TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Hans-Toomas Saarest 163310IAPM

MOBIILSE ISETEENINDUSSÜSTEEMI LIIDESTUSKOMPONENTIDE DISAIN, ANALÜÜS JA REALISATSIOON

Magistritöö

Juhendaja: Enn Õunapuu
PhD

Tallinn 2018

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: Hans-Toomas Saarest

07.05.2018

Annotatsioon

Smarts iseteenindusplatvorm on infosüsteem, mis võimaldab osta poest tooteid, makstes nende eest krüptovaluutas. Töö eesmärk on luua tarkvara, mis võimaldab planeeritaval mobiilsel iseteenindussüsteemil suhelda sellega liitunud poega. Selleks rajatakse Smarts mobiilse iseteenindusplatvormi juurde kaks osa: liidestustarkvara ja suhtlusteenus. Planeeritav liidestustarkvara on mõeldud kasutamiseks poe infosüsteemis, et võimaldada suhtlust Smarts iseteenindusplatvormiga. Smarts infosüsteemis asuv suhtlusteenus on mõeldud olema Smartsi poolne osa, mis suhtleb poe infosüsteemiga.

Antud tööga valmib paralleelselt teinegi magistritöö, mis käsitleb Smarts iseteenindusplatvormi üldist arhitektuuri, maksmist ja detailsemalt turvalisusloogikat. Selles töös aga leitakse nõuded, mida liidestustarkvara ja suhtlusteenus peavad täitma. Nõuete põhjal kavandatakse planeeritavate komponentide arhitektuur, leitakse sobivad tehnoloogiad tarkvara realiseerimiseks ning realiseeritakse planeeritavad komponendid. Seejärel testitakse valminud tarkvara iseseisvalt (üksiku komponendina) kui ka integreerituna Smarts iseteenindusplatvormi, kus testitakse tarkvara toimimist üldises arhitektuuris koos teiste komponentidega.

Töö tulemusena valmivad Smarts iseteenindusplatvormi poega liidestamise komponendid koos neid toetavate teiste osadega (vahemälu andmebaas, otsingumootor). Toote info pärimisel kasutatakse otsingumootorit, et kiirendada süsteemi tööd ja vältida liigset koormust poe infosüsteemile. Vahemälu andmebaasi kasutatakse arve hoidmiseks kasutaja pooliku ostu ajal. Töös valminud komponente on uute nõuete ilmnemisel võimalik edasi arendada. Selles töös valmis põhifunktsionaalsus, et iseteenindussüsteemi põhilisi toiminguid teha.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 54 leheküljel, 5 peatükki, 7 joonist, 3 tabelit.

Abstract

Design, analyze and development of mobile based self-service integration components

Smarts is new mobile based self-service system that uses cryptocurrency as payment currency. This system must communicate with retail store's infosystems for product and invoice exchange purposes. For that reason, the purpose of this work is to design, analyse and develop two communication components for Smarts platform that help Smarts to communicate with stores that are linked to Smarts system. Communication service is a web service that interacts with store's system. Communication service is one of Smarts microservices. On the other hand, second component that will be developed is Smarts SDK (Software Development Kit). Smarts SDK is used in store system to help linking into Smarts self-service platform.

Beside this work there is another thesis on Smarts self-service platform that focuses on general architecture, payments and security. In this work, requirements are formed for both components. Then, based on those requirements the architecture of those components is planned and put into practice. Finally, after development those components will be tested and validated against formed requirements.

The result of this work will be a pair of components that can be used to link any retail store system into Smarts self-service platform. While querying for product a search engine will be used as well as message broker to diminish the load on store's systems. In case new requirements occur, there is possibility to add new features into those two components. The basic functionality for Smarts and linked store is developed with this thesis.

The thesis is in Estonian and contains 54 pages of text, 5 chapters, 7 figures, 3 tables.

Lühendite ja mõistete sõnastik

MVP	*Minimum Viable Product*, Vähimate vajalike võimalustega toode
POJO	*Plan Old Java Object*, Tavaline Java objekt
Pood	Pood tähendab selle töö kontekstis mingi jaekaubandusettevõtte ühte füüsilist poodi (asub äripinnal), kus toimub äritegevus. Siin ei ole poe mõiste all mõeldud jaekaubandusketti üldisemalt.
QR kood	*Quick Response Code*, Ruutkood
REST	*Representational State Transfer*, Veebiteenuse tüüp
Ribakood	Tootel olev numbriline tunnuskood, EAN – *European Article Number*
RPC	*Remote Procedure Call*, Kaugprotseduurikutse
Smarts	Smarts iseteenindusplatvorm ehk kogu infosüsteem koos kõigi mikroteenustega
SDK	*Software Development Kit*, Tarkvaraarenduskomplekt
SSDK	*Smarts SDK*, Smarts iseteenindusplatvormi liidestustarkvara
SCOM	*Smarts Communication*, Suhtlusteenus – mikroteenus Smarts iseteenindusplatvormis
UUID	*Universally unique identifier*, universaalne kordumatu tunnus

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tänapäeval on virtuaalrahas ehk krüptovaluutas maksmine muutumas üha populaarsemaks. Lõuna-Koreas on käimas projekt, mille raames loodetakse võimaldada krüptorahas maksmist rohkem kui 6000 poes [1]. See tekitab omakorda küsimuse, kas krüptorahas maksmine on saamas tuleviku lahutamatuks osaks? Praegu eksisteerivaid iseteenindussüsteeme analüüsides paistab selgelt, et suurem osa neist on kaubandusettevõtte spetsiifilised. Turul on suhteliselt vähe lahendusi, mis võimaldaksid kasutada ühte ja sama iseteenindussüsteemi laialdaselt erinevate jaekaubandusettevõtete kauplustes.

Selles töös käsitletav Smarts iseteenindusplatvorm on iseteenindussüsteem (edaspidi lühidalt Smarts), mis võimaldab selle kasutajal osta poest tooteid tasudes nende eest krüptovaluutas. Nimi „Smarts“ mõeldi välja selle infosüsteemi rajajate poolt. Poes toodete ostmisel on Smartsi kasutamine lihtne: vaja on mobiiltelefoni ja krüptovaluuta olemasolu. Smartsi kasutamine pakub selle kasutaja jaoks sarnast kogemust puldiga iseteenindussüsteemile, sest koosneb sarnaselt kolmest etapist, mida võib üldistada kui: poodi sisenemine (vrd iseteeninduspuldi võtmine), toodete skaneerimine ja maksmine. Puldi asemel saab Smartsi kasutaja kasutada oma nutitelefoni, millel peab olema toimiv kaamera ning internetiühendus.

1.1 Taust ja probleem

Krüptoraha on oma olemuselt eksisteerinud juba mitmeid aastaid. Sellega maksmist aktsepteerivad mitmed ettevõtted ning sellega maksmine levib üha laialdasemalt. Seetõttu võib oletada, et tulevikus muutub krüptoraha veel rohkem võrdväärseks maksevahendiks võrreldes tavalise traditsiooniliste valuutadega [2]. Paraku aga ei ole krüptoraha eest hetkel võimalik mugavalt osta ei toidu- ega tarbekaupu tavalistest poodidest. See tõsiasi viis omakorda ideeni luua selline iseteenindussüsteem, mida oleks poel võimalik oma süsteemi liidestada ning mis oleks poe kliendile lihtne kasutada.

Selle infosüsteemi loomisega tegeleb väike grupp inimesi, kuhu kuuluvad nii infotehnoloogia, reklaami kui ka ärivaldkonna inimesed. Meeskonna üldine eesmärk on saada valmis esialgne MVP, mida seejärel minna potentsiaalsetele klientidele tutvustama. Selle magistritöö kirjutamise hetkel valmib Tallinna Tehnikaülikoolis paralleelselt teinegi magistritöö, mis pakub välja rajatava infosüsteemi üldise arhitektuuri ja käsitleb detailselt krüptomaksete ning süsteemi turvalisusega seotud teemasid.

1.2 Eesmärk

Antud magistritöös keskendutakse Smarts infosüsteemi kahe olulise alamkomponendi disainile, analüüsile, arendusele ja testimisele:

- Liidestustarkvara – *Smarts SDK (Software development kit)* – valmis tarkvara, mille liidestuv pood peab oma infosüsteemis kasutusele võtma. Liidestustarkvara abil on poel võimalus lihtsamalt suhelda Smarts iseteenindussüsteemiga üle HTTP protokolli.

- Suhtlusteenus – *Smarts Communication (SCOM)* – komponent, mis võimaldab Smartsi sisemistel teenustel suhelda Smartsiga liidestunud poe infosüsteemiga. Kasutab sisemiselt liidestustarkvara osi.

Lisaks töö käigus valminud analüüsile ja programmeeritud tarkvarale on plaanis antud magistritööst kujundada Smarts infosüsteemi liidestustarkvara käsitlev dokumentatsioon, mida hiljem on võimalus kasutada seletamaks liidestustarkvara ülesehitust ja sellele rakenduvaid nõudeid.

1.3 Metoodika

Töö koostamisel plaanitakse lähtuda agiilse tarkvaraarenduse põhimõtetest, kus tähtsal kohal on tihedate iteratsioonide kasutamine. Ühe iteratsiooni käigus valmib osa nõuetest ja tarkvarast ning seejärel testitakse valminud osa üldises süsteemis.

Töös plaanitakse kasutada sobivat projektihaldustarkvara, et tagada töö hetkeseisu ülevaade teistele meeskonna liikmetele ning võimaldada seeläbi paremat tööde planeerimist tulevikus. Projektihaldustarkvara eesmärgiks on tagada piisav töövoog ühe iteratsiooni jooksul ning arendusmeeskonna pidev informeeritus hetkeolukorra suhtes.

Analüüsi käigus selgitatakse välja liidestustarkvarale ja suhtlusteenusele rakenduvad funktsionaalsed ja mittefunktsionaalsed nõuded. Kuna suhtlusteenus peab vahendama liidestustarkvara abil infot Smartsi ja poe infosüsteemi vahel, siis uuritakse, kas suhtlusteenusel on veel lisaülesandeid lisaks liidestustarkvaras sätestatule.

Nõuete loomisel arvestatakse erinevate standarditega, et pakkuda võimalikult kvaliteetset ja arusaadavalt dokumenteeritud tarkvara. Kuna töös peab tegelema ka arvet ja maksmist puudutava teabega, siis leitakse sobiv lahendus masinloetava arve genereerimiseks.

Arenduse käigus implementeeritakse nõuetele vastav liidestustarkvara koos selle juurde kuuluva suhtlusteenusega. Valitakse sobivad tehnoloogiad, et realiseerida nõuetele vastav tarkvara. Arendamise etapis suheldakse aktiivselt meeskonna teise arendajaga, et tagada üle kogu projekti ühtne stiil ning lähtuda *Clean Code* põhimõtetest nii tarkvara klasside loomisel kui erinevate teekide kasutamisel [3]. Lisaks sellele leitakse sobiv mikroteenuste vaheline kommunikatsiooni viis.

Testimise faasis uuritakse, kuidas suhtlusteenus toimib koostöös teiste Smarts iseteenindussüsteemi teenustega ning kuidas liidestustarkvara abil on võimalik andmevahetus testpoega. Testimisel kasutatakse nii ühiktestimist, koodi läbivaatust kui ka vastuvõtutestimist, et tagada valminud tarkvara kvaliteeti. Lisaks sellele hinnatakse valitud meetodite, protsesside ja tehnoloogiate sobivust antud ülesande lahendamiseks.

Lõpetuseks tuuakse välja olulisemad järeldused antud tööst, mis ilmnesid töö käigus.

1.4 Ülevaade tööst

Töö teises peatükis käsitletakse Smarts iseteenindussüsteemi üldist tehnilist arhitektuuri ning selgitatakse täpsemalt, millisele osale keskendutakse selles magistritöös. Lisaks sellele tuuakse välja Smartsile sarnased teised lahendused, mis kas on juba turul saadaval või mis on peatselt turule tulemas.

Kolmandas peatükis leitakse nõuded, millele liidestustarkvara ja suhtlusteenus peavad vastama. Nõuete disainil lähtutakse põhimõttest, et nõuded peavad olema reaalsed ja testitavad.

Neljandas peatükis kirjeldatakse planeeritavate komponentide realisatsiooniga seotud teemasid sh pakutakse põhilise valdkonnamudeli disain.

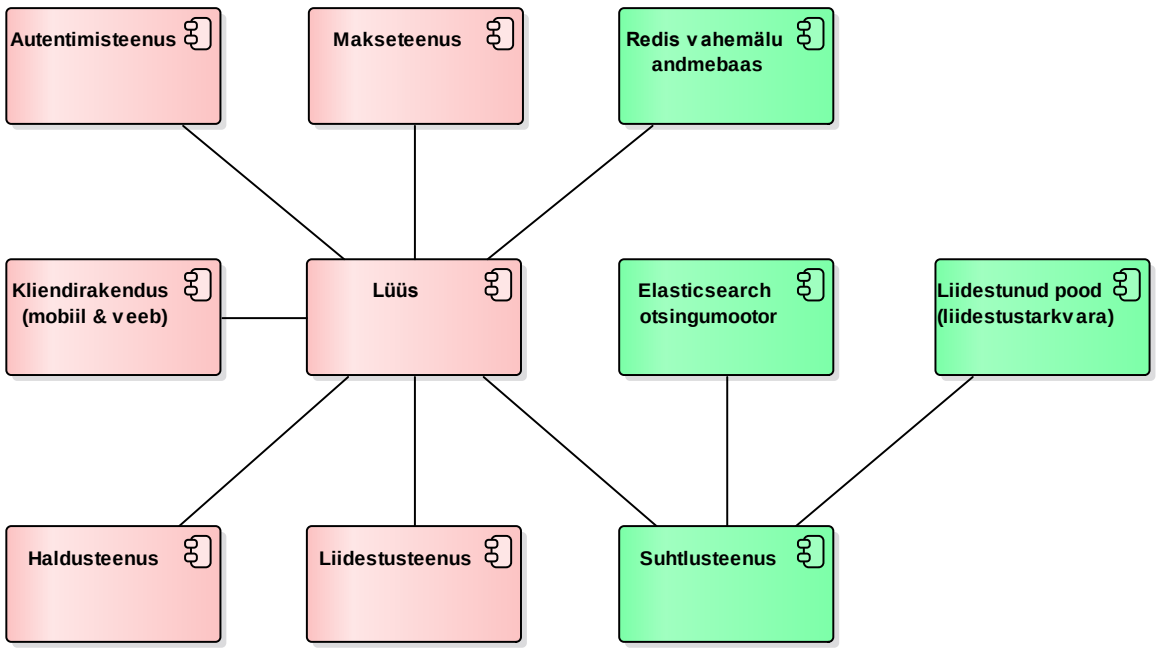
Viiendas peatükis analüüsitakse valminud süsteemi ning viiakse läbi testid. Selgitatakse, kas arendatud tarkvara vastab algselt püsitatud nõuetele ning tuuakse välja olulisemad töö käigus tekkinud järeldused.

2 Üldine arhitektuur koos planeeritavate komponentidega

Selles peatükis tutvustatakse Smarts iseteeninduslatvormi üldist arhitektuuri. Lisaks sellele näidatakse, kuidas suhestuvad selles töös rajatavad komponendid rajatava infosüsteemi üldisesse arhitektuuri.

Smarts iseteenindussüsteem põhineb mikroteenuste arhitektuuril. Mikroteenuste arhitektuur on valitud selleks, et võimaldada paremat skaleeritavust, vastutuste hajutatust eri osade vahel ja võimalust süsteemi kergemini tulevikus uuendada, testida ja monitoorida. Võrreldes monoliitarhitektuuriga võib mikroteenustel põhineva arhitektuur olla ka odavam ülal pidada [4]. Erinevad teenused on koondatud ühise lüüsi taha, mille poole on võimalik avaliku võrgu vahendusel pääseda.

Joonisel 1 on kujutatud Smarts iseteenindussüsteemi üldist arhitektuuri mikroteenuste tasemel. Kuna käesoleva magistritööga valmib paralleelselt veel teinegi magistritöö, mis käsitleb detailselt Smarts infosüsteemi arhitektuuri, on järgneval joonisel kujutatud rohelisega neid mikroteenuseid ja komponente, millega tegeletakse selles töös, ning roosaga neid, millega tegeletakse põhiliselt teises paralleelselt valmivas töös.



Joonis 1 Lihtsustatud arhitektuurijoonis

Järgnevas tabelis on kirjeldatud iga komponendi põhilist ülesannet ning vajadust Smarts infosüsteemis. Kuna antud töös tegeletakse põhiliselt liidestustarkvara ja suhtlusteenusega, siis ei ole ei ülal oleval joonisel 1 ega järgnevas tabelis 1 sügavuti kirjeldatud Smarts infosüsteemi teisi komponente.

Komponent	**Kirjeldus**	**Vajadus**
Kliendirakendus (mobiil & veeb)	Mobiilirakendus on mõeldud poe kliendile ning veebirakendus on mõeldud poele oma seadete haldamiseks Smartsis.	Võimaldab Smarts infosüsteemi selle klienditel (nii pood kui ka poe klient) kasutada.
Lüüs	Avalik ligipääsupunkt kogu süsteemile.	Tagab infosüsteemi teiste osade varjatuse avalikus võrgus.
Makseteenus	Tegeleb maksetega ning arvetega.	Võimaldab krüptomakseid ja seega tagab, et poe kliendi kontolt makstaks poe esitatud arve ostetud kauba kohta.
Haldusteenus	Võimaldab hallata ja jälgida infosüsteemis toimuvat liiklust.	Tuvastab lüüsist läbi minevaid võimalikke vigaseid päringuid.
Autentimisteenus	Tegeleb Smartsis kogu autoriseerimisega ning kasutajate loomisega.	Vastutab turvalisuse eest kasutades OAuth 2.0 protokolli. Valideerib nii teenuste kui kasutajate ligipääsusid.
Liidestusteenus	Võimaldab CRUD operatsioone poodidega seotud ühendusparameetrite asjus.	Võimaldab saada Smartsiga liidestunud poe andmeid, samuti vastutab uute poodide lisamise eest.
Suhtlusteenus	Võimaldab suhtluse poe ja Smarts infosüsteemi vahel.	Võimaldab suhtluse poe ja Smartsi vahel. Smartsi

	On veebiteenus, mille poole saab pöörduda HTTP protokolliga.	teised teenused saavad Suhtlusteenuse vahendusel küsida infot poe infosüsteemist.
Liidestunud pood (liidestustarkvara)	Komplekt abstraktseid klasse, mille pood oma infosüsteemis peab implementeerima.	Tagab poepoolse õige konfiguratsiooni suhtlemiseks Smartsiga.
Elasticsearch otsingumootor	Elasticsearch otsingumootor poe toodete puhverdamiseks.	Lühendab toote pärimiseks kuluvat aega ja kiirendab seega kasutaja jaoks süsteemi tööd, sest iga päringuga ei pea poe infosüsteemi poole pöörduma.
Redis vahemälu andmebaas	Redis NoSQL vahemälu andmebaas võimaldab ajutiselt infot vahemällu salvestada.	Võimaldab hoida pooleli olevat arvet vahemälus, et Suhtlus- ja Makseteenus arve kätte saaksid

Tabel 1 Komponentide ja mikroteenuste üldine kirjeldus

2.1 Teised iseteenindussüsteemid

Maailmas on käesoleval hetkel olemas mitmeid iseteenindussüsteeme. Osa neist on mõeldud töötama suurtes toidupoodides, osa ka väiksemates kauplustes. Laiemas plaanis võib väita, et iseteenindussüsteemid jagunevad kolmeks: iseteeninduspuldil põhinevad, kaalukontrollil põhinevad ja mobiilil põhinevad iseteenindussüsteemid. Neist lähemalt on juttu järgnevates alamosades.

2.1.1 mTasku ja Selveri iseteenindussüsteem

mTasku on Telia Eesti AS poolt arendatud süsteem, mis võimaldab süsteemi kasutajal hallata oma kliendikaarte ning maksta mobiili vahendusel pangaülekandega kauba müüjale või teenuse osutajale [5]. Kuna see võimaldab maksta mobiili vahendusel, siis

nimetatakse seda ka virtuaalseks rahakotiks. mTasku on universaalne makselahendus, millega võivad liituda eri kauplejad. Erinevalt poodide iseteenindussüsteemidest ei ole mTasku mõne poe või kaupluseketi põhine lahendus. Tegu on hetkel ainsa Eestis turul oleva mobiilimakseid võimaldava platvormiga.

Selveri iseteeninduskassat kasutades on võimalik maksta mTasku mobiilirakenduse abil. Tegu on iseteeninduspuldil põhineva süsteemiga, mille kasutamiseks on vajalik Partnerkaardi olemasolu [6]. Tooteid tuleb aga skaneerida Selveri iseteeninduspuldi abil. Ostmise protsess koosneb seega kolmest etapist: poodi sisenemisel arve avamine (mTasku rakendusega vastava QR-koodi skaneerimine või NFC kiibi lugemine), toodete skaneerimine Selveri iseteeninduspuldiga ja lõpuks kassas uuesti vastava QR-koodi või NFC kleebise skaneerimine ja seeläbi maksmisprotsessi alustamine. Maksmisel tuleb valida Selveri iseteeninduskassa seadme monitoril valik maksta mTasku rakenduse abil. Seepeale jõuab telefoni pärast edukat makset mKviitung, mis on digitaalsel kujul ostutšekk.

2.1.2 COOP Nutikassa

COOP Nutikassat on saadaval kahte varianti: iseteeninduspuldil põhinev ja toote kaalumisel põhinev kassa [7]. COOP iseteeninduskassas ei ole võimalik maksta mTasku rakendusega ega ka krüptovaluutaga. Ajakirja [Digi] toimetaja subjektiivsel arvamusel on COOP Nutikassa hetke parim iseteeninduskassa lahendus Eestis. Selle on tinginud hetke kõige kaasaegsem skaneerimispult koos lihtsa kasutajaliidesega [8]. Sarnaselt COOPi toote kaalumisel põhinevale Nutikassale töötavad ka Rimi, Maxima ja Prisma iseteeninduskassad.

2.1.3 Sam’s Club Scan & Go mobiilirakendus

Scan & Go on mobiilirakendus, mille abil on võimalik osta tooteid täielikult kassavabalt, st seda mobiilirakendust kasutades ei ole vajalik kassat kui poes asetsevat füüsilist seadet kasutada [9]. Selle asemel saab tooted skaneerida mobiilist ning makse tehakse samuti mobiili vahendusel. Osta saab kõiki tooteid peale alkoholi, tubakatoodete, ehete, ravimite ja kinkekaartide. Mobiilirakendus on kasutatav ainult USA territooriumil U.S clubs poeketis.

2.1.4 Eterbank krüptomakseplatvorm

Eterbank krüptomakseplatvorm on Smarts iseteenindusplatvormiga paralleelselt arenev krüptomakseplatvorm ettevõtetele, mis sarnaselt Smartsile on suunatud poodidele [10]. See võimaldab poe kliendil maksta ostetud toodete eest krüptorahas sedasi, et pood ise ei pea vastutama krüptoraha väärtuse võimaliku kõikumise eest ja pood saab seega kätte selle rahalise väärtuse, mida ta toote eest saada soovib. Erinevalt Smarts iseteenindussüsteemist ei võimalda Eterbank krüptomakseplatvorm selveteenust, st poe klient ei saa Eterbank rakenduse abil tooteid poes skaneerida nagu seda saab teha iseteeninduspulti kasutades.

2.1.5 Z Basket mobiilirakendus

Z Basket on Eestis välja töötatud iseteenindussüsteem, mis sarnaselt Smartsile on mõeldud kasutamiseks füüsilistes poodides [11]. Selle eelis eespool mainitud puldiga iseteenindussüsteemide ees seisneb võimaluses skaneerida tooteid mobiiliga ning maksta hiljem ostukorvi eest samuti mobiilis. Z Basketi erinevus võrreldes Smarts iseteenindusplatvormiga seisneb peamiselt makseviisis. Kui Smarts on keskendunud krüptomaksetele, siis Z Basekt võimaldab makseid tavavaluutas.

3 Planeeritava tarkvara nõuded

Järgnevates alapeatükkides tuuakse välja põhilised Smarts liidestustarkvara ja suhtlusteenuse funktsionaalsed ja mittefunktsionaalsed nõuded. Nõuete loomisel on arvesse võetud ISO/IEC 25010:2011 toote kvaliteedimudelit [12]. Kuna need nõuded on mõeldud eeskätt Smarts liidestustarkvara ja suhtlusteenuse loomist arvestades, siis ei ole siin kirjeldatud detailselt teisi komponente ja nendele rakenduvaid nõudeid.

Nõuete kirjeldamisel eristatakse erinevat tüüpi tegutsejaid:

- Pood – Smarts iseteenindusplatvormiga liitunud kaubandusettevõte
- Kasutaja – Smarts iseteenindusplatvormi Poes kasutav inimene ehk Poe klient, kes ostab Poest tooteid Smartsi vahendusel
- SCOM – Suhtlusteenus *(Smarts Communication)* – mikroteenus, mis vastutab Poega suhtlemise eest (liidestustarkvara vahendusel)
- SDK – Smarts SDK ehk liidestustarkvara – abstraktne komponent, mille Pood oma infosüsteemis implementeerib; võimaldab kahepoolset suhtlust Smarts iseteenindusplatvormi ja Poe infosüsteemi vahel
- Elasticsearch – Elasticsearch otsingumootor, kuhu salvestatakse Poest päritud tooteid
- Redis – vahemälu andmebaas, kuhu makseteenus salvestab genereeritud e-arve formaadis arve ning mida Suhtlusteenus seejärel sealt pärib
- Süsteem – Smarts iseteenindusplatvorm tervikuna koos selle all olevate mikroteenustega (mikroteenused on kirjeldatud joonisel 1)

3.1 Liidestustarkvara funktsionaalsed nõuded

Siin peatükis tuuakse välja põhilised Smarts SDK komponendi ehk liidestustarkvara funktsionaalsed nõuded. Nõuded on kirjeldatud kasutusjuhu tabelitena, kus iga nõude juurde on märgitud nõude nimetus, tegutsejad, eel- ja järeltingimused ning

põhistsenaarium. Liidestustarkvara funktsionaalsete nõuete kasutusjuhtude tunnus on SDKF *(Software Development Kit Functional)*.

Kasutusjuhu ID: **SDKF-1**	
Nimetus:	Toodet on võimalik otsida Poe infosüsteemist ribakoodi järgi
Tegutsejad:	Kasutaja, SCOM, SDK, Pood
Eeltingimused:	Poes eksisteerib otsitav toode
Järeltingimused:	Toote info jõuab SDK ja SCOMi vahendusel Kasutajani
Põhistsenaarium:	1. Kasutaja mobiililt tuleb info ribakoodi numbriga 2. SCOM teeb päringu poe tunnusele vastava Poe infosüsteemi ribakoodi alusel 3. Poe infosüsteem tagastab toote info SDK vahendusel SCOM teenusele, mis omakorda tagastab selle Kasutajale

Kasutusjuhu ID: **SDKF-2**	
Nimetus:	Toodet on võimalik otsida Poe infosüsteemist toote nime järgi
Tegutsejad:	Kasutaja, SCOM, SDK, Pood
Eeltingimused:	Poes eksisteerivad nimele vastavad tooted
Järeltingimused:	Toodete info jõuab SDK ja SCOMi vahendusel Kasutajani
Põhistsenaarium:	1. Kasutaja mobiililt tuleb info, mis sisaldab osaliselt või täielikult toote nime 2. SCOM teeb päringu poe tunnusele vastava Poe infosüsteemi toote nime alusel 3. Poe infosüsteem tagastab nimekirja nimele vastavatest toodetest

Kasutusjuhu ID: **SDKF-3**	
Nimetus:	Süsteemil on võimalik teavitada Poodi saabunud Kasutajast
Tegutsejad:	Pood, Süsteem, Kasutaja
Eeltingimused:	Kasutaja siseneb Poodi ja skaneerib vastava NFC kleebise või QR-koodi
Järeltingimused:	Poe süsteem on registreerinud Kasutaja, kes sisenes Poodi ja kasutab Smarts iseteenindusplatvormi (Süsteemi)
Põhistsenaarium:	1. Kasutaja siseneb Poodi ja skaneerib vastava NFC-kleebise või QR-koodi 2. SCOM saadab Poele SDK vahendusel andmed kasutaja kohta

Kasutusjuhu ID: **SDKF-4**	
Nimetus:	Poel on võimalik määrata Kasutajale maksmisel ostukontroll
Tegutsejad:	Pood, SDK, SCOM, Kasutaja
Eeltingimused:	Kasutaja siseneb Poodi ja skaneerib vastava NFC kleebise või QR-koodi
Järeltingimused:	Kasutaja ei saa maksta enne edukat ostukontrolli läbimist
Põhistsenaarium:	1. Süsteem teavitab Poodi saabunud kasutajast vastavalt kasutusjuhule SDKF-3 2. Pood määrab Süsteemi vahendusel Kasutajale ostukontrolli 3. Süsteem ei lase Kasutajal toodete eest maksta enne ostukontrolli läbimist 4. Pood (poe müüja) kontrollib Kasutaja ostetud tooteid a. Kui ostukontroll ei tuvasta puuduseid, siis suunatakse Kasutaja maksma b. Kui ostukontroll tuvastab puudused, palutakse kasutajal enne maksmist puuduolevad tooted skaneerida

Kasutusjuhu ID: **SDKF-5**	
Nimetus:	Pood saab failist laadida tooteid Elasticsearch repositooriumisse
Tegutsejad:	Pood, Süsteem
Eeltingimused:	Poel on Excel fail toodetega
Järeltingimused:	Tooted on failist lisatud Elasticsearch repositooriumisse
Põhistsenaarium:	1. Pood saadab faili toodetega Süsteemile 2. Süsteem valideerib faili vastavalt etteantud struktuurile a. Kui fail valideerub, lisab Süsteem tooted Elasticsearch repositooriumisse b. Kui fail ei valideeru, tagastab Süsteem Poele nimekirja veateadetega

3.2 Suhtlusteenuse funktsionaalsed nõuded

Siin peatükis tuuakse välja põhilised SCOM komponendi ehk suhtlusteenuse funktsionaalsed nõuded. Nõuded on kirjeldatud kasutusjuhu tabelitena, kus iga nõude juurde on märgitud nõude nimetus, tegutsejad, eel- ja järeltingimused ning põhistsenaarium. Suhtlusteenuse funktsionaalsete nõuete kasutusjuhtude tunnus on SCOMF *(Smarts Communication Functional)*.

Kasutusjuhu ID: **SCOMF-1**	
Nimetus:	Toodet on võimalik salvestada Elasticsearch repositooriumisse
Tegutsejad:	Kasutaja, SCOM, SDK, Elasticsearch, Pood
Eeltingimused:	Poes eksisteerib otsitav toode
Järeltingimused:	Toode on salvestatud Elasticsearch repositooriumisse
Põhistsenaarium:	Toote salvestamise Elasticsearch repositooriumisse algatab Kasutaja toote skaneerimisega või Pood pöördudes vastava päringuga SCOM poole. 1. Tuleb info toote lisamise kohta a. Kasutaja mobiililt tuleb info toote kohta (ribakood) b. Pood saadab päringu SCOM teenusele toote lisamiseks 2. SCOM küsib toote info vastavalt kasutusjuhule SDKF-1 3. Pood tagastab SCOM teenusele toote info 4. SCOM salvestab toote info Elasticsearch repositooriumisse

Kasutusjuhu ID: **SCOMF-2**	
Nimetus:	Poel on võimalik toode Elasticsearch repositooriumist kustutada
Tegutsejad:	SCOM, SDK, Elasticsearch, Pood
Eeltingimused:	Elasticsearch repositooriumis eksisteerib ribakoodile ja poe tunnusele vastav toode
Järeltingimused:	Elasticsearch repositooriumis ei eksisteeri ribakoodile ja poe tunnusele vastavat toodet
Põhistsenaarium:	Toote kustutamise Elasticsearch repositooriumis algatab Pood 1. Pood teeb SDK vahendusel päringu SCOM teenuse poole andes kaasa kustutatava toote ribakoodi ja poe ID 2. SCOM kustutab repositooriumist toote ja tagastab Poele teate a. Teade on „*success*“ kui kustutamine oli edukas b. Teade on „*Product not found*“ kui toodet ei leitud c. Tõrke korral püstitatakse vastav erind

Kasutusjuhu ID: **SCOMF-3**	
Nimetus:	Poel on võimalik toote infot Elasticsearch repositooriumis uuendada
Tegutsejad:	SCOM, SDK, Elasticsearch, Pood
Eeltingimused:	Poes eksisteerib soovitud ribakoodiga toode; Elasticsearch repositooriumis eksisteerib sama ribakoodiga antud Poele vastav toode
Järeltingimused:	Sama ribakoodiga toote info on Elasticsearch repositooriumis muudetud
Põhistsenaarium:	Toote muutmise Elasticsearch repositooriumis initsieerib Poe infosüsteem 1. Pood alustab toote info muutmist vastava päringuga SCOM poole andes kaasa muudetava toote uue kuju 2. SCOM kustutab toote Elasticsearch repositooriumist sarnaselt kasutusjuhus SCOMF-2 kirjeldatule 3. SCOM lisab toote uuesti Elasticsearch repositooriumisse sarnaselt kasutusjuhus SCOMF-1 kirjeldatule

	Juhul kui toodet kustutamise Elasticsearch repositooriumist ei leita, siis lisatakse toode Elasticsearch repositooriumisse vastavalt kasutusjuhule SCOMF-1

Kasutusjuhu ID: **SCOMF-4**	
Nimetus:	Arvet on võimalik pärida Redis vahemälu andmebaasist
Tegutsejad:	SCOM, Redis
Eeltingimused:	Redis vahemälus eksisteerib otsitav arve
Järeltingimused:	SCOM teenus saab vahemälust leitud arve
Põhistsenaarium:	1. SCOM pöördub Redis vahemälu poole arve ID alusel 2. Redis vahemälu tagastab arve ID alusel objekti, kuhu kuulub arve koos poe ühenduse infoga

Kasutusjuhu ID: **SCOMF-5**	
Nimetus:	Muudetud arvet on võimalik salvestada Redis vahemälu andmebaasi
Tegutsejad:	SCOM, Redis
Eeltingimused:	Redis vahemälus eksisteerib arve, mille ID on X
Järeltingimused:	Redis vahemälus olev arve ID-ga X on muudetud
Põhistsenaarium:	1. SCOM pärib arve Redis vahemälust vastavalt kasutusjuhule SCOMF-4 2. SCOM muudab arvet vastavalt kasutusjuhtule SCOMF-6 3. SCOM salvestab arve selle sama arve ID alusel Redis vahemälu andmebaasi

Kasutusjuhu ID: **SCOMF-6**	
Nimetus:	Toodet on võimalik lisada Kasutaja pooleliolevale (veel mitte tasutud) arvele
Tegutsejad:	Kasutaja, SCOM, Redis
Eeltingimused:	Eksisteerib Kasutaja, kellel on pooleliolev arve. Eksisteerib Kasutaja valitud toode, mida ei kajastu pooleliolevas arves
Järeltingimused:	Toode on arvele lisatud
Põhistsenaarium:	1. SCOM pärib Redis vahemälust arve vastavalt kasutusjuhule SCOM-4 2. SCOM lisab toote arvele a. Toode lisatakse gruppi *default*, kui tootel puudub grupi tunnus (e-arve formaadis `InvoiceItemGroup`) b. Toode lisatakse gruppi vastavalt toote grupi tunnusele 3. SCOM salvestab arve Redis vahemällu vastavalt kasutusjuhule SCOMF-5

Kasutusjuhu ID: **SCOMF-7**	
Nimetus:	Süsteemil on võimalik saata Poele lõplik koostatud arve
Tegutsejad:	Kasutaja, SCOM, SDK, Pood
Eeltingimused:	Kasutaja on ostnud Poest tooteid ja nende eest edukalt maksnud
Järeltingimused:	Pood saab SCOM teenuselt Kasutaja makstud arve
Põhistsenaarium:	1. Kasutaja maksab Süsteemi vahendusel toodete eest 2. Süsteem töötleb makset ja genereerib lõpliku arve, mis vastab e-arve formaadile 3. SCOM saadab arve poele, mille Pood saab SDK vahendusel avada (SDK sisaldab e-arve komponente)

3.3 Liidestustarkvara ja suhtlusteenuse mittefunktsionaalsed nõuded

Siin peatükis tuuakse välja põhilised Smarts SDK komponendi ehk liidestustarkvara ja SCOM ehk suhtlusteenuse mittefunktsionaalsed nõuded. Nõuded on kirjeldatud kasutusjuhu tabelitena, kus iga nõude juurde on märgitud nõude nimetus, tegutsejad, eel- ja järeltingimused ning põhistsenaarium. Mittefunktsionaalsed nõuded ei ole rajatud ühe

komponendi põhiselt (suhtlusteenus või liidestustarkvara) ja seetõttu on siin kasutusjuhu id SSCNF *(Smarts SDK-Communication Non Functional)*.

Kasutusjuhu ID: **SSCNF-1**	
Nimetus:	Süsteem peab võimaldama mitmekeelsust vastavalt ISO 639-1:2002 standardile
Tegutsejad:	Süsteem, Pood, Kasutaja
Eeltingimused:	Poel on soov näidata toote infot erinevates keeltes
Järeltingimused:	SDK komponent vahendab toote infot Süsteemile mitmes keeles
Põhistsenaarium:	1. Kasutaja otsib toodet 2. Süsteem kuvab toote ja pärib selle vajadusel Poelt 3. Kasutaja vahetab Süsteemi keelt 4. Süsteem kuvab toote info teises keeles a. Süsteem kuvab toote info teises keeles kui Pood tagastab toote info selles keeles b. Kui Pood ei tagasta toote infot soovitud keeles, kuvab Süsteem toote info vaikimisi keeles

Kasutusjuhu ID: **SSCNF-2**	
Nimetus:	Suhtlusteenus peab toote infot küsima eeskätt Elasticsearch repositooriumist ja alles seejärel Poe infosüsteemist
Tegutsejad:	SCOM, Elasticsearch, Pood, Kasutaja
Eeltingimused:	Kasutaja soovib saada toote infot
Järeltingimused:	Süsteem kuvab kasutajale toote info
Põhistsenaarium:	1. Kasutaja skaneerib toote ribakoodi 2. SCOM otsib toote infot Elasticsearch otsingumootorist a. Elasticsearch tagastab toote info 1. SCOM tagastab toote info Kasutajale b. Toodet ei eksisteeri Elasticsearch otsingumootoris 1. SCOM pärib toote info Poe infosüsteemist vastavalt kasutusjuhule SDKF-1

Kasutusjuhu ID: **SSCNF-3**	
Nimetus:	Süsteem ei pöördu toote küsimiseks poe poole kui toote info on salvestatud Elasticsearch repositooriumisse
Tegutsejad:	SCOM, Elasticsearch, Kasutaja
Eeltingimused:	Kasutaja soovib saada toote infot. Toote info on salvestatud Elasticsearch repositooriumisse.
Järeltingimused:	Süsteem kuvab kasutajale toote info
Põhistsenaarium:	1. Kasutaja otsib toodet. 2. SCOM pöördub Elasticsearch otsingumootori poole 3. Otsingumootor tagastab SCOM teenusele toote info 4. SCOM tagastab toote info Kasutajale

Kasutusjuhu ID: **SSCNF-4**	
Nimetus:	Kriitiline info (kontaktandmed, poe ühenduse info) peab olema krüpteeritud kujul (krüpteeritud avaliku võtme krüptograafia algoritmiga)
Tegutsejad:	Süsteem, Pood
Eeltingimused:	Eksisteerib kriitiline info
Järeltingimused:	Kriitiline info on krüpteeritud

4 Planeeritavate komponentide realisatsioon

Selles peatükis leitakse sobiv realisatsioon planeeritavale liidestustarkvarale ja suhtlusteenusele. Seega kirjeldatakse liidestustarkvara tehnilist disaini, tuuakse välja eri komponendid suhtlusteenuses kasutamiseks, kirjeldatakse suhtlusteenuses võimaldatud protsesse ning näidatakse, mis tehnoloogiaid kasutades planeeritav tarkvara programmeeritakse.

Planeeritav tarkvara realiseeritakse ühe mikroteenuse ja ühe SDK *(Software development kit)* komponendi näol. SDK ehk tarkvaraarenduse komplekt sisaldab hulka vajalikke dokumenteeritud tarkvaraklasse koos vajalike teekide ning dokumentatsiooniga, et võimaldada selle kasutajatel arendada tarkvara, mida saaks hiljem kasutada Smarts iseteenindussüsteemiga suhtlemiseks [13]. Lisaks mikroteenusele ja SDK komponendile võimaldatakse otsingumootori ja vahemälu andmebaasi kasutamine.

4.1 Liidestustarkvara ülesehitus

Liidestustarkvara ehk Smarts SDK on üks Smarts süsteemi alamkomponentidest. Selle ülesandeks on tagada poe infosüsteemi valmisolek suhtlemiseks Smartsi infosüsteemiga. Liidestustarkvara vahendab poelt saadud tooteinfot koos toote hinnainfoga.

Liidestustarkvara disainimisel on arvestatud mitmete standarditega. Standardiseeritud lähenemine on vajalik, et poodide jaoks oleks liidestumine Smarts süsteemiga arusaadavam ja kergem. Lisaks lihtsustab standardiseeritud lähenemine erinevate osapoolte (Smarts ja liidestuv pood) vahelist kommunikatsiooni.

4.1.1 Eesti e-arve formaat

E-arve formaat loodi Pangaliidu poolt 2014 aastal, et muuta arvelduprotsessi tänapäevasemaks ja efektiivsemaks. E-arve on XML vormingus dokument, mis on seega nii inim- kui ka masinloetav [14]. See koosneb kolmest põhilisest osast: päis failiinfo jaoks, arve arveinfo jaoks ning jalus esitatud arvete ja kogusumma jaoks. Smarts iseteenindussüsteem kasutab e-arve formaati arveinfo töötlemisel. See tähendab, et Smarts platvormi ja platvormiga liidestunud poe vahel liigub info, mille struktuur lähtub e-arve formaadistx. E-arve formaadile vastavuse osas lähtutakse minimaalse e-arve vajadustest.

4.1.2 ISO 639-1 standard

ISO 639-1 standard pärineb aastast 2002. See defineerib kahetähelise lühendi, mis viitab maailmas populaarsematele keeltele [15]. Samast standardist on olemas kokku 5 erinevat versiooni, mis erinevad järjenumbri poolest. Seega lisaks ISO 639-1:2002 veel ISO 639-2:1998, ISO 639-3:2007, ISO 639-4:2010, ISO 639-5:2009 [16]. Põhjus, miks antud töös valiti kasutamiseks ISO 639 standardist versioon 1, seisneb e-arve formaadis, mis sisemiselt arve keele valikus toetub samuti ISO 639-1 standardile.

4.1.3 Poe toote valdkonnamudel

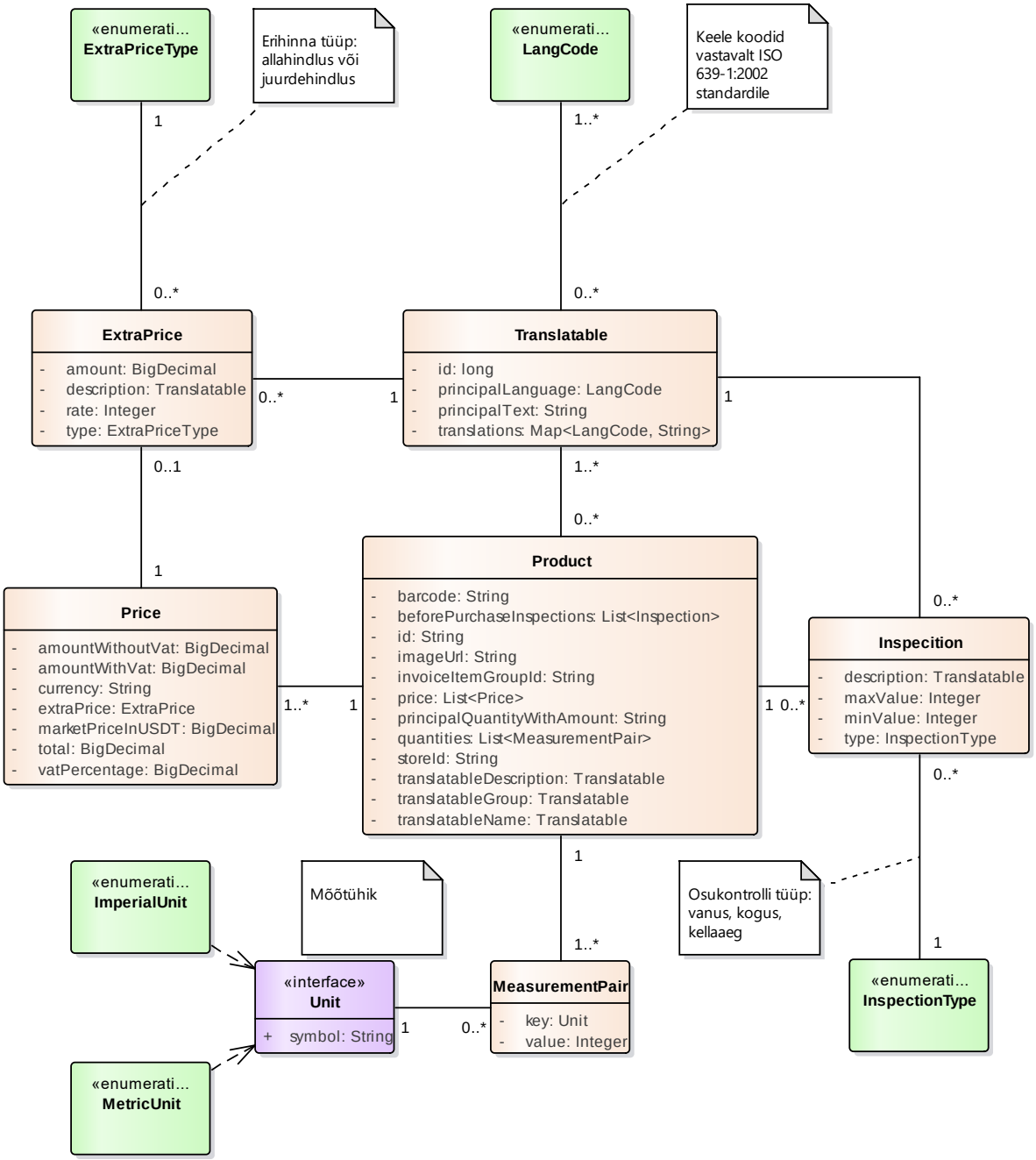
Poe toote valdkonnamudeli disainimisel on lähtutud minimaalsest e-arve formaadist ehk minimaalselt nõutud väljadest e-arve struktuuris. Lisaks on arvesse võetud GRASP mustreid, mis kirjeldavad põhilisi objektorienteeritud programmeerimise juurde kuuluvaid disainimustreid [17]. GRASP mustritest on valdkonnamudeli loomisel arvestatud eelkõige eksperdi kõrge kokkukuuluvuse mustrite ideedega. Järgneval joonisel 2 on näha SDK komponendis kasutatav poe toote valdkonnamudel, mida kasutatakse poe infosüsteemist toote pärimiseks. Tabelis 2 on kirjeldatud valdkonnamudeli erinevaid komponente ja põhjendatud nende vajalikkust. Minimaalse e-arve formaat on toodud töö lisas 1.

Andmetüüp	**Vajadus**
Product	Toode – põhiobjekt, mis sisaldab kõiki teisi joonisel 2 ja siin tabelis olevaid andmetüüpe. Koondab vajaliku toote info, mille põhjal kuvab Smarts selle kasutajale ostetud toodete nimekirja ja koostab arve.
Price	Sisaldab toote põhihinna infot koos käibemaksu infoga. Lisaks on võimalus lisada erihinda (soodustus või juurdehindlus). Arve koostamisel moodustatakse *Price* objekti põhjal e-arve formaadis tuntud *VatRecord* objekt koos *ItemEntry* juurde kuuluva hinnainfoga.

ExtraPrice	Erihinna määramise objekt, mis võimaldab määrata erihinda numbrilise suurusena *(amount)* ja protsendilise sõltuvusena põhihinnast *(rate)*. Väli *type* viitab erihinna tüübile.
ExtraPriceType	Enum, mis määrab erihinna tüüpi. Omab kahte väärtust: DSC ja CHR, mis tulenevad e-arve formaadist. DSC tähendab soodustust (erihind on väiksem kui põhihind) ja CHR tähendab juurdehindlust (erihind on suurem kui põhihind).
Translatable	Universaalne tõlkekomponent, mis sisaldab ühte põhikeelt *(principalLanguage)* ja ühte põhiteksti *(principalText)*. Põhikeel ja põhitekst on ühtlasi vaikimisi keeleks ja tekstiks, st juhul, kui on vaja toote nime või kirjeldust, aga keelt ei ole täpsustatud, siis kuvatakse põhikeeles olevat põhiteksti. Tõlked on eraldi *Map* andmestruktuuris, kus võtmeks on keele kood *(LangCode)* ja väärtuseks keele koodile vastavas keeles olev tekst.
LangCode	Enum, mis sisaldab ISO 639-1:2002 standardile vastavalt keelte kahetähelisi koode. Lisaks on võimalik pärida keele nimetusele (näiteks *English, Estonian, Russian*) vastavalt keele koodi.
Inspection	Ostukontrolli objekt, mis võimaldab poel määrata, kas antud tootega on nõutud ostukontroll või mitte. *MinValue* ja *maxValue* on arvulised suurused, millega saab määrata ostukontrolliga kaasnevat alam- ja ülempiiri. Ostukontrolli tüüp (vanuse põhine, kellaaja põhine, koguse põhine) määrab väli *type*. Tootel võib olla korraga määratud mitu ostukontrolli.
InspectionType	Enum, mis määrab ostukontrolli tüübi. Sisaldab kolme liiki eeldefineeritud väärtuseid vanuse, koguse ja kellaaegade kohta.
MeasurementPair	Toote mõõteobjekt, mis oma olemuselt on võti-väärtus tüüpi objekt. Võtmeks on mõõtühik ja väärtuseks mõõtühikule vastav arvuline väärtus. Kuna mõõteviise võib ühel tootel olla mitu, on võimalus kirjeldada massiivis ka mitut mõõteobjekti. Toote juures

	arvestatakse, et massiivis esimene mõõteobjekt sisaldab toote peamist mõõtühikut koos väärtusega.
Unit	Liides, mis võimaldab kirjeldada mõõtühikuid. Defineerib nõude, et igal mõõtühikul peab olema sümbol. Põhjus liidese kasutamiseks seisneb vajaduses tulevikus mõõtühikute valikut suurendada. Seetõttu peavad tulevikus defineeritud mõõtühikute kirjeldused vastama selle liidese nõuetele.
MetricUnit	*Unit* liidese realisatsioon enum kujul, kus on kirjeldatud meetermõõdustikus mõõtühikuid.
ImperialUnit	*Unit* liidese realisatsioon enum kujul, kus on kirjeldatud imperiaalseid mõõtühikuid.

Tabel 2 Toote andmeobjekti andmetüüpide kirjeldus



Joonis 2 Toote valdkonnamudel

Nagu ülal olnud tabelist ja jooniselt võis näha, peab toote küsimise andmeobjekt oskama arvestada mitmete eluliste tõsiasjadega: seadusandlusest tulenevad piirangud (vanusepiirangud, ajalised piirangud), kaupluse kehtestatud piirangud (koguse limiidid näiteks suurte allahindluste ajal), vajadus eri (krüpto)valuutade järele kui ka võimalus toodet mitmes eri mõõtühikus mõõta. Selle info põhjal oskab Smarts kontrollida, kas toote ostmine antud kasutajale on lubatud. Vajadusel pöördutakse müüja poole kontrollimaks ostja vanust füüsilise dokumendikontrolliga.

4.2 Elasticsearch otsingumootor

Elasticsearch on vabavaraline otsingutarkvara (otsingumootor), mis baseerub Apache Lucenel. Seda tarkvara saab kasutada ka dokumendihoidlana (andmebaasina), mis võimaldab kiiret tekstipõhist otsingut [18]. Arvestades Elasticsearchi kiirusomadusi (madal latentsus - lisatud dokument on otsitav umbes 1 sekund pärast lisamist), on see Smarts iseteenindusplatvormi kontekstis mõistlik kasutada, sest poe poolt lisatud toode on otsingumootoris praktiliselt kohe saadaval. Lähtudes DB-Engines otsingumootorite edetabelist on Elasticsearch käesoleval töö kirjutamise hetkel tabelis esikohal. DB-Engines edetabel on koostatud populaarsushinnanguid arvestades (sh otsingumootori nime esinemine foorumites, tööpakkumistes) [19]. Lisaks eelmainitule pakub Elasticsearch koos Kibanaga võimalust lihtsalt otsingumootorit (andmebaasi) monitoorida [20]. Olgu öeldud, et võimalik monitooringulahendus ei kuulu selle töö skoopi ja on mõeldud realiseerimiseks tulevikus.

Elasticsearchi kasutamisel on vaja esmalt selgitada põhilisi Elasticsearchiga seotud termineid. Termineid ja nende seletusi kirjeldab alljärgnev tabel 3, mis põhineb Elasticsearch 6.2 dokumentatsioonil (veerud „Termin“ ja „Seletus“) [21]. Tabeli veerg „Kasutamine“ kirjeldab antud termini kastutamist Smarts iseteenindusplatvormi kontekstis.

Termin	**Seletus**	**Kasutamine**
Klaster *(cluster)*	Klaster koondab endas ühte või mitut sõlme. Igal klastril peab olema unikaalne nimi, sest nime järgi ühenduvad sõlmed klastrisse. Klaster on vajalik otsingute tegemisel, sest võimaldab paralleelotsingut *(federated search)* üle klastris olevate sõlmede, kus iga klaster saadab päringu igale sõlmele ning ühendab seejärel sõlmedelt saadud vastuse üheks tulemuseks [22].	Elasticsearch konfigureerimisel on toodete salvestamiseks loodud klaster nimega *„smarts-products“*. Kui tulevikus tekib soov kasutada Elasticsearchi veel millegi salvestamiseks, tuleb selleks luua eraldi oma klaster.

Sõlm *(node)*	Sõlm moodustab ühe sisemise osa klastrist. Klastris võib olla üks või mitu sõlme. Sõlm on server, mis tegeleb andmetöötlusoperatsioonidega. Sõlm peab olema viidatud nime abil õigele klastrile.	Klaster sisaldab ühte sõlme. Sõlmel võib olla ka oma nimi, kuid antud juhul on selleks vaikimisi UUID. Sõlmede lisandumisel on selguse huvides vajalik sõlmedele nimed anda.
Indeks *(index)*	Indeks koondab endas hulka sarnaste omadustega dokumente. Indeks on määratud nime järgi, mis tuleb indeksile selle defineerimisel anda. Indeksi nime kasutab Elasticsearch hiljem vastavate dokumentide indekseerimisel, otsimisel, lisamisel, muutmisel ja kustutamisel.	Klastris *„smarts-produkts“* on loodud indeks nimega *„product-nested“*, mis viitab (poe) tootele. Nimes on lisatud *„-nested“* näitamaks, et tegu on sellist tüüpi dokumendiga, mis sisaldab lisaks lihttüüpidele ka komplekstüüpi välju.
Dokument *(document)*	JSON-formaadis andmehulk, mida saab indekseerida, otsida, lisada, muuta ja kustutada.	Dokument on antud juhul üks toode, mis kuulub indeksi *„smarts-nested“* alla ja on omakorda *„smarts-products“* klastris.
Kild *(shard)*	Kuna indeks koos oma dokumentidega võib sisaldada niivõrd suurel hulgal andmeid, et need ei mahu füüsiliselt ühe sõlme (serveri) peale ära (andmemahupiirang tuleb ette või	Indeksi *„product-nested“* loomisel määrati loodavate kildude arvuks

	muutub otsimine ebaotstarbekalt aeglaseks), siis on võimalus seda indeksit jagada väiksemateks osadeks – kildudeks. Iga kild on oma omadustelt sarnane indeksile ja on võimeline töötama suvalises sõlmes. See annab võimaluse hajutada andmeoperatsioone eri sõlmede vahel ja muuta tööd paralleelsemaks ning tagada sellega süsteemi suurem läbilaskevõime.	vaikimisi pakutud 5, sest hetkel arendatakse alles Smartsist MVP versiooni, kus on kasutusel arenduskeskkonna riistvara ning taolise seadistuse saab jätta hilisemaks.
Koopia *(replica)*	Koopia tähendab koopiat killust *(shard replica)*. See on vajalik, et tagada süsteemi töö juhul, kui mõni sõlm (koos killuga) peaks mingil põhjusel vigaseks muutuma või peatuma. Sel juhul käivitatakse koopia abil selles killus olev info mõnes teises sõlmes. Koopiat ei hoita koopiale vastava killuga kunagi samas sõlmes.	Indeksi *„product-nested“* loomisel määrati loodavate koopiate arvuks vaikimisi pakutud 1, sest hetkel arendatakse alles Smartsist MVP versiooni, kus on kasutusel arenduskeskkonna riistvara ning taolise seadistuse saab jätta hilisemaks.

Tabel 3 Elasticsearch otsingumootori terminid ja seadistus

Elasticsearch otsingumootori kasuks otsustati peamiselt selle kasutamise lihtsuse tõttu. Elasticsearch otsingumootor on lihtsalt seadistatav ja arendaja jaoks on selle kasutamine lihtne – päringuid on võimalik teha lisaks päringukeele kasutamisele ka POJO *(Plain Old Java Object)* väljade põhjal.

4.3 Toote info pärimise ja arve uuendamise protsess

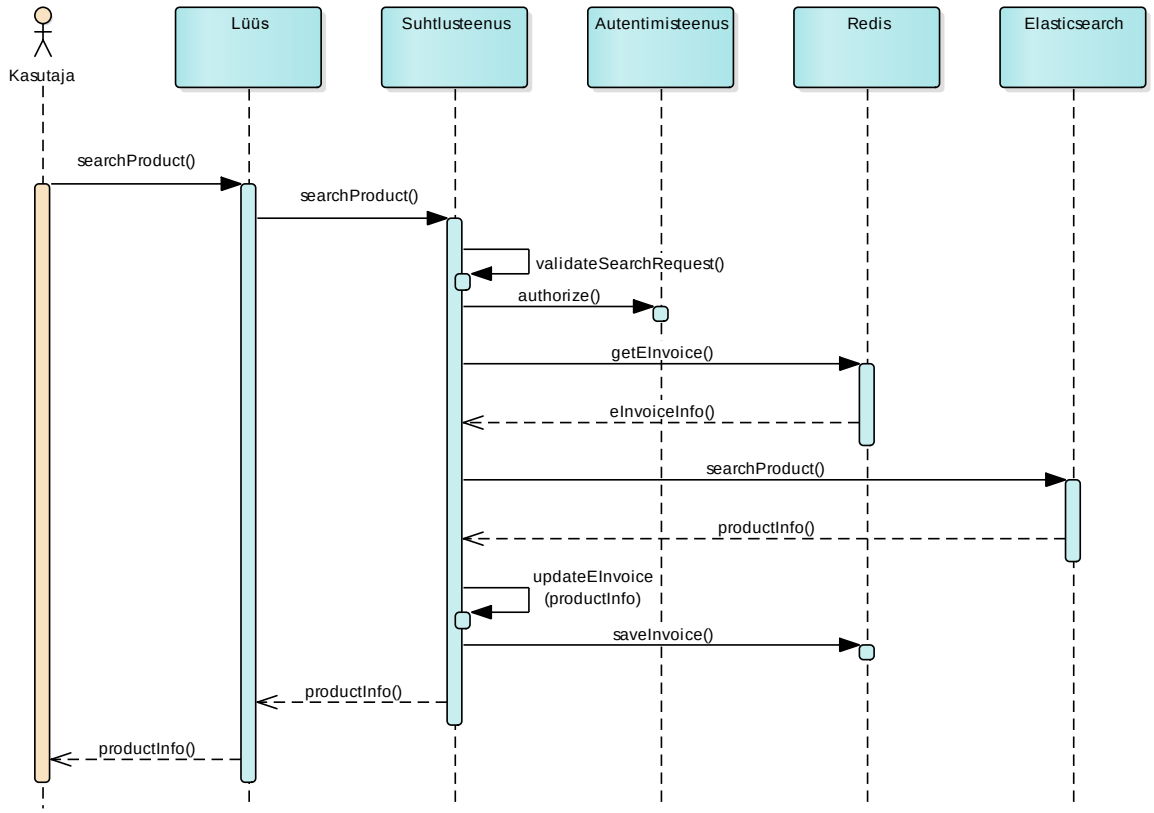
Toote küsimise protsess on üks Smartsi kesksemaid tegevusi. Ilma poelt toodete kohta infot saamata, ei ole võimalik Smartsi iseteenindussüsteemil sellele pandud rolli täita. Selles osas välja toodud interaktsioonidiagrammid kirjeldavad toote küsimise protsessi ja selle eri variante.

Joonis 3 ja joonis 4 visualiseerivad toote info pärimise protsessi. Jadadiagrammidel on kirjeldatud erinevad eluteed *(lifeline)* ja tegijad *(actor)* [23]:

- Kasutaja – Smarts infosüsteemi kasutaja
- Lüüs – mikroteenus, mis on ainsana avalikus võrgus kättesaadav ja mis suhtleb kasutajaga
- Suhtlusteenus (SCOM) – mikroteenus, mis vastutab poe ning otsingumootoriga suhtlemise eest
- Autentimisteenus – mikroteenus, mis vastutab turvalisuse eest ja jagab ligipääsulubasid
- Redis – vahemälu funktsionaalsust pakkuv teenus, mis võimaldab pooleli olevat arvet hoida ajutiselt teatud aja vahemälus ning seda sealt kiirelt küsida [24]
- Elasticsearch – otsingumootor, mis salvestab poe infosüsteemist päritud tooteid
- Pood (liidestustarkvara vahendusel) – poe enda infosüsteem, mis on liidestunud Smarts iseteenindusplatvormiga

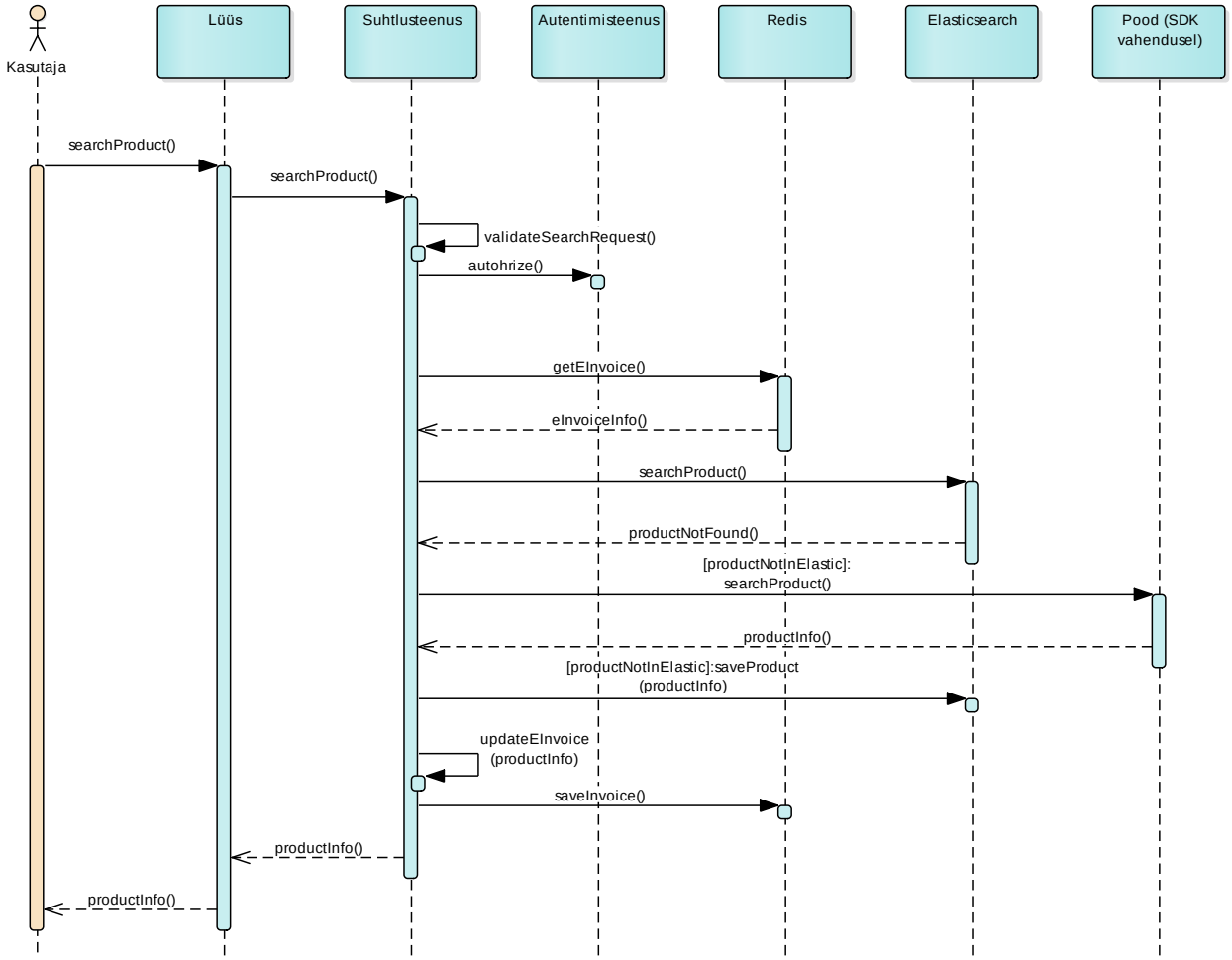
Lisaks toote info pärimise protsessile on joonistel 3 ja 4 näidatud ka suhtlusteenuse tööd arvega. Kui kasutaja otsib toodet (ehk skaneerib ribakoodi ja soovib seega toodet osta), siis küsib suhtlusteenus Redis vahemälust arve id alusel hetkel pooleli oleva arve. Seejärel olles arve saanud uuendab suhtlusteenus arvet lisades sinna päritud toote. Pärast toote lisamist salvestab suhtlusteenus arve selle sama id alusel uuesti Redis vahemällu. Seejärel saab makseteenus sarnaselt vahemälust arve küsida, et sellega edasisi vajalikke toiminguid teha (maksmisele suunamine jm).

Joonis 3 kirjeldab protsessi, mis sätestab süsteemi mittevajaduse pöörduda poe infosüsteemi poole kui otsitava toote info on kättesaadav ka Elasticsearch otsingumootorist.



Joonis 3 Toote info küsimise interaktsioonidiagramm ilma poe poole pöördumiseta

Järgneval joonisel 4 on näidatud protsessi, mis sätestab tingimuse, et toote info pärimisel pöördub Smarts esmalt Elasticsearch otsingumootori poole ja juhul, kui sealt toodet ei leita, siis teeb päringu poe infosüsteemi poole.

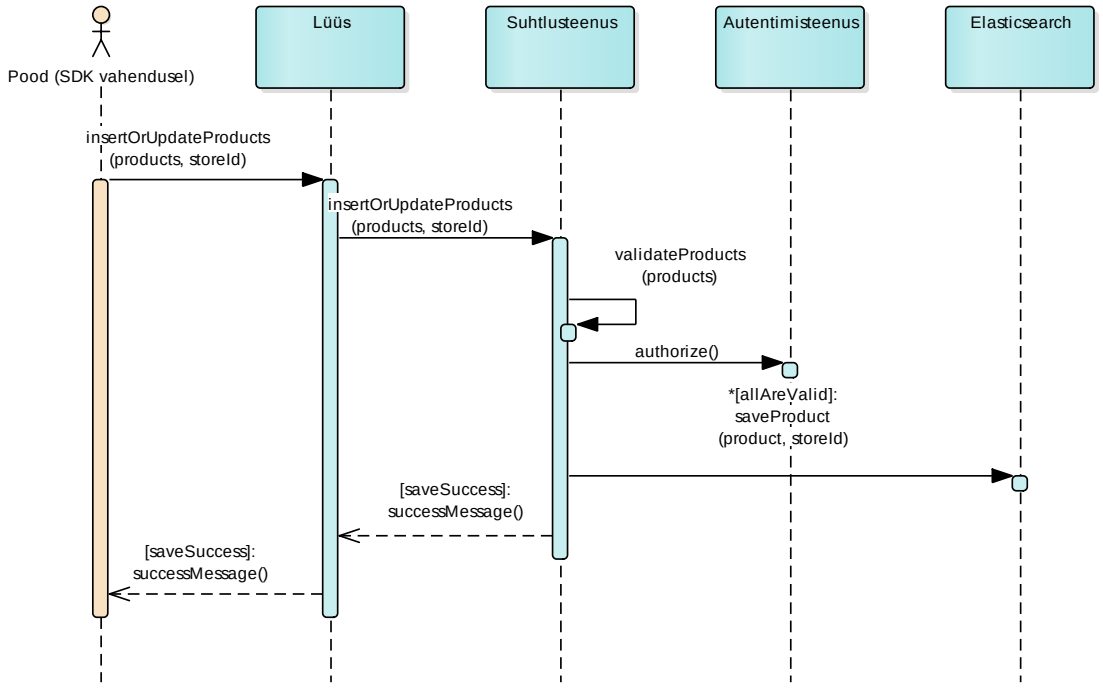


Joonis 4 Toote info küsimise interaktsioonidiagramm koos poe poole pöördumisega

4.4 Poe tegevused tootega Elasticsearch otsingumootoris

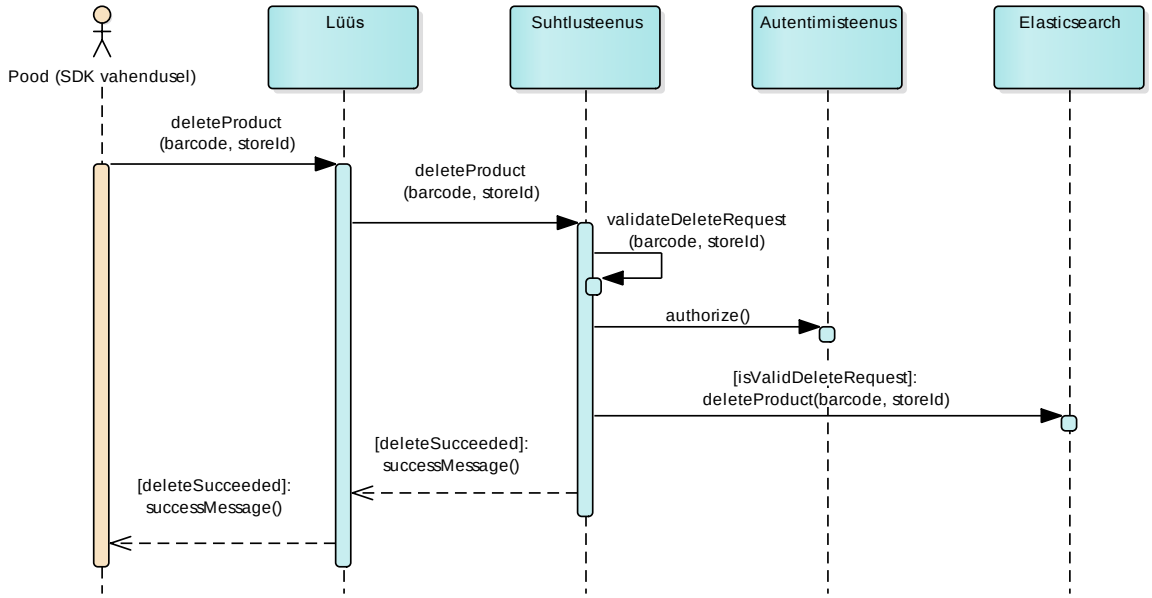
Poel (Smarts liidestustarkvara vahendusel) on võimalus oma tooteid Smartsi otsingumootoris muuta, neid sinna lisada või neid sealt kustutada. See tegevus on vajalik võimaldamaks poele teha hinnakorrektsioone, võtta toode vajadusel müügist maha või lisada müüki uus toode. Elasticsearch otsingumootorit kasutatakse toote ostu protsessi kiirendamiseks, sest kui päritakse otsingumootori andmebaasis olevat toodet, ei pea enam poe infosüsteemi suunas päringut tegema. Kuna Elasticsearch otsingumootor on üks Smarts infosüsteemi osadest, siis peab poel olema võimalus seal olevat infot vajadusel muuta. Järgneval joonisel 5 on näidatud poepoolne toote lisamise või uuendamise protsess. Pood annab Smartsile ette nimekirja tooteid. Smarts valideerib nimekirja ning lisab seejärel tooted otsingumootori andmebaasi poe tunnuse ja ribakoodi alusel. Need

tooted, mida poe tunnuse ja ribakoodi järgi otsingumootorist ei leita, lisatakse sinna ja need, mis leitakse, kustutatakse ja lisatakse seejärel uuesti (ehk uuendatakse).



Joonis 5 Toote lisamine või toote andmete uuendamine otsingumootori andmebaasis

Järgnev joonis 6 kirjeldab toote kustutamist otsingumootori andmebaasist. Erinevus toote lisamise ja uuendamise vahel seisneb selles, et korraga saab kustutada ainult ühte toodet. Kustutamisel valideeritakse samuti poelt tulnud päringu sisu ning pärast kustutamise operatsiooni tagastatakse sõnum toote kustutamise kohta.



Joonis 6 Toote kustutamine otsingumootori andmebaasist

4.5 Komponentide turvalisus

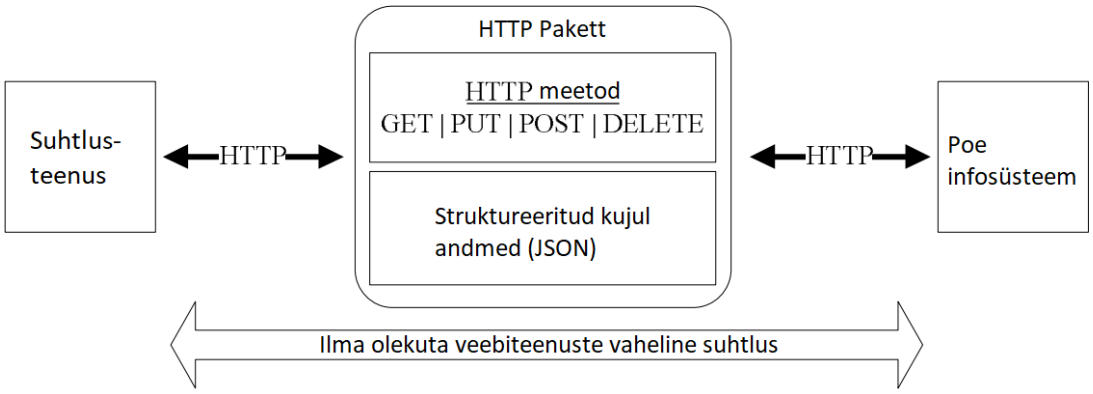
Suhtlusteenuse ja liidestustarkvara turvalisust tagatakse mitmel viisil. Esiteks on suhtlusteenuse poole pöörduda ainult läbi lüüsi. See tähendab et suhtlusteenus koos teiste mikroteenustega on kättesaadavad ainult sisevõrgus, mis on tulemüüriga kaitstud.

Teiseks kasutatakse transpordikihis SSL *(Secure Sockets Layer)* krüpteeringut, mis tagab võrgupakettide krüpteerituse transpordikihis (HTTPS). Sellega välditakse võimalikku pealtkuulamist andmevahetusel avalikus võrgus (poe ja Smartsi vahel).

Kolmandaks on teenustevahelistes suhetes kasutusel ligipääsuload *(access token)*, mida süsteemis ainsana väljastab autentimisteenus. Kõik teised teenused peavad ligipääsuloa valideerimiseks pöörduma autentimisteenus poole. Rakenduse tasemel on turvalisus tagatud OAuth 2.0 protokolliga, mis põhineb ligipääsulubade jagamisel. Lisaks sellele hoitakse olulisi andmeid krüpteeritud kujul. Olulisteks andmeteks on näiteks poe infosüsteemiga ühendumise info (aadress), müüja-ostja kontaktandmed, ligipääsuload. Krüpteerimisel kasutatakse avaliku võtme krüptograafiat.

4.6 Mikroteenustevaheline kommunikatsioon

Smarts iseteenindussüsteem rajaneb mikroteenuste põhisel arhitektuuril. See tähendab, et iga mikroteenus on iseseisvalt arendatav, hallatav, paigaldatav, uuendatav ja jälgitav. Et kogu infosüsteem tervikuna toimima hakkaks, peavad mikroteenused omavahel suhtlema. Mikroteenustevaheline kommunikatsioon põhineb REST *(Representational State Transfer)* arhitektuuril. REST arhitektuur kasutab andmevahetuseks HTTP protokolli. Joonis 7 näitab REST arhitektuuril põhinevat kommunikatsiooni üle HTTP protokolli.



Joonis 7 REST arhitektuur kasutades HTTP protokolli

REST arhitektuur on valitud seetõttu, et see on üks lihtsamaid ja kergemini kasutatavaid mikroteenuste vahelisi kommunikatsioonivõimalusi, mis võimaldab paremat laiendatavust ja jõudlust võrreldes tavaliste RPC-stiilis *(Remote Procedure Call)* veebiteenustega (Soap) [25]. Ilma olekuta *(stateless)* veebiteenuste vaheline suhtlus tähendab, et veebiteenus ise ei hoia endas infot kliendi kohta, st klient peab iga päringuga andma kaasa kogu vajaliku info, et server päringut õigesti tõlgendaks. See võimaldab paremat skaleeritavust, sest kogu süsteem on ehitatud olekuvabana.

4.7 Kasutatud tehnoloogiad

Selles peatükis tutvustatakse arenduse faasis kasutatud programmeerimiskeelt koos rakendusraamistiku ja muude teekidega. Lisaks sellele põhjendatakse ka projektihaldustarkvara kasutamist ja tuuakse välja kasutatud tarkvara.

4.7.1 Programmeerimiskeel

Mikroteenustel põhinev arhitektuur annab erinevalt monoliitsele arhitektuurile eelise kirjutada iga mikroteenus erinevas programmeerimiskeeles. Kuna tegu on REST teenustega, siis suhtlevad need omavahel üle HTTP protokolli. Programmeerimiskeelena kasutatakse suhtlusteenuse ja liidestustarkvara rajamisel Java SE 8. Java valiti seetõttu, et see oli arendusmeeskonnale selge, sellel põhinevad paljud kasulikud raamistikud REST teenuste ehitamiseks ning see on TIOBE indeksi järgi maailmas seisuga aprill 2018 kõige populaarsem programmeerimiskeel [26]. Lisaks sellele on Javas kirjutatud ka teised Smarts infosüsteemi mikroteenused.

4.7.2 Suhtlusteenuse rakendusraamistik

Suhtlusteenus on oma olemuselt veebiteenus, mis peab olema võimeline suhtlema teiste veebiteenustega üle HTTP protokolli. Rakendusraamistikuks valiti Spring ning töö lihtsustamiseks omakorda Spring Boot. Spring Boot võtab enda kanda suure hulga konfigureerimist, mis tehakse automaatselt [27]. See võimaldab arendajal luua kiirelt toimivaid lahendusi, mis Smarts iseteenindussüsteemi MVP arenduse juures on oluline. Lisaks sellele on Spring Boot kasutusel ka teistes mikroteenustes ning selle kasutamine ka suhtlusteenuses tagab projektiülese ühtsuse tarkvarakomponentide osas, mis lihtsustab hilisemat süsteemi edasiarendust.

4.7.3 Teised kasutatud teegid

Suhtlusteenuse ja liidestustarkvara loomisel kasutatakse lisaks Java programmeerimiskeelele ja rakendusraamistikule (suhtlusteenuse juures) ka erinevaid teisi teeke, mis muudavad tarkvaraarenduse kiiremaks ja lihtsamaks ning koodi loetavamaks.

Lombok

Lombok on teek, mis muudab programmikoodi lühemaks ja kergemini loetavamaks [28]. Seda saab kasutada põhiliselt POJO *(Plain Old Java Object)* juures asendamaks *getter* ja *setter* meetodid Lombok klassitaseme annotatsioonidega. Lisaks on Lombokit kasulik kasutada koos Spring rakendusraamistikuga, kus on vaja ühendada omavahel kontrollereid ja teenusklasse annotatsiooniga `@Autowired`. Kui ühes teenusklassis X on viiteid paljudele teistele teenusklassidele, võib nimekiri koos `@Autowired` annotatsiooniga minna pikaks. Selle vältimiseks ja klassi kujutava tekstifaili pikkuse lühendamiseks saab kõigil selles teenusklassis X viidatud teenustel `@Autowired` annotatsioonide asemel kasutada Lomboki klassitaseme annotatsiooni `@RequiredArgsConstructor(onConsturctor__@(@Autowired))`. See asendab kõik viidatud teiste teenusklasside `@Autowired` annotatsioonid, kuid siinkohal tuleb kõik viidatud teenusklassid määrata kõnealuses teenusklassis X *final* tüüpi.

Guava

Guava on Google arendatud teek Java programmeerimiskeele juurde, mis sisaldab suurel hulgal operatsioone massiividega, vahemälu võimalust, stringitöötlust ja palju muud [29]. Peamiselt on tegu tuumikfunktsionaalsusega, mida saab kasutada laialdaselt väga erinevates projektides. Antud töös on Guavat kasutatud kollektsioonide kirjeldamisel, sest Guava võimaldab massiive kirjeldada palju loetavamalt ja lühemalt kui puhas Java programmeerimiskeel.

4.7.4 Kasutatud projektihaldus- ja integratsioonitarkvara

Infosüsteemi arendamisel kasutati Atlassian JIRA, Confluence ja Bitbucket rakendusi [30]. Need rakendused on vajalikud projekti haldamise juures, et tagada kogu meeskonna kaasatust ja info dokumenteeritust. Atlassiani tooted valiti seetõttu, et need on kõigile meeskonna liikmetele vähemalt mingil määral tuttavad ning Atlassiani tooted on omavahel ka hästi integreeritud. Atlassiani tooteid kasutatakse antud projektis järgnevalt:

- JIRA – projektihaldusseks ja tööde ülevaateks; võimaldab jälgida tööde edenemist ja inimeste koormatust
- Confluence – dokumendihalduseks; võimaldab hoida ühes kohas infosüsteemi dokumentatsiooni ning siduda seda vajadusel JIRA töödega

- Bitbucket – koodivaramu, kus hoitakse kogu infosüsteemi programmikoodi; programmikood on jagatud eri koodihoidlate vahele nii, et igas koodihoidlas on üks mikroteenus

Integratsioonitarkvarana kaustatakse antud projektis Jenkinsit [31]. Jenkins on kasutusel süsteemi ehitusel, kus kõik mikroteenused tuleb tervikuna tööle panna. See võimaldab jälgida eri teenuste olekut, automatiseerida nende ehitust (koostöös Bitbucket koodihoidlaga) ning hallata teenuseid vastavalt vajadusele. Lisaks sellele jooksutab Jenkins igal teenuse ehitamisel läbi kõik selle teenuse testid.

4.7.5 Kasutatud tööriistad

Infosüsteemi osi modelleeriti kasutades Enterpise Architect versioon 12.1 modelleerimistarkvara. Arenduse etapil kasutati IntelliJ IDEA 2018 arendusvahendit. Arenduse käigus testiti REST päringuid Postman tarkvaraga. Tööks MongoDB andmebaasiga kasutati MongoDB Compass rakendust.

5 Valminud süsteemi analüüs

Suhtlusteenust ja liidestustarkvara disainides püstitati hulk nõudeid. Arenduse käigus said kõik püstitatud nõuded realiseeritud. Valminud tarkvara integreeriti Smarts iseteenindusplatvormi üldisesse arhitektuuri ja testiti selle toimimist koos teiste komponentidega. Ilmnesid mitmed tähelepanekud, mida kirjeldatakse allpool.

5.1 Valminud tarkvara testimine

Valminud tarkvara testiti, et kontrollida selle vastavust algselt püstitatud nõuetele. Tarkvara testiti esmalt ühiktestidega. Ühiktestid kirjutati olulisematele tarkvaraklassidele, et veenduda klassi meetodite korrektses käitumises. Ühiktestid aitavad lisaks koodi korrektse käitumise kontrollimisele leida ka vigu, mida võidakse teha hiljem nende samade tarkvaraklasside meetodite muutmisel. Ühiktestid kirjutati JUnit raamistikus, mis on Java programmeerimiskeele juurde kuuluv ühiktestide raamistik.

Lisaks ühiktestimisele tehti koodi läbivaatust. Läbivaatuse eesmärk oli kontrollida valminud tarkvara vastavust nõuetele, parandada koodi kvaliteeti, hoida üle projekti ühtlast taset ning võimaldada meeskonnal vahetada omavahel teadmiseid.

Viimaks, kui selles töös kavandatud tarkvara oli valmis ja eelnevad testimise etapid olid läbitud, viidi läbi vastuvõtutest. Vastuvõtutesti käigus integreeriti valminud osad Smarts iseteenindusplatvormi ning kontrolliti nende osade toimimist tervikus. Kuna käesoleval hetkel luuakse alles süsteemist esialgset töötavat versiooni (MVP), siis kasutati vastuvõtutestidel tellija süsteemi rollis Smarts meeskonna enda arendatud testpoe infosüsteemi. Testpoe infosüsteem programmeeriti enne antud töös rajatud suhtlusteenust ja liidestustarkvara. Seega oli vastuvõtutesti raames võimalik kontrollida, kui keeruline on liidestustarkvara testpoe infosüsteemiga liidestada.

5.1.1 Testimisel tuvastatud probleemid ja nende lahendused

Testimise käigus ilmnesid mitmed tõsiasjad, mis vajasid tähelepanu. Algselt oli püsitatud nõue, mille kohaselt liiguks Smartsi ja poe vahel krüpteeritud arve (kasutatakse avaliku võtme krüptograafiat), mis sisaldaks ostmisele minevaid tooteid. See tähendas, et kui kasutaja skaneeris uue toote, saatis Smarts poele hetkelise krüpteeritud arve ja toote ribakoodi. Pood pidi seejärel toodet otsima, arve lahti krüpteerima, toote lisama ja enne

Smartsile arve tagastamist arve uuesti krüpteerima. Seejärel pidi Smarts arve lahti krüpteerima, uue arve kogusumma arvutama ning arve uuesti krüpteerima. Koormustestides selgus, et suure kasutajaskonna korral võib arve pidev krüpteerimine hakata vigaseid tulemusi andma. Lahendusena loobuti kogu arve krüpteerimisest ja pidevalt Smartsi ja poe vahel saatmisest. Võeti kasutusele Redis vahemälu andmebaas, kuhu salvestatakse pooleliolev arve ning mida sealt erinevad mikroteenused vajadusel pärivad. Redis vähemälu andmebaasist arve küsimiseks peab teenus teadma õiget arve tunnust. See toimub kõik Smarts infosüsteemi sees. Poele saadetakse alles lõpuks arve, mis kajastab kõiki kasutaja oste.

Peale selle ilmnes ühisel läbivaatusel probleem toote valdkonnamudeli kirjeldamisega. Algselt oli see seotud otse e-arve formaadiga nii, et toote objekt oli sõltuvuses e-arve *ItemEntry* objektist. *ItemEntry* on e-arve formaati kuuluv objekt, mis kajastab arvel ühte rida (toodet). Taoline sõltuvus on aga halb, sest see seob omavahel kaks täiesti erineva vastutusega objekti: valdkonnamudelis kirjeldatud toode on põhimõtteliselt DTO *(Data Transfer Object)* ehk andmevahetusobjekt poe ja Smarts infosüsteemi vahel; *ItemEntry* on aga e-arve formaadis arve osa, millel puudub seos valdkonnamudelis kirjeldatud tootega. Seetõttu otsustati lahendusena hoida e-arve formaadile vastavat arvet täiesti puhtana, st see koosneb ainult e-arve formaadis kirjeldatud objektidest. Selline lähenemine lihtsustab tulevikus teistel osapooltel arvest arusaamist, sest Smartsi väljastatud arve vastab täielikult e-arve formaadis kirjeldatule.

5.2 Valitud metoodika sobivus

Agiilse tarkvaraarenduse põhimõtetest lähtumine oli kasulik. Kuna tööd alustades ei olnud kõik nõuded veel täpselt paigas, oli agiilseid meetodeid kasutades võimalik varem alustada ka arenduse ja testimisega. Agiilsetest tarkvaraarendusmetoodikatest lähtuti antud töös põhiliselt RUP *(Rational Unified Process)* ideest, mis põhineb iteratsioonide kasutamisel. Projektihaldustarkvaras kirjeldati ära hulk nõudeid, mis seejärel ka kohe realiseeriti. Kui võrrelda valitud agiilset tarkvaraarendusmetoodikat kose mudeliga, kus järgmist faasi saab alustada pärast eelmise faasi lõppu, oli agiilne tarkvaraarendusmetoodika paindlikum.

Kuna JIRA kasutamine võimaldab jagada tööd suurematekstükkideks *(epic)* ja suuremaid omakorda väiksemateks *(task)* ning neid töid konkreetsele inimesele suunata, siis

võimaldas see paika panna töövoo. Meeskonnas oli kaks arendajat, kes tegid töö (Jira *task*) valmimisel koodiülevaatuse teisele arendajale. Teine arendaja pidi seejärel kontrollima, et valminud kood vastab töös püsitatud nõuetele ning on kirjutatud korrektselt. Vajadusel tuli siin teha teise arendaja ettepanekul parandusi. Vastasel juhul, kui kõik oli korras, sai töö koodihoidlasse põhiharru panna. Selline töövoog aitas hoida projektis ühtset stiili, vähendas võimalike programmeerija vigade tekkimist ja hoidis meeskonna mõlemad arendajad kursis projekti hetkeseisuga.

5.3 Olulisemad järeldused

Liidestustarkvara arendamisel selgus, et ühel hetkel on seda vaja sõltuvusena kasutada testpoe infosüsteemis. See tingis vajaduse leida võimalus, kuidas pood saaks selle tarkvara Maven sõltuvusena oma projektis kasutusele võtta. Kuna käesoleval hetkel tuleb arvestada, et liidestustarkvara võib veel täiendusi vajada, siis ei ole seda veel otstarbekas Maven kesksesse koodihoidlasse *(Maven Central Repository)* üles panna. Seetõttu leiti lahendus, kuidas süsteemi arendusega edasi minna nii, et liidestustarkvara sõltuvus oleks saadaval mujal allikas kui Maven keskses koodihoidlas. Lahendusena kasutati Githubi võimalust pakkuda sarnast sõltuvuse serveerimise funktsionaalsust. Selleks tehti Githubi avalik repositoorium, kuhu paigaldati käsuga `mvn install` liidestustarkvara, mis oli eelnevalt pakitud JAR (Java Archive) failiks. Seejärel oli võimalik testpoe infosüsteemis Maven sõltuvuste seadistuses määrata lisaks Maven kesksele koodihoidlale eraldi koodihoidla (Github), kust Maven laadis alla vajaliku liidestustarkvara.

Mikroteenustel põhinev tarkvara annab vabaduse programmeerida iga mikroteenus keeles, mis on selles infosüsteemis kasutusel. See tähendab, et Smarts iseteenindussüsteemiga liidestuvad poed ei pea olema ehitatud kasutades Java programmeerimiskeelt ja Spring rakendusraamistikku. Liidestustarkvara peab tagama ainult õigel kujul andmeesituse, mis saadetakse seejärel üle interneti HTTP protokolli kasutades Smarts iseteenindusplatvormile.

5.4 Edasised tegevused

Hetkel on liidestustarkvara abil küll võimalik määrata, kas toote ostmisel tuleb läbi viia ostukontroll, kuid Smarts süsteemis puudub funktsionaalsus selle läbiviimiseks. Selleks tuleb rajada eraldi mobiilirakendus, mis on mõeldud poes müüja telefonis kasutamisele.

Teiseks tuleb liidestustarkvara programmeerida ka teistes keeltes peale Java. Programmeerimisel tuleb jälgida turu nõudlust teiste programmeerimiskeelte osas. Lisaks tuleb liidestustarkvara teha saadavaks ka teistes kesksetes koodihoidlates peale Maveni (näiteks Gradle).

Kolmandaks võib tulevikus kaaluda Smartsis profileerija kasutuselevõttu. Profileerija võimaldab kergemini määrata süsteemis neid kohti, mis kasutajaid segavad kas aegluse või liigse ressursikasutuse tõttu. Arenduse faasis sobib selleks näiteks XRebel ning hiljem käitluse faasis näiteks Plumbr.

Kokkuvõte

Töö eesmärk oli analüüsida ja realiseerida Smarts mobiilse iseteenindusplatvormi kaks komponenti: suhtlusteenus ja liidestustarkvara, mis seejärel integreerida Smarts iseteenindusplatvormi üldisesse arhitektuuri ja testida süsteemi tööd koos rajatud kahe komponendiga. Töös pöörati põhirõhku nende komponentide analüüsile, arendusele ja testimisele.

Töö tulemusena valmisid nimetatud kaks komponenti, mis seejärel integreeriti Smarts iseteenindusplatvormi üldisesse arhitektuuri. Need kaks komponenti võimaldavad Smarts iseteenindusplatvormil luua ühenduse liidestunud poe infosüsteemiga ning teha seeläbi vajalikke päringuid poe infosüsteemi.

Järeldustena võib välja tuua õige metoodika valiku. Taolise infosüsteemi arendamise juures oli oluline järgida agiilset arendusmetoodikat, sest nõuded ei pruukinud alguses kõik kas paika pidada või üleüldse eksisteerida. Agiilsete meetoditega lähenemine võimaldas hiljem muudatusi kergemini sisse viia. Teiseks oluliseks järelduseks võib tuua mikroteenustel põhineva arhitektuuri sobivuse taolise ülesande lahendamiseks. Kuna süsteemi koormus võib kasutamise järgus minna suureks, on mikroteenustel põhieva arhitektuuriga süsteemis kerge servereid juurde paigaldada.

Süsteemi edasiste arenduste juures tuleks liidestustarkvara programmeerida ka teistes programmeerimiskeeltes ja viia see sõltuvusena saadavale ka teistes koodihoidlastes peale Maveni. Peale selle tuleb võimaldada täielik ostukontrolli funktsionaalsus, mis võimaldab poe müüjal oma telefoniga ostukontrolli kliendi ostetud toodete peal läbi viia.

Kasutatud kirjandus

[1] N. Marinoff, „South Koreans Will Be Able to Pay in Cryptocurrency at Over 6,000 Stores,“ BitcoinMagazine, 28 03 2018. [Võrgumaterjal]. Available: https://bitcoinmagazine.com/articles/south-koreans-will-be-able-pay-cryptocurrency-over-6000-stores/. [Kasutatud 27 04 2018].

[2] U. W. Chohan, „A History of Bitcoin,“ University of New South Wales (UNSW), UNSW Business School, Canberra, 2017.

[3] R. C. Martin, Clean Code A Handbook of Agile Software Craftsmanship, Boston: Pearson Education, Inc, 2009.

[4] Mario Villamizar, Oscar Garcés, Harold Castro, Mauricio Verano, Lorena Salamanca, Rubby Casallas, „Evaluating the Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the Cloud,“ %1 *Computing Colombian Conference*, Bogotá, 2015.

[5] „mTasku,“ Telia Eesti AS, 2017. [Võrgumaterjal]. Available: http://www.mtasku.ee/. [Kasutatud 26 03 2018].

[6] „SelveEkspress,“ Selver AS, [Võrgumaterjal]. Available: https://www.selver.ee/selveekspress. [Kasutatud 26 03 2018].

[7] „Nutikassa,“ Coop Eesti Keskühistu, [Võrgumaterjal]. Available: https://www.coop.ee/nutikassa. [Kasutatud 26 03 2018].

[8] K. Jung, „Sinustki võib saada kassapidaja,“ *[Digi]*, pp. 62-63, 2018.

[9] „Sam's Club Scan & Go,“ Sam's West, Inc, [Võrgumaterjal]. Available: https://www.samsclub.com/sams/html/mobile/help/scan-and-go.html. [Kasutatud 27 03 2018].

[10] Ivan Nanut, Alex Mihaljcic, Photis Georgiades, „Eterbank Whitepaper,“ 2018. [Võrgumaterjal]. Available: https://eterbank.com/whitepaper.pdf. [Kasutatud 21 04 2018].

[11] G. Bidzilja, „Z Basket,“ [Võrgumaterjal]. Available: https://zbasket.eu/. [Kasutatud 26 04 2018].

[12] „ISO/IEC 25010:2011,“ International Organization for Standardization, [Võrgumaterjal]. Available: https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en. [Kasutatud 02 04 2018].

[13] K. SANDOVAL, „What is the Difference Between an API and an SDK?,“ NordicApis, 02 06 2016. [Võrgumaterjal]. Available: https://nordicapis.com/what-is-the-difference-between-an-api-and-an-sdk/. [Kasutatud 26 04 2018].

[14] E. Pangaliit, „Eesti e-arve kirjeldus ver 1.2,“ 01 12 2013. [Võrgumaterjal]. Available: http://pangaliit.ee/images/files/E-arve/Eesti_e-arve_kirjeldus_ver1.2_est.pdf. [Kasutatud 30 03 2018].

[15] „ISO 639-1:2002,“ International Organization for Standardization, 07 2002. [Võrgumaterjal]. Available: https://www.iso.org/standard/22109.html. [Kasutatud 25 03 2018].

[16] „Language codes - ISO 639,“ International Organization for Standardization, [Võrgumaterjal]. Available: https://www.iso.org/iso-639-language-codes.html. [Kasutatud 25 03 2018].

[17] C. Larman, Applying UML and Patterns: An Introduction to Object-oriented Analysis and Design and the Unified Process, Upper Saddle River, New Jersey: Prentice Hall Professional, 2002.

[18] Rafal Kuc, Marek Rogozinski, Elasticsearch Server, Birmingham: Packt Publishing Ltd., 2013.

[19] „DB-Engines,“ solid IT, 04 2018. [Võrgumaterjal]. Available: https://db-engines.com/en/ranking/search+engine. [Kasutatud 11 04 2018].

[20] „Kibana,“ Elasticsearch BV, 2018. [Võrgumaterjal]. Available: https://www.elastic.co/products/kibana. [Kasutatud 15 04 2018].

[21] „Elasticsearch Reference,“ Elasticsearch BV, [Võrgumaterjal]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/6.2/_basic_concepts.html. [Kasutatud 10 04 2018].

[22] Milad Shokouhi, Luo Si, „Federated Search,“ *Foundations and Trends® in Information Retrieval*, kd. 5, p. 120, 2011.

[23] C. AS|, „STANDARDIPÕHINE TARKVARATEHNIKA SÕNASTIK,“ CYBERNETICA AS|, [Võrgumaterjal]. Available: https://stats.cyber.ee/. [Kasutatud 22 04 2018].

[24] „Redis,“ Redis Labs, [Võrgumaterjal]. Available: https://redis.io/. [Kasutatud 25 04 2018].

[25] Xinyang Feng, Jianjing Shen, Ying Fan, „REST : An Alternative to RPC for Web Services,“ %1 *First International Conference on Future Information Networks*, 2009.

[26] „TIOBE,“ TIOBE software BV, 04 2018. [Võrgumaterjal]. Available: https://www.tiobe.com/tiobe-index/. [Kasutatud 30 04 2018].

[27] Phil Webb, Dave Syer, „Spring Boot – Simplifying Spring for Everyone,“ Pivotal Software, Inc., 06 08 2016. [Võrgumaterjal]. Available: https://spring.io/blog/2013/08/06/spring-boot-simplifying-spring-for-everyone. [Kasutatud 24 04 2018].

[28] Reinier Zwitserloot, Roel Spilker, „Project Lombok,“ [Võrgumaterjal]. Available: https://github.com/rzwitserloot/lombok. [Kasutatud 26 04 2018].

[29] „Guava: Google Core Libraries for Java,“ Google, [Võrgumaterjal]. Available: https://github.com/google/guava. [Kasutatud 26 04 2018].

[30] „Atlassian,“ [Võrgumaterjal]. Available: https://www.atlassian.com/. [Kasutatud 30 04 2018].

[31] „the Jenkins project,“ Jenkins, [Võrgumaterjal]. Available: https://jenkins.io/. [Kasutatud 26 04 2018].

[32] „Eesti Pangaliit,“ 2018. [Võrgumaterjal]. Available: http://pangaliit.ee/et/arveldused/e-arve. [Kasutatud 30 03 2018].

[33] A. Brasetvik, „Elastic,“ Elasticsearch BV, 16 09 2013. [Võrgumaterjal]. Available: https://www.elastic.co/blog/found-elasticsearch-from-the-bottom-up. [Kasutatud 12 04 2018].

Lisa 1 – Minimaalselt vajalik e-arve formaat

Selles lisas on toodud Pangaliidu koostatud e-arve formaadi struktuur, kus on kirjeldatud minimaalselt nõutud välju. Struktuur põhineb Pangaliidu e-arve dokumentatsioonil [14].

```
<?xml version="1.0" encoding="UTF-8"?>
<E_Invoice xsi:noNamespaceSchemaLocation="e-invoice_ver1.2.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
	<Header>
		<Date>2013-12-01</Date>
		<FileId>1</FileId>
		<Version>1.2</Version>
	</Header>
	<Invoice sellerRegnumber="12345678" invoiceId="1234"
regNumber="30101011234">
		<InvoiceParties>
			<SellerParty>
				<Name>TESTMÜÜJA AS</Name>
				<RegNumber>12345678</RegNumber>
			</SellerParty>
			<BuyerParty>
				<Name>TESTOSTJA AS</Name>
			</BuyerParty>
		</InvoiceParties>
		<InvoiceInformation>
			<Type type="DEB"/>
			<DocumentName>ARVE</DocumentName>
			<InvoiceNumber>1234</InvoiceNumber>
			<InvoiceDate>2013-12-01</InvoiceDate>
		</InvoiceInformation>
		<InvoiceSumGroup>
			<TotalSum>1.20</TotalSum>
		</InvoiceSumGroup>
		<InvoiceItem>
			<InvoiceItemGroup>
				<ItemEntry>
					<Description>Ostetud teenus</Description>
				</ItemEntry>
			</InvoiceItemGroup>
		</InvoiceItem>
		<PaymentInfo>
			<Currency>EUR</Currency>
			<PaymentDescription>Arve number 1234</PaymentDescription>
			<Payable>NO</Payable>
```

```
                    <PaymentTotalSum>1.20</PaymentTotalSum>
                    <PayerName>TESTOSTJA AS</PayerName>
                    <PaymentId>1234</PaymentId>
                    <PayToAccount>EE909900123456789012</PayToAccount>
                    <PayToName>TESTMÜÜJA AS</PayToName>
                </PaymentInfo>
        </Invoice>
        <Footer>
                <TotalNumberInvoices>1</TotalNumberInvoices>
                <TotalAmount>1.2</TotalAmount>
        </Footer>
</E_Invoice>
```