selle abil saab tuvastada keskkonnas anomaalset käitumist, seada häireid, visualiseerida logisid ja mõõdikuid kõrvuti, teha automatiseeritud toiminguid, probleemide tõrkeotsingut ja avastusi, et hoida oma rakendusi sujuvalt töös [14].

- AWS halduskonsool

AWS halduskonsool on veebirakendus Amazoni veebiteenuste haldamiseks. See konsool pakub sisseehitatud kasutajaliidest AWS-i ülesannete täitmiseks, näiteks tööks Amazon S3 salvestusruumidega või Amazon CloudWatchi häirete seadistamiseks[15].

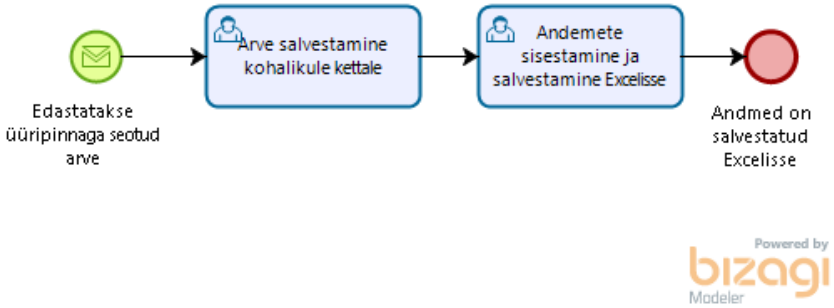
3 Olemasoleva olukorra kirjeldus

Käesolevas peatükis antakse ülevaade arvete ja andmete salvestamise protsessi hetkeseisut, mis aitab paremini arusaada esile kerkinud probleemidest.

Kõik teenusepakkujate arved saadetakse Securebadgeri e-posti aadressile. Arve saatmiseks on mitu võimalust. Üks võimalus on, et üürileandja ehk kinnisvaraomanik edastab üüriobjektiga seonduvad arved otse Securebadgeri e-posti aadressile. Teine võimalus on üüriobjektiga seotud arvete saatmiseks lisada teenusepakkuja arve edastamise aadressiks Securbadgeri e-post, mille tulemusena suunab teenusepakkuja arve otse Securebadgerile.

Ühe üüriobjektiga võib seonduda mitmeid kulutusi. Näiteks korteriühistu saadab iga kuu kommunaalarve, kuid tihti ei kata see kõiki kinnisvara kasutamisega seotud kulutusi. Eraldi peab maksma tarbitud elektri või muude kulutuste eest. Kui üüripinnaks on maja, siis arveid, mida tuleb üürnikul tasuda võib tekkida veel rohkem kui korterite puhul. Kõik kinnisvaraomanike või teenusepakkujate poolt saadetud kommuunaalarved salvestab Securebadger kohalikule kettale. Iga üüriobjekti kohta on tekitatud eraldi kaust, kus arved salvestatakse. Arvete salvestamine on oluline, et pärast koondarve genereerimist oleks võimalik vajaduse korral kontrollida andmete õigsust.

Peale üüriobjektiga seotud arve salvestamist sisestatakse manuaalselt vajalikud parameetrid Excelisse. Excelisse salvestatakse teenusepakkuja nimi, maksetähtaeg, teenused, kogu summa, aadress ning arve saaja. Andmete sisestamine Excelisse annab ülevaate kõikidest üüriobjektidest ja nendega seotud igakuistest kulutustest. Joonisel 2 on visuaalselt antud ülevaade olemasolevast olukorrast.



Joonis 2. Arvete salvestamise ja andmete sisestamise ning salvestamise AS IS protsess.

Praegune kommunaalarvete salvestamine on väga primitiivne ja ei ole jätkusuutlik suureneva kliendibaasi tõttu. AS IS protsessi jätkamine on ajakulukas ja tülikas ning arvete mahu kasvamise tõttu raskesti hoomatav. Järgnevalt on kaardistatud olemasoleva olukorraga kaasnevad probleemid:

- Kommunaalarvete manuaalne salvestamine kohalikule kettale ei ole hea praktika. Vahel võib jääda arve salvestamata ja seejärel tuleb hakata seda otsima suure hulga e-kirjade seast;
- Andmete sisestamine manuaalselt on aeganõudev ja tülikas;
- Mahu suurenemise tõttu suureneb järjest enam risk inimvigade tekkimiseks. Andmete manuaalsel sisestamisel Excelisse võib tekkida näpuvigu;
- Manuaalne andmete salvestamine Excelisse on suurte mahtude puhul kallim võrreldes automatiseerimisega, sest ajakulu ja tööjõukulu on suurem suurte mahtude puhul;
- Puudub hea ülevaade kõikidest andmetest.

3.1 Domeenimudel

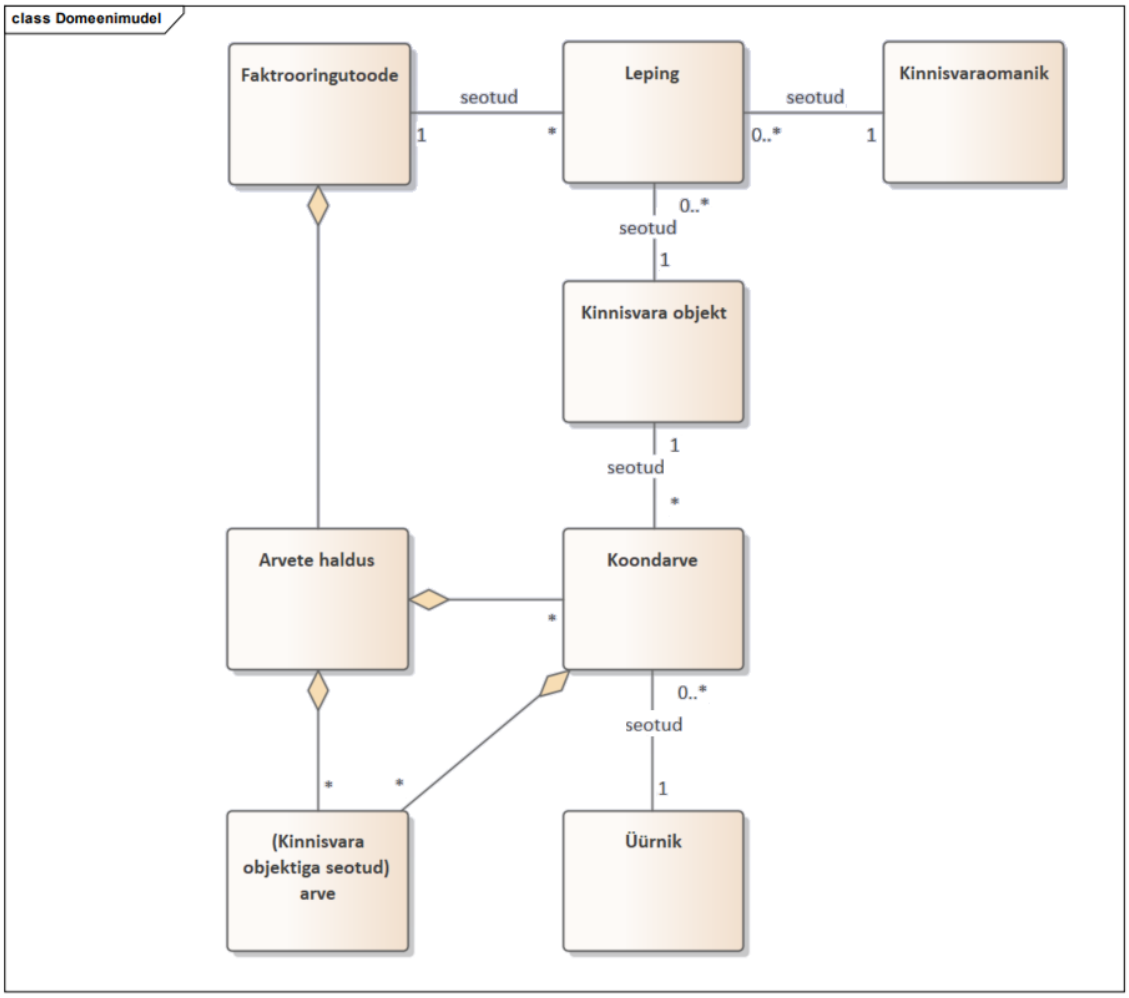
Käesolevas peatükis annab töö autor ülevaate ettevõtte domeenimudelist. Domeenimudel on konkreetse domeeni mõistete ja suhete kogum [16]. Joonisel 3 on domeenimudel, mis annab tervikpildi arvete haldusega kaasnevast andmete salvestamise kontseptsioonist.

Domeenimudeli kirjeldus:

Faktooringutoode sisaldab kinnisvara objektiga seotud arvete haldust. Faktooringutoote kasutamiseks peab kinnisvaraomanikul olema sõlmitud leping. Kui kinnivaraomanik tahab kasutada faktooringutoodet mitmel kinnivara objektil, siis tuleb sõlmida iga objekti

kohta eraldi leping. Leping on seotud alati ühe kinnivaraomanikuga ja kinnisvara objektiga. Arvete haldus sisaldab mitmeid kinnivara objektiga seotud arveid ja koondarveid. Koondarvete koostamisel kasutatakse kinnivara objektiga seotud arvete andmeid. Koondarve üldjuhul sisaldab andmeid mitmelt üüriobjektiga seotud arvelt. Koondarve on alati seotud ühe üürnikuga ja üürnikul võib olla mitu koondarvet. Iga kuu esitatakse üürnikule üks koondarve. Kui arvete halduse teenust ei kasutata, siis ei teki ühtegi koondarvet üürnikule. Koondarve on seotud alati ühe kinnisvara objektiga, kuid kinnisvara objektil võib olla mitu koondarvet. Kinnisvara võib olla seotud nulli või mitme lepinguga.

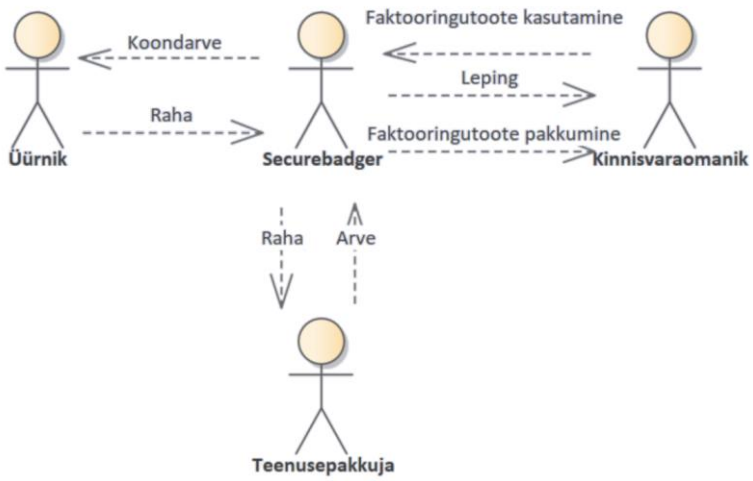
Kinnisvara objektiga seotud arvete olemi attribuudid on kirjeldatud Joonisel 12 koos andmetüüpidega.



Joonis 3. Faktooringutoote domeenimudel.

3.2 Väärtusvahetuse mudel

Antud peatükis on ülevaade väärtusvahetuse mudelist. Väärtusvahetuse mudel on graafiliselt modelleeritud seos osalejate vahel [17]. Joonisel 4 on väärtusvahetuse mudel, mis kirjeldab Securebadgeri ja vastavate osapoolte vahelist seost.



Joonis 4. Faktooringutoote arve halduse väärtusvahetuse mudel.

Järgnevalt on Joonise 4 alusel kirjeldatud osalejate tegevused:

Kinniavaraomanik – isik, kes kasutab Securebadgeri poolt pakutavat faktooringutoodet.

Üürnik – isik, kellele esitatakse koondarve, mis sisaldab üüriobjektiga seonduvaid kulusid.

Teenusepakkuja – ettevõte, kes esitab Kinnisvaraomanikule üüriobjektile pakutava teenuse eest arve, mis liigub läbi Securebadgeri arvehalduse süsteemi üürnikule.

Securebadger – ettevõte, pakub faktooringutoodet kinnisvaraomanikule, esitab koondarveid üürnikele ja tasub teenusepakkujate poolt esitatud arved.

4 Nõuded süsteemile

Selleks, et arvete salvestamist ja andmete välja lugemist ning salvestamist automatiseerida, tuleb eelnevalt koostöös Securebadgeriga välja selgitada nõuded. Nõuete kogumiseks kasutati struktureerimata intervjuu metoodikat huvigruppide võtmeisikutega. Kui kõik andmed on kogutud, siis toimub nõuete kaardistamine. Kaardisatatud nõuded on jagatud funktsionaalseteks ja mittefunktsionaalseteks nõueteks.

4.1 Funktsionaalsed nõuded

Järgnevalt on kirjeldatud funktsionaalsed nõuded, mis täpsustavad funktsiooni, mida süsteem või komponent peab tegema [18]. Tabelis 1 on kirjas funktsionaalsed nõuded süsteemile:

FN01	Süsteem peab ettevõtte e-posti aadressile saadetud arved automaatselt salvestama.
FN02	Süsteem peab iga arve salvestama unikaalse nimega.
FN03	Süsteem peab võimaldama manuaalselt arvet alla laadida ja üles laadida.
FN04	E-posti aadressile saadetud arvete vajalikud andmed salvestatakse automaaselt andmebaasi.
FN05	Süsteem peab võimaldama teavituste välja saatmist Slacki.
FN06	Salvestatud andmeid peab olema võimalik käsitsi muuta andmebaasis.
FN07	Süsteem peab arusaama, millise üüripinna kohta arve saadeti.
FN08	Süsteem peab arusaama, millise teenusepakkuja arvega on tegemist.

Tabel 1. Arvete salvestamise ja andmete välja lugemise ning salvestamise lahenduse funktsionaalsed nõuded.

4.2 Mittefunktsionaalsed nõuded

Järgnevalt on kirjeldatud süsteemi mittefunktsionaalsed nõuded. Mittefunktsionaalsed nõuded määravad, kuidas süsteem peab vajalikke funktsioone täitma. Mittefunktsionaalsed nõuded puudutavad tarkvarasüsteemi või tarkvaratoote kvaliteeti, keskkonda, milles tarkvarasüsteem või tarkvaratoode tuleb rakendada ja mida see peab teenima, ning tarkvara arendamiseks ja hooldamiseks kasutatavaid protsesse ja tehnoloogiat [18]. Tabelis 2 on kaardistatud mittefunktsionaalsed nõuded.

MFN01	Süsteem peab kasutama Amazon Web Service (AWS) pilveandmetöötluse platvormi.
MFN02	Programmikood peab jooksma Lambdades.
MFN03	Süsteem salvestab arved Amazoni S3 salvestusruumi.
MFN04	Arvetelt andmete välja lugemiseks peab kasutama Amazoni Textracti.
MFN05	Kommunaalarvete andmed peab salvestama Amazon DynamoDB-sse.
MFN06	Süsteem peab toetama Python 3.5 või uuemaid versioone.

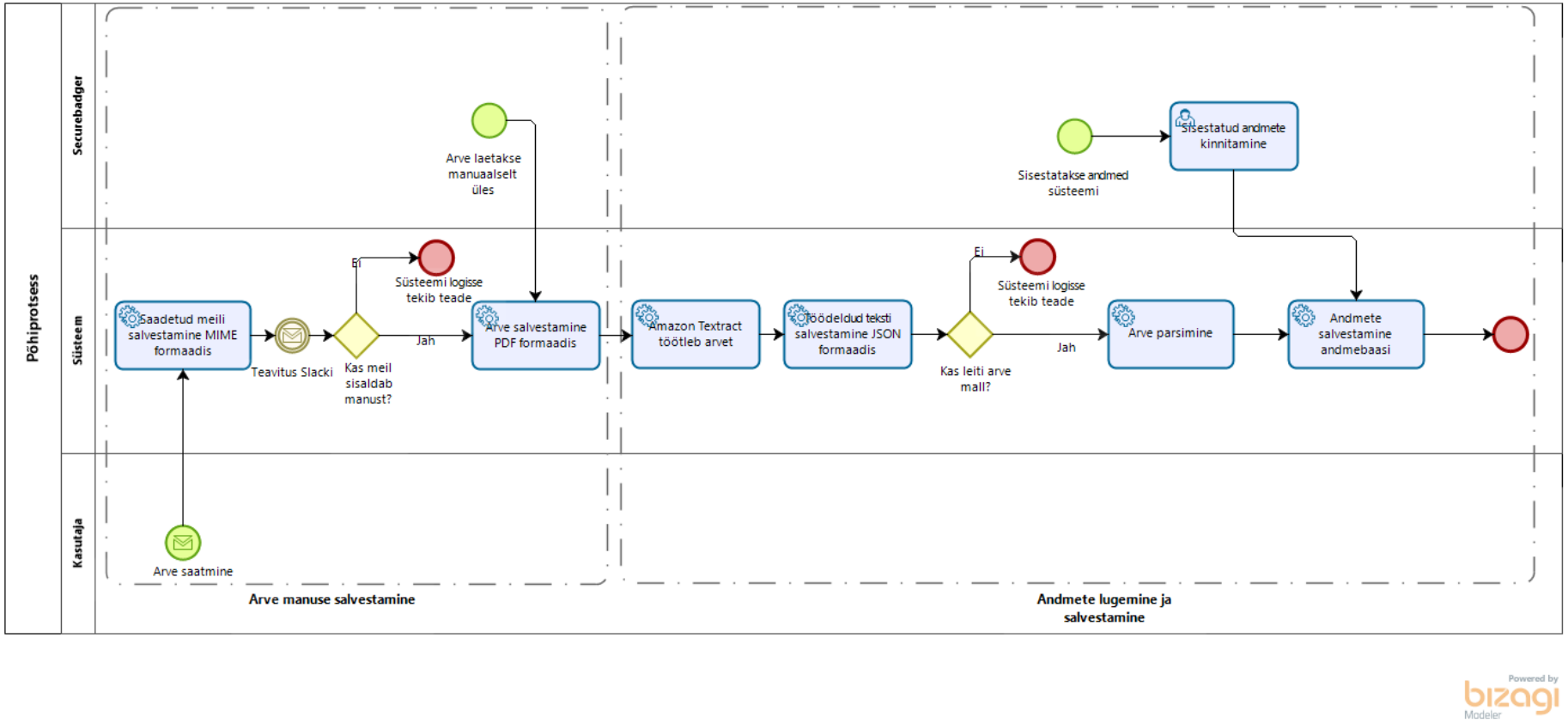
Tabel 2. Automatiseeritud lahenduse mittefunktsionaalsed nõuded.

5 TO BE protsess

Peale funktsionaalsete ja mittefunktsionaalsete nõuete kaardistamist saab edasi liikuda arvete salvestamise ja andmete välja lugemise ning salvestamise protsessi kaardistamisega. TO BE protsessi kaardistamine aitab paremini mõista tervet süsteemi ning kontrollida, kas kõik osapooled saavad protsessist ühtemoodi aru. Joonisel 5 on antud ülevaade TO BE põhiprotsessist. Põhiprotsess jaguneb arve salvestamiseks ja andmete lugemise ning salvestamise protsessiks.

Põhiprotsessi sõnaline kirjeldus:

Põhiprotsess algab sellega, et Securebadgerile saadetakse e-posti aadressile üüriobjektiga seotud arve. Kui arve on saadetud, siis süsteem salvestab automaatselt saadetud meili Amazon S3 salvestusruumi MIME formaadis. Peale meili salvestamist edastatakse teavitus Slacki, mis annab Securebadgerile märku arve laekumisest. Slacki teavituse saatmiseks kasutatakse Amazon lambdat. Järgmisena kontrollitakse, kas salvestatud MIME formaadis asub manus. Kui manus puudub, siis Amazon CloudWatchi logisse tekib teade. Manuse olemasolul salvestatakse arve PDF formaadis Amazoni S3 salvestusruumi. Iga salvestatud arvele antakse unikaalne nimi. Kasutades Amazoni halduskonsooli on võimalik arveid manuaalselt üles ja alla laadida. Peale arve salvesatamist teavitab lambda Amazon Textracti, et võib alustada PDF formaadis arve töötlemisega, mis muudab arve tekstiks. Amazon Textracti töö lõpetamisel salvestatakse töötluse tulemusena välja loetud ja eraldatud tekst JSON formaadina S3 salvestusruumi. Salvestamise tulemusel käivitatakse lambda, mis kasutades teenupakkuja nime kontrollib, kas antud arve mall on olemas. Malli puudumisel tekib Amazon WatchCloud logisse teade. Kui leiti arve mall, siis siis hakkab süsteem arvet parsima. Arve parsimise tulemusena saadud andmed salvestatakse DynamoDB andmebaasi. Kasutades Amazoni halduskonsooli on võimalik DynamoDB-sse sisestatud andmeid manuaalselt muuta. Protsess lõppeb andmete salvestamisega andmebaasi.



Joonis 5. Arvete salvestamine ning andmete lugemise ning salvestamise automatiseetitud lahenduse TO BE protsess.

6 Arendusprotsess

Antud peatükis keskendub töö autor arendusprotsessi kirjeldamisele. Kogu arendus viidi läbi kasutades testipõhist arendust. Testipõhine arendus on disaini ja programmeerimise distsipliin, kus iga uue koodi rida kirjutatakse vastusena testile, mis kirjutatakse vahetult enne kodeerimist [18]. Testimise kohta lähemalt on kirjutatud peatükis kaheksa.

Funktsionaalsuste sisse viimiseks kasutatakse AWS lambdat, mis toetab väheseid programeerimiskeeli. Üheks selleks keeleks on programmeerimiskeel Python. Töö lahenduse kodeerimisel kasutatakse Python 3.8 programmeerimiskeelt. Pythonis lambda funktsiooni kirjutamist lihtsustakse kasutades Chalice'i. Chalice'i abil kirjutatakse Lambda funktsiooni, testitakse ja tarnitakse AWS keskkonda [19]. Chalice on Pythoni mikroraamistik serverita arenduste tegemiseks kasutades AWS lambdat, mille kasutamiseks tuleb see kohalikku masinasse installida [20]. Chalice seadistatakse integreeritud arenduskeskkonnaga Pycharm, peale mida saab Chalice funktsioone Lambdas juurutada ja luua nende funkstioonde jaoks API-lüüsid. Lambda funktsioonide tarnimiseks AWS keskkonda kasutatakse Pycharmi käsureal Chalice käsku. Kui tarne on tehtud tagastatakse API-lüüsi avalik URL [19].

Versioonikontrolli tööriistana kasutatakse Bitbucketit, mis aitab jälgida koodi muutusi. Bitbucket on Gitil põhinev koodide majutamise ja koostöö tööriist, mis on loodud meeskondadele [21]. Töö käigus valminud kood lükati Bitbucketi, kuhu on tekitatud eraldi repositoorium antud osa jaoks.

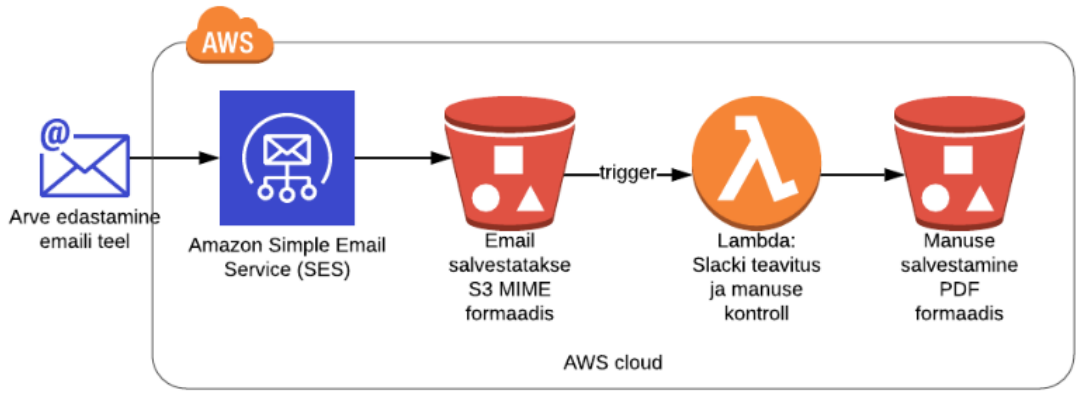
7 Realisatsioon

Antud peatükis kirjeldab töö autor arenduse käigus valminud tulemust. Töö realisatsioon on jagatud kaheks protsessiks, mis on esile toodud ka TO BE protsessi juures (Joonis 5).

- Arve manuse salvestamine;
- Andmete lugemine ja salvestamine.

7.1 Arve manuse salvestamine

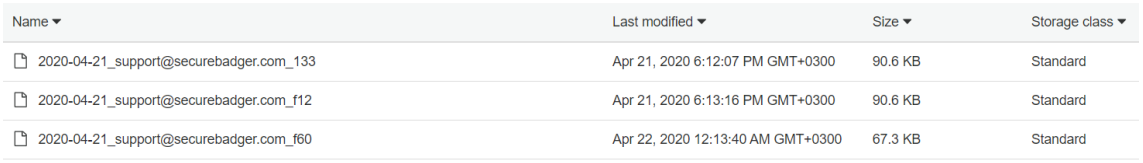
Käesolevas peatükis antakse ülevaade realiseeritud arve manuse salvestamisest (Joonis 6, lk 27). Protsess algab üüriobjektiga seotud arve saatmisega ettevõtte e-posti aadressile.



Joonis 6. Kommunaalarve manuse salvestamise protsess.

Selleks, et e-posti aadressile saadetud arved kätte saada, tuleb Amazoni halduskonsoolis seadistada SES teenus ettevõtte e-posti aadressiga. Tuleb luua reegel, kus määratakse e-posti aadress ja S3 salvestusruum [22]. Peale reegli lisamist salvestatakse ettevõtte aadressile saadetud kirjad Amazon S3 salvestusruumi MIME formaadis. Peale MIME formaadi salvestamist peab süsteem teavitama Slacki ja kontrollima, kas salvestatud meilil on olemas manus. Slacki teavitus ja manuse kontroll ning salvestamine on lisatud ühe lambda alla. Selleks, et lambda käivitatakse on vaja trigerit. Lambda trigeriks on lisatud objekti tekkimine S3 salvestusruumi. Peale objekti lisamist käivitub lambda (Lisa 1). Kui lambda saadab Slacki teavituse ja kontrollib manuse olemasolu, siis manuse olemasolul salvestatakse dokument S3 salvestusruumi PDF formaadis. Igal arve

salvestatakse unikaalse nimega, mis sisaldab arve saatmise kuupäeva, arve saatjat ja genereeritud unikaalset juhuslikku ID-d. Joonisel 7 on näha Amazon S3 salvestatud arveid, mis on unikaalse nimega.

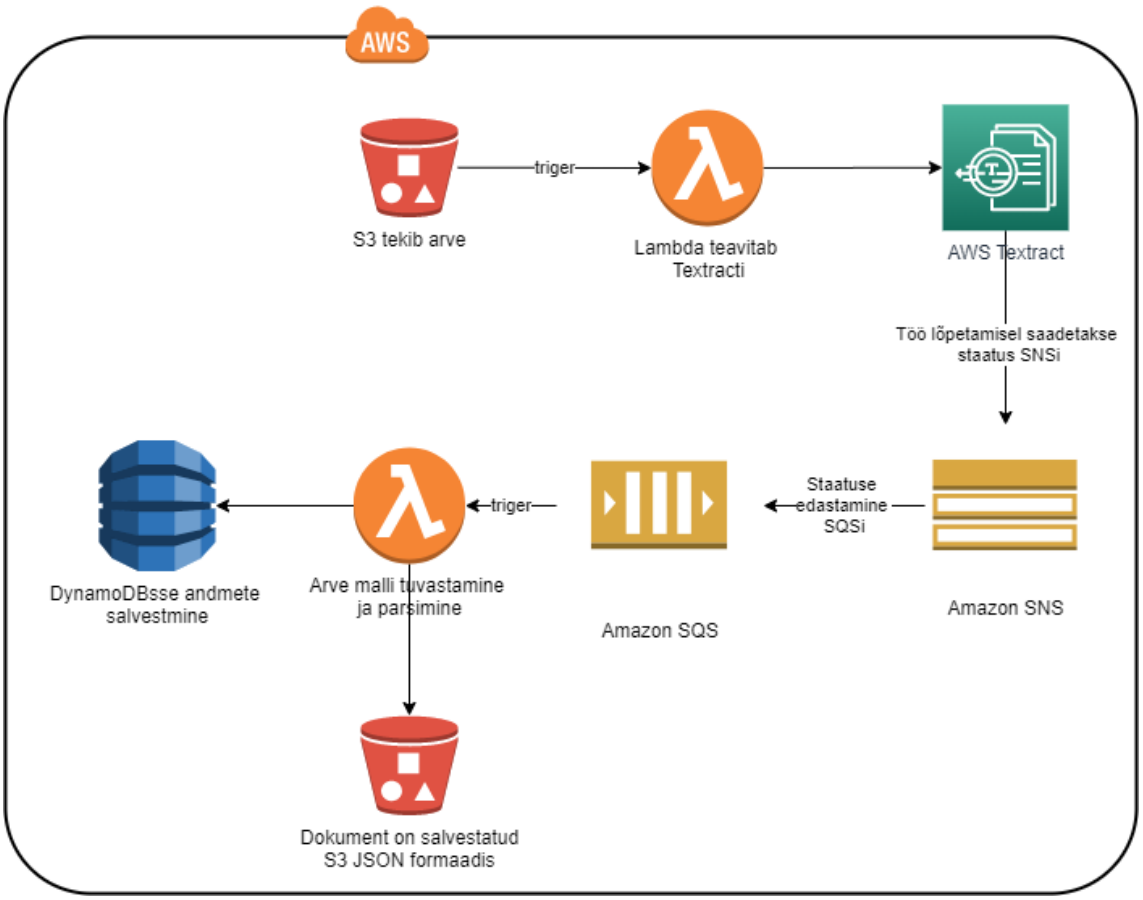


Joonis 7. Arvete salvestamine S3 salvestusruumi unikaalse nimega.

Kasutades Amazoni halduskonsooli on võimalik arveid manuaalselt üles või alla laadida. Antud protsess lõppeb manuse salvestamisega S3 salvestustuumis ning edasi toimub arvelt andmete välja lugemine ja salvestamine.

7.2 Andmete lugemine ja salvestamine

Kui arve manus on salvestatud S3 salvestusruumi nagu kirjeldatud peatükis 7.1, siis saab edasi liikuda andemete lugemise ja salvestamise realisatsiooni kirjeldamisega (Joonis 8, lk 29). Andmete lugemiseks tuleb kasutusele võtta Amazon Textract teenus, mida kasutatakse dokumentide lugemiseks ning teksti ja andmete eraldamiseks. Amazon Textract pakub sünkroonseid toiminguid väikeste üheleheküljeliste dokumentide töötlemiseks ja reaalajas vastuste saamiseks. Lisaks pakub Textract ka asünkroonseid toiminguid, mida on võimalik kasutada suuremate, mitmeleheküljeliste dokumentide töötlemiseks [23]. Kasutusele tuleb võtta asünkroone toiming, sest sünkroone toiming ei toeta PDF formaati.



Joonis 8. Andmete töötlus kasutades Amazon Textracti ja andmete salvestamine DynamoDBsse.

Protsess algab sellega, et lambda käivitamiseks tuleb tekitada triger. Triger on lisatud lambdale juurde:

```
@app.on_s3_event(bucket='securebadger-invoices', prefix='incoming',
                 events=['s3:ObjectCreated:*'])
```

Joonis 9. Lambda trigeri koodi näide

Joonisel 9 on näha, et triger kutsub lambda funktsiooni esile (Lisa 2), kui S3 salvestusruumi on uus objekt loodud. Lambda teavitab Textracti uue arve olemasolust. Textract võtab salvestusruumist PDF arve ja hakkab asünkroonselt töötlema tekstiks. Kui Textract on oma töö lõpetanud, siis saadetakse staatus SNS teavitusteenusele. SNS teavitusteenus omakorda saadab staatuse SQS sõnumi järjekorra teenusele. SNS ja SQS loodi läbi Amazoni halduskonsooli, peale mida tuleb tellida SQS-i järjekord SNS teemale [24]. Staatuse saabumine SQSi on arve malli tuvastamise ja parsimise lambda trigeriks (Lisa 3). Lambda funktsioon võtab Textracti abil töödeldud teksti, mis on JSON formaadis ja tuvastab teenusepakkuja nime, kelle arvega on tegu. Teenusepakkuja nime

järgi otsib süsteem üles vastava malli, mille alusel JSONit parsida. Joonisel 10 on toodud näide parsitud arvest.

```
"ParsedFields": {
    "Aadress": "Akadeemia tee 10 - 20",
    "InvoiceDeadline": "16.01.2020",
    "InvoiceItems": {
        "TaastuvenergiaTasuJaElektriaktsiis": "2,02",
        "Vorguteenus": "7,92"
    },
    "InvoiceNumber": "448793659010",
    "Name": "Karl Marka",
    "ServiceProvider": "elektrilevi",
    "Total": "11,93"
}
```

Joonis 10. Parsitud arve näide

Malli abil saab määrata, kus asuvad vajalikud andmed. Mallis määratud parameetrid on aadress, maksetähtaeg, teenuse sisu, arve number, arve saaja, teenusepakkuja nimi ja arve summa. Peale JSONi parsimist salvestab lambda andmed andmebaasi ja parsitud JSONi S3 salvestusruumi. Lambda abil sisestatakse vajalikud andmed DynamoDB andmebaasi. Joonisel 11 on näidatud lambda koodi, kus määratakse, millised andmed sisestatakse andmebaasi.

```
def insert_invoice_info_into_ddb(filename, invoice_fields):
    ddb = boto3.resource('dynamodb')
    table = ddb.Table(DDB_TABLE_NAME)
    print("Inserting items to dynamodb table".format(DDB_TABLE_NAME))
    item_to_insert = {
        'FileName': filename,
        'ServiceProvider': invoice_fields['ServiceProvider'],
        'Name': invoice_fields['Name'],
        'InvoiceNumber': invoice_fields['InvoiceNumber'],
        'Aadress': invoice_fields['Aadress'],
        'InvoiceDeadline': invoice_fields['InvoiceDeadline'],
        'InvoiceItems': invoice_fields['InvoiceItems'],
        'Total': invoice_fields['Total']
        }
    table.put_item(Item=item_to_insert)
```

Joonis 11.Andmete andmebaasi sisestamise lambda.

Andmete salvestamiseks on tekitatud Amazoni DynamoDB andmebaasi tabel (Joonis 12, lk 31). Andmebaasi tabeli seos on nähtav ka domeenimudelil (Joonis 3). Domeenimudeli olemi nimetus on kinnisvara objektiga seotud arve.

Utility_invoices		
PK	FileNameID	String
	ServiceProvider	String
	InvoiceNumber	String
	InvoiceItems	Map
	InvoiceDeadline	String
	Total	Number
	Name	String
	Aadress	String

Joonis 12. Arve andmete andmebaasi tabel.

Andmebaasi sisestatud andmeid on võimalik vaadata ka Amazoni halduskonsooli kaudu. Halduskonsooli abil saab vajadusel andmeid andmebaasis manuaalselt muuta. Joonisel 13 on antud ülevaade, kuidas halduskonsooli kaudu kuvatakse DynamoDBsse salvestatud andmeid.

FileName	Aadress	InvoiceDeadline	InvoiceItems	InvoiceNumber	Name	ServiceProvider	Total
PDF_attachment	Vindi 18a	30.01.2020	{ "AvariiJaH...	80148352		KU Vindi 18A	130,73
PDF_attachment	Vindi 18a	30.01.2020	{ "AvariiJaH...	80148352		KU Vindi 18A	130,73
PDF_attachment	Vindi 18a	30.01.2020	{ "AvariiJaH...	80148352		KU Vindi 18A	130,73
PDF_attachment	Akadeemia te...	16.01.2020	{ "Taastuve...	448793659010	Karl Marka	elektrilevi	11,93
PDF_attachment	Akadeemia te...	16.01.2020	{ "Taastuve...	448793659010	Karl Marka	elektrilevi	11,93

Joonis 13. Amazoni halduskonsooli andmete kuvamine.

Andmete lugemise ja salvestamise protsess lõppeb andmete salvestamisega andmebaasi.

7.2.1 Arve malli loomine

Peatükis 7.2 mainiti, et teenusepakkuja nime põhjal otsustatakse, millise malli alusel JSONi parsida. Selles peatükis vaatame lähemalt, kuidas malli loomine toimub.

Textract töötleb PDF arvet ja väljastab JSON formaadis teksti. Kui Textract on PDF dokumendist andmed ekstraheerinud, siis lisatakse iga andmeväljale number ning prinditakse välja CloudWatch-is. Joonisel 14 on näide CloudWatch-is välja prinditud logist:

Joonis 14. Amazon CloudWatch-is välja prinditud arve.

Kui on teada, kus real asuvad vajalikud andmed, siis saab lambdale juurde lisada malli. Kõikidel mallidel on väljade nimetused samad, kuid asukohad on erineva numbriga. Joonisel 15 on lisatud näide mallist, kus on defineeritud parameetrite asukohad:

```
        if sp == "elektrilevi":
            parsed_fields = {
                'ServiceProvider': response['Blocks'][1]['Text'],
                'Name': response['Blocks'][3]['Text'],
                'InvoiceNumber': response['Blocks'][5]['Text'],
                'Aadress': response['Blocks'][10]['Text'],
                'InvoiceDeadline': response['Blocks'][12]['Text'],
                'InvoiceItems': {
                    'Vorguteenus': response['Blocks'][19]['Text'][:-
2],
                    'TaastuvenergiaTasuJaElektriaktsiis':
response['Blocks'][22]['Text'][:-2],
                },
                'Total': response['Blocks'][36]['Text'][:-2],
            }
```

Joonis 15. Koodi näide mallist, kus on defineeritud parameetrite asukohad.

Peale malli lisamist tuleb veel määrata, kus asub arvel teenusepakkuja nimi, mille alusel mall leitakse. Teenusepakkuja nimi võetakse parsitud JSONist.

8 Testimine

Arvete ja andmete salvestamise testimiseks kasutati automaattestimist. Testide kirjutamiseks tuleb eelkõige kindlaks teha, mida peab testitav funktsioon tegema ja mida see peab erinevate argumentide korral tagastama [25]. Antud testimise korral kasutatakse unittest tööriista. Üksuste testimise (inglise k. unit testing) puhul testitakse ühte konkreetset üksust koodist.

Joonisel 16 on toodud töö tulemuse testimisel kasutatud üksuste testi näide:

```
class TestApp(TestCase):

    def setUp(self):

        self.sns_message =
self.to_sns_message_format(MAILS3BUCKET,"incoming/2akcmcqk
1lj7fomt8oa58p4n0niaj94jsff4ss01")


    def to_sns_message_format(self, bucket, key):
        return {"Records": [
        {
            "s3": {
                "bucket": {
                    "name": bucket,
                },
                "object": {
                    "key": key,
                }
            }
        }
    ]
}
    def test_send_sns(self):

assert_that(handle_s3_event(self.sns_message,None), True)
```

Joonis 16. Arvete manuste salvestamise üksuste testi näide.

Antud üksuste testiga testiti e-posti teel saadetud arvete manuste salvestamist S3 salvestusruumi. Kõikide AWS lambdade testimiseks koostati üksuste testid, mille abil on võimalik tulemusi valideerida.

9 Tulemuste analüüs ja järeldused

Käesolevas peatükis toimub tulemuste analüüsimine ning järelduste tegemine. Lisaks kirjeldatakse võimalikke alternatiive ja, millised edasiarendused on plaanis tulevikus.

Magistritöö tulemusena töötati välja automatiseeritud lahendus, mis on võimeline e-posti aadressile saadetud arve manuselt vajalikud andmed salvestada DynamoDB andmebaasi. Valminud lahendus vastab töös püstitatud funktsionaalsetele- ja mittefunktsionaalsetele nõuetele. Lahendus võimaldab saadetud arve manuse salvestatamist automaatselt AWS platvormil olevale salvestusruumile. Ja lahenduse põhiliseks osaks on andmete andmebaasi salvestamine, mis on lahenduses realiseeritud kasutades AWS platvormil olevat tarkvara Textract ja DynamoDB andmebaasi.

Protsessi automatiseerimisel pidi kasutama AWS platvormi, mis oli lahenduse realiseerimisel üheks piiranguks. AWS platvorm on paindlik ning võimaldab vajalike funktsioonide implementeerimist. Arvete andmete salvestamise protsessi automaatseks muutmisel ei tekkinud muid kitsaskohti.

Töö tulem on kasutusele võetud ettevõttes. Tulemuste realiseerimine on aidanud lahendada mitmeid probleeme ja kiirendanud tööprotsessi. Protsessi automatiseerimisega on muutunud andmete salvestamise aja- ja tööjõukulu väiksemaks. Enam ei pea salvestama kohalikule kettale arvet, vaid süsteem teeb töö kasutaja eest. Ja rohkem ei toimu andmete salvestamist Excelisse, nüüd salvestatakse andmed automaatselt andmebaasi. Lisaks annab AWS platvormi halduskonsool hea ülevaate kõikidest arvetest ja salvestatud andmetest. Ainuke ajakulu ning tööjõukulu nõudev koht on uue malli lisamine. Kui uue teenusepakkuja arve saadetakse, siis tuleb selleks luua mall, mis oskab välja lugeda vajalikud andmed. Kindlasti on vähenenud risk inimvigade tekkimiseks, sest malli alusel võetakse andmed arvelt õigest kohast ning salvestatakse otse andmebaasi.

Tulemuste valideerimisel kasutati üksuste testimist. Kuna oli teada, millist väljundit oodatakse lambda funktsioonidelt, mille pärast üksuste testide abil testimine oli sujuv ja edukas.

Lahenduse kasutamisega on tekkinud mõned mõtted selle täiustamiseks. Üheks mõtteks on saata rohkem Slacki teavitusi. Kui saadetakse uue teenusepakkuja arve ettevõtte e-posti aadressile, millel puudub mall, siis süsteem võiks sellest märku anda Slackis. Seljuhul kasutaja on teadlik ning lisab uue malli. Lisaks võiks Slack teavitada andmete salvestamise protsessiga kaasnevatest erroritest ja, kui saadetud e-kirjal puudub manus. Teiseks ideeks on täiustada malli valimist. Hetkel kontrollitakse, kes on arve teenusepakkuja ning selle alusel toimub malli valimine. Plaanis on lisada topelt kontroll malli valimiseks, et alguses kontrollitakse e-posti saatja aadressi ja järgmisena kontrollitakse teenusepakkuja nime. Need on mõned täiustused, mis on plaanis loodud tulemuse juures muuta.

Töö tulemina valminud arvete automatiseerimise protsessi on võimalik kasutusele võtta ka teistes ettevõtetes või valdkondades, kelle on vaja mitmete arvete alusel välja saata koondarveid.

Lahenduse kasutamisega tekkisid ideed, kuidas muuta veel paremaks süsteemi, aga kokkuvõttes loodud tulemus vastab püstitatud eesmärgile ning lahenduse automatiseerimine muutis tööprotsessi efektiivsemaks ja kiiremaks.

9.1 Alternatiivid

Ülesandepüstituses oli määratud, et andmete salvestamine tuleb rakendada Amazon Web Service platvormil. Antud peatükis kirjeldatakse lühidalt, millised oleks olnud võimalikud alternatiivsed lahendused protsessi automatiseerimisel.

Üheks võimalikuks alternatiiviks oleks olnud kasutada muud pilvteenuse platvormi, mis vastaks vajaminevatelt nõuetele. Üheks näiteks on pilvetöötluse teenus Microsoft Azure. Sarnaselt AWS platvormile on võimalik hoiustada faile, tekstituvastust kasutada ning teavitusi edasta. Alternatiivina ei võetud kasutusele muud pilveteenuse tarkvara, sest kogu ülejäänud süsteem on juba ülesehitatud AWS platvormil.

Teine võimalus on võtta kasutusele majandustarkvara ja integreerida tekstituvastuse tarkvaraga. Majandustarkvara salvestab ja hoiustab andmeid ning dokumente ja tekstituvastuse tarkvara abil saab arvelt andmed kätte. Majandustarkvara aga ei võetud kasutusele, sest ei pakutud ühtegi sellist tarkavara, mis oleks toetanud Eesti teenusepakkujate arveid ning ei toetanud ettevõtte jaoks olulisi funktsioone.

Kolmas alternatiiv on võtta kasutusele e-arved, mille abil saab lihtsalt arveid vastuvõtta ja välja saata. E-arvete tarkvara on võimalik osta või e-arveldamise teenust kasutada. Põhilised arved, mis aga arvete halduse teenuse kasutamisel saadetakse on kommunaalarved. Korteriühistud ei ole võimelised neid saatma. Suuremad teenusepakkujad toetavad e-arvete tellimist, aga see eeldab, et klient viib eelnevalt ise läbi e-arvete tellimise. Korteriühistute poolt saadetud kommunaalarved on kõige enam saadetud arved, kuna korteriühistud ei toeta e-arved, siis selle tulemusena antud alternatiivi ei võetud kasutusele.

9.2 Edasiarendused

Töö lahendus lõppes arvel olevate andmete salvestamisega andmebaasi. Andmebaasi salvestamise vajadus tekkis faktooringutoote arvete halduse teenuse tõttu, mis edastab koondarveid üürnikele igakuiselt. Peale andmebaasi salvestamist on plaanis integreerida DynamoDB andmebaas raamatupidamistarkvaraga Waveapp, kus toimub koondarve genereerimine. Kui on vaja koondarvet koostada, siis edastab süsteem vajalikud andmed WaveAppi. Peale arve koostamist tuleb edastada koondarve üürnikule, kes üürib kinnisvara objekti. Lisaks koondarve genereerimisele ja edastamisele on plaanis luua lahendus, mis maksetähtaja möödumisel edastab meeldetuletusi üürnikule sõnumi teel.

10 Kokkuvõte

Käesolev magistritöö sai alguse ettevõtte vajadusest muuta arve salvestamise ja andmete kätte saamise protsessi. Arvete ning andmete salvestamise protsessis tekkisid probleemid suureneva kliendibaasi tõttu, mille tulemusena suurenes aja- ja tööjõukulu ning tõusis risk inimvigade tekkimiseks. Probleemi püstitusest tulenevalt sai töö eesmärgiks arvete ja andmete salvestamise protsessi automatiseerimine. Töö viidi läbi kasutades tegevusliku disainiuuringu metoodikat. Lahenduse realiseerimisel kasutati testipõhist arendust. Kogu arendus viidi läbi kasutades Amazon Web Service platvormi teenuseid. Töö tulemusena loodi lahendus, mis automatiseeris arvete salvestamise ja andmete lugemise ning salvestamise. Peale arve edastamist ettevõtte e-posti aadressile käivitub automaatne protsess, mis lõppeb vajalike andmete salvestamisega andmebaasi. Tulemuste valideerimisel kasutati üksuste testimist. Valmis lahendus on ettevõttes kasutusele võetud ning tulevikus on plaanis lahenduse edasiarendused. Magistritöö eesmärk on saavutatud.

Kasutatud kirjandus

[1] „Mida tähendab üüritulu faktooring?,“ [Võrgumaterjal]. https://www.securebadger.com/articles-et/mida-tahendab-uuritulu-faktooring. [Kasutatud 05 12 2019].

[2] K. M. Sein, O. Henfridsson, S. Purao, M. Rossi ja R. Lindgren, „Action Design Research,“ MIS Quarterly, 2011. [Kasutatud 17 03 2020].

[3] Carolyn B. Seaman, „Qualitative Methods in Empirical Studies of Software Engineering, “ 1999. [Võrgumaterjal]. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=799955. [Kasutatud 17 03 2020].

[4] J. H. Hayes, A. Dekhtyar, D.S. Janzen, „Towards Traceable Test-Driven Development?,“ 2009. [Võrgumaterjal]. https://dl.acm.org/doi/pdf/10.1109/TEFSE.2009.5069579. [Kasutatud 20 03 2020].

[5] „What is Python?,“ [Võrgumaterjal]. https://www.pythonforbeginners.com/learn-python/what-is-python/. [Kasutatud 12 04 2020].

[6] „Cloud computing with AWS?,“ [Võrgumaterjal]. https://aws.amazon.com/what-is-aws/. [Kasutatud 12 04 2020].

[7] „Amazon Simple Email Service, “ [Võrgumaterjal]. https://aws.amazon.com/ses/. [Kasutatud 04 03 2020].

[8] „Amazon Simple Queue Service Documentation, “ [Võrgumaterjal]. https://docs.aws.amazon.com/sqs/?id=docs_gateway. [Kasutatud 04 03 2020].

[9] „Amazon Simple Notification Service Documentation, “ [Võrgumaterjal]. https://docs.aws.amazon.com/sns/?id=docs_gateway. [Kasutatud 04 03 2020].

[10] „Amazon Simple Storage Service Documentation, “ [Võrgumaterjal]. https://docs.aws.amazon.com/s3/. [Kasutatud 22 03 2020].

[11] „AWS Lambda Documentation, “ [Võrgumaterjal]. https://docs.aws.amazon.com/lambda/?id=docs_gateway. [Kasutatud 04 03 2020].

[12] „What Is Amazon Textract?, “ [Võrgumaterjal]. https://aws.amazon.com/textract/. [Kasutatud 04 03 2020].

[13] „Amazon DynamoDB Documentation, “ [Võrgumaterjal]. fhttps://docs.aws.amazon.com/dynamodb/?id=docs_gateway. [Kasutatud 10 04 2020].

[14] „Amazon CloudWatch, “ [Võrgumaterjal]. https://aws.amazon.com/cloudwatch/. [Kasutatud 04 03 2020].

[15] „AWS – Management Console,“ [Võrgumaterjal]. https://www.tutorialspoint.com/amazon_web_services/amazon_web_services_management_console.htm. [Kasutatud 07 04 2020]

[16] C. Larman, „Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition), “ 2005. [Kasutatud 09 05 2020].

[17] R. Schuster, T. Motal, „From e3-value to REA: Modeling multi-party eBusiness Collaborations, “ 2009. [Võrgumaterjal]. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5210795. [Kasutatud 08 05 2020].

[18] „Systems and software engineering — Vocabulary, “ 2010. [Võrgumaterjal]. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5733835.

[19] „Getting started with Chalice to create AWS Lambdas in Python — Step by Step Tutorial, “ 2019. [Võrgumaterjal]. https://medium.com/swlh/getting-started-with-chalice-to-create-aws-lambdas-in-python-step-by-step-tutorial-3ccf01701259. [Kasutatud 17 02 2020].

[20] „Quickstart and Tutorial, “ 2016. [Võrgumaterjalid]. https://chalice.readthedocs.io/en/latest/quickstart.html. [Kasutatud 17 02 2020].

[21] „A brief overview of Bitbucket, “ [Võrgumaterjalid]. https://bitbucket.org/product/guides/getting-started/overview. [Kasutatud 06 04 2020].

[22] „Setting up Email with Amazon SES,“ [Võrgumaterjalid]. https://docs.aws.amazon.com/ses/latest/DeveloperGuide/send-email-set-up.html. [Kasutatud 18 02 2020]

[23] „Calling Amazon Textract Asynchronous Operations, “ [Võrgumaterjal]. https://docs.aws.amazon.com/textract/latest/dg/api-async.html [Kasutatud 31 03 2020].

[24] „Tutorial: Subscribing an Amazon SQS queue to an Amazon SNS topic, “ [Võrgumaterjal]. https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-subscribe-queue-sns-topic.html. [Kasutatud 15 02 2020]

[25] R. Jeffries, G. Melnik, „Guest Editors' Introduction: TDD--The Art of Fearless Programming, “ 2007. . [Kasutatud 22 04 2020].

Lisa 1 – Slacki teavituse ja manuse kontrollimise lambda

```
import json
import email
import re
from email.mime.application import MIMEApplication
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from datetime import datetime
import uuid

import boto3
from botocore.exceptions import ClientError
from chalice import Chalice
from chalicelib.settings import MAILRECIPIENT, MAILS3BUCKET,
MAILS3INVOICEPDFFOLDER, MAILS3PREFIX, MAILSENDER, REGION

app = Chalice(app_name='PDF_Generator')
app.debug = True


@app.on_s3_event(bucket='securebadger-invoices', prefix='incoming',
                 events=['s3:ObjectCreated:*'])
def handle_s3_event(event):
    app.log.debug(f"Event on: {event.bucket}, key on {event.key}")
    try:
        publish_to_sns({"key": event.key})
        file_dict = get_message_from_s3(event.key)
        extract_invoice_and_upload_s3(file_dict)
        return True
    except Exception as e:
        return False


def publish_to_sns(message):
    # publishing to SNS:
    sns_client = boto3.client('sns', region_name='eu-central-1')
    response = sns_client.publish(
        TopicArn='arn:aws:sns:eu-central-1:205682069028:invoice-
received',
        Message=json.dumps({'default': json.dumps(message)}),
        MessageStructure='json'
    )
```

```
def get_message_from_s3(message_key):
    incoming_email_bucket = MAILS3BUCKET
    object_path = message_key
    object_http_path = (

f"http://s3.console.aws.amazon.com/s3/object/{incoming_email_bucket}/{
object_path}?region={REGION}")
    print(f"S3 URL: {object_http_path}")
    # Create a new S3 client.
    client_s3 = boto3.client("s3")

    # Get the email object from the S3 bucket.
    object_s3 = client_s3.get_object(Bucket=incoming_email_bucket,
                                     Key=object_path)
    # Read the content of the message.
    file = object_s3['Body'].read()

    file_dict = {
        "file": file,
        "path": object_http_path
    }

    return file_dict


def extract_invoice_and_upload_s3(file_dict):
    s3_client = boto3.client("s3")
    dest_bucket = MAILS3BUCKET
    dest_folder_path = MAILS3INVOICEPDFFOLDER
    print(f"dest_bucket={dest_bucket}")
    print(f"dest_folder_path={dest_folder_path}")
    mailobject =
email.message_from_string(file_dict['file'].decode('utf-8'))
    from_address = mailobject['From']
    print(f'from email {from_address}')
    for part in mailobject.walk():
        if part.get_content_type() == "application/pdf":
            filename = part.get_filename()
            f_name = datetime.today().strftime('%Y-%m-%d') + "_" +
from_address + "_" + str(uuid.uuid4())[:3] + ".pdf"
            print(f'the new filename is {f_name}')
            if bool(filename):
                print(f"Processing invoice {filename}")
                invoice_file = part.get_payload(decode=True)
                s3_client.put_object(Body=invoice_file,
Bucket=dest_bucket, Key=f"{dest_folder_path}/{f_name}")


def create_message(file_dict):
    sender = MAILSENDER
    recipient = MAILRECIPIENT

    separator = ";"
```

```
    # Parse the email body.
    mailobject =
email.message_from_string(file_dict['file'].decode('utf-8'))

    # Create a new subject line.
    subject_original = mailobject['Subject']
    subject = "FW: " + subject_original

    # The body text of the email.
    body_text = ("The attached message was received from "
                 + separator.join(mailobject.get_all('From'))
                 + ". This message is archived at " +
file_dict['path'])

    # The file name to use for the attached message. Uses regex to
remove all
    # non-alphanumeric characters, and appends a file extension.
    filename = re.sub('[^0-9a-zA-Z]+', '_', subject_original) + ".eml"

    # Create a MIME container.
    msg = MIMEMultipart()
    # Create a MIME text part.
    text_part = MIMEText(body_text, _subtype="html")
    # Attach the text part to the MIME message.
    msg.attach(text_part)

    # Add subject, from and to lines.
    msg['Subject'] = subject
    msg['From'] = sender
    msg['To'] = recipient

    # Create a new MIME object.
    att = MIMEApplication(file_dict["file"], filename)
    att.add_header("Content-Disposition", 'attachment',
filename=filename)

    # Attach the file object to the message.
    msg.attach(att)

    message = {
        "Source": sender,
        "Destinations": recipient,
        "Data": msg.as_string()
    }

    return message


def send_email(message):
    aws_region = REGION

    # Create a new SES client.
    client_ses = boto3.client('ses', REGION)
```

```
    # Send the email.
    try:
        # Provide the contents of the email.
        response = client_ses.send_raw_email(
            Source=message['Source'],
            Destinations=[
                message['Destinations']
            ],
            RawMessage={
                'Data': message['Data']
            }
        )

    # Display an error if something goes wrong.
    except ClientError as e:
        output = e.response['Error']['Message']
    else:
        output = "Email sent! Message ID: " + response['MessageId']

    return output


def lambda_handler(event, context):
    # Get the unique ID of the message. This corresponds to the name
of the file
    # in S3.
    message_id = event['Records'][0]['ses']['mail']['messageId']
    print(f"Received message ID {message_id}")

    # Retrieve the file from the S3 bucket.
    file_dict = get_message_from_s3(message_id)
    # Put the invoice PDF File into S3
    extract_invoice_and_upload_s3(file_dict)
    # Create the message.
    message = create_message(file_dict)

    # Send the email and print the result.
    result = send_email(message)
    print(result)
```

Lisa 2 – Textracti tesktituvastamise alustamise lambda

```
from chalice import Chalice
from urllib.parse import unquote_plus
import boto3
import pprint

app = Chalice(app_name='Textract')
app.debug = True


@app.on_s3_event(bucket='securebadger-invoices',
prefix='PDF_attachment',
                 events=['s3:ObjectCreated:*'])
def handle_s3_event(event):
    app.log.debug(f"Event on: {event.bucket}, key on {event.key}")
    print(f"Event on: {event.bucket}, key on {event.key}")
    try:
        handler(event)
    except Exception as e:
        print(f"{e}")
        raise e


s3_client = boto3.client('s3')
textract_client = boto3.client('textract')

SNS_TOPIC_ARN = 'arn:aws:sns:eu-west-1:205682069028:AmazonTextract'  #
We need to create this
ROLE_ARN = 'arn:aws:iam::205682069028:role/TextractRole'  # This role
is managed by AWS


def handler(event):
    print(f'In the handler function got {event}')
    pp = pprint.PrettyPrinter(indent=4)
    pp.pprint(event)
    print(f"Event on: {event.bucket}, key on {event.key}")
    key = event.key
    bucket = event.bucket
    job_id = textract_client.start_document_text_detection(
        DocumentLocation={
            'S3Object': {
                'Bucket': bucket,
                'Name': key
            }
```

```
    },
    NotificationChannel={
        'RoleArn': ROLE_ARN,
        'SNSTopicArn': SNS_TOPIC_ARN
    })
print(f'Job ID: {job_id}')
```

Lisa 3 – Textracti tekstituvastuse tulemi salvestamise lambda

```
import json
import boto3
from chalice import Chalice
import pprint
from chalicelib.settings import DDB_TABLE_NAME

app = Chalice(app_name='Textract_json')

textract_client = boto3.client('textract')
s3_bucket = boto3.resource('s3').Bucket('securebadger-invoices')
s3_client = boto3.client("s3")
dest_bucket = 'securebadger-invoices'
dest_folder_path = 'JSON_failid'


# sqs = boto3.client('sqs', region_name="eu-west-1")


@app.on_sqs_message(queue='AmazonTextract')
def handle_sqs_message(event):
    pprint.pprint(vars(event))
    for record in event:
        body_str = record.body
        body = json.loads(body_str)
        print(f'record ...type {type(body)} content {body}')
        # message = record.messageId
        # print(f'message {message}')
        status = body['Status']
        print(f'Status {status}')
        job_id = body['JobId']
        print(f'Job ID {job_id}')
        filename = body['DocumentLocation']['S3ObjectName']
        print(f'file name {filename}')
        print(f'JobId {job_id} has finished with status {status} for
file {filename}')

        if status != 'SUCCEEDED':
            return

        text, parsed_fields = get_document_analysis(job_id)
        to_json = {'Document': filename, 'ExtractedText': text,
'ParsedFields': parsed_fields, 'TextractJobId': job_id}
        json_content = json.dumps(to_json, indent=4,
sort_keys=True).encode('UTF-8')
```

```
        output_file_name = filename.split('/')[-1].rsplit('.', 1)[0] +
'.json'
        print(f'The output filename is {output_file_name}')
        s3_client.put_object(Body=bytes(json_content),
Bucket=dest_bucket, Key=f"{dest_folder_path}/{output_file_name}")
        insert_invoice_info_into_ddb(filename, parsed_fields)
        #
s3_bucket.Object(f'{output_file_name}').put(Body=bytes(json_content))

        return


def insert_invoice_info_into_ddb(filename, invoice_fields):
    ddb = boto3.resource('dynamodb')
    table = ddb.Table(DDB_TABLE_NAME)
    print("Inserting items to dynamodb table".format(DDB_TABLE_NAME))
    item_to_insert = {
        'FileName': filename,
        'ServiceProvider': invoice_fields['ServiceProvider'],
        'Name': invoice_fields['Name'],
        'InvoiceNumber': invoice_fields['InvoiceNumber'],
        'Aadress': invoice_fields['Aadress'],
        'InvoiceDeadline': invoice_fields['InvoiceDeadline'],
        'InvoiceItems': invoice_fields['InvoiceItems'],
        'Total': invoice_fields['Total']
        }
    table.put_item(Item=item_to_insert)


def get_document_analysis(job_id: str, keep_newlines: bool = True) ->
str:
    max_results = 1000
    pagination_token = None
    finished = False
    text = ''

    while not finished:
        if pagination_token is None:
            response =
textract_client.get_document_analysis(JobId=job_id,

MaxResults=max_results)
        else:
            response =
textract_client.get_document_analysis(JobId=job_id,

MaxResults=max_results,

NextToken=pagination_token)

        #total = float(response['Blocks'][36]['Text'])


        # sep = '' if not keep_newlines else '\n'
```

```
        text = ''
        counter = 1
        for x in (response['Blocks']):
            if x['BlockType'] == 'LINE':
                print(f"{counter}. {x['Text']}")
                text += x['Text']
                counter += 1
        parsed_fields = {}
        try:
            sp = response.get('Blocks')[1].get('Text')
        except IndexError as e:
            print(e)
            sp = ""
        if sp == "elektrilevi":
            parsed_fields = {
                'ServiceProvider': response['Blocks'][1]['Text'],
                'Name': response['Blocks'][3]['Text'],
                'InvoiceNumber': response['Blocks'][5]['Text'],
                'Aadress': response['Blocks'][10]['Text'],
                'InvoiceDeadline': response['Blocks'][12]['Text'],
                'InvoiceItems': {
                    'Vorguteenus': response['Blocks'][19]['Text'][:-
2],
                    'TaastuvenergiaTasuJaElektriaktsiis':
response['Blocks'][22]['Text'][:-2],
                },
                'Total': response['Blocks'][36]['Text'][:-2],
            }
        elif sp == "KU Vindi 18A":
            parsed_fields = {
               'ServiceProvider': response['Blocks'][1]['Text'],
               'Name':
response['Blocks'][4]['Text'].replace('Tarbija:', ''),
               'InvoiceNumber':
response['Blocks'][2]['Text'].replace('.nr.', ''),
               'Aadress':
response['Blocks'][6]['Text'].replace('Aadress: ', ''),
               'InvoiceDeadline':
response['Blocks'][11]['Text'].replace('Maksetahtaeg:', ''),
               'InvoiceItems': {
                   'Uhistujuhtimiskulud':
response['Blocks'][21]['Text'],
                   'AvariiJaHooldus': response['Blocks'][25]['Text'],
                   'Remondifond': response['Blocks'][29]['Text'],
                   'Prugivedu': response['Blocks'][33]['Text'],
                   'Kute': response['Blocks'][41]['Text'],
                   'VesiJaKanal': response['Blocks'][45]['Text'],
                   'KulLiimemaks': response['Blocks'][49]['Text'],
                   'Korrashoid': response['Blocks'][53]['Text'],
                   'Raamatupidamine': response['Blocks'][57]['Text'],
                   'Uldelekter': response['Blocks'][61]['Text'],
                   'Kindlustus': response['Blocks'][65]['Text'],
                   'Gaas': response['Blocks'][69]['Text'],
               },
```

```
            'Total': response['Blocks'][83]['Text'],
        }
    pprint.pprint(sp)


    #print(response['Blocks'])

    if 'NextToken' in response:
        pagination_token = response['NextToken']
    else:
        finished = True

return text, parsed_fields
```