TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Sander Ruul 121979IAPM

MIKROTEENUSTEL TARKVARA ARHITEKTUURI LOOMINE NUTIKASSA ÄRIPROTSESSIDE NÄITEL

Magistritöö

Juhendaja: Tarmo Veskioja
Doktori kraad
Teadur

Tallinn 2017

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: Sander Ruul

08.05.2017

Annotatsioon

Mikroteenustel tarkvara arhitektuuri loomine nutikassa äriprotsesside näitel

Käesolev magistritöö eesmärgiks oli prototüüpimise abil tõestada mikroteenustel tarkvara teostatavust.

Töö käigus kirjeldati nutikassa funktsionaalsus vastavalt Zachmani raamistikule, disainiti andmemudel ning mikroteenustel põhinev süsteem.

Töö tulemusena valmis süsteemi kirjeldav dokumentatsioon ning sellele vastav süsteem, mille jõudlust valideeriti jõudlustestidega.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 60 leheküljel, 7 peatükki, 32 joonist, 11 tabelit.

Abstract

Software architecture with micro-services in smart-cashier business processes

The goal of this thesis is to design application with micorservices architecture and demostrate it with smart-cashier business processes.

This thesis describes the functionality used by main business processes of the smart-cashier system. The decsription follows Zachman enterprise architecture framework.

The outcome of this thesis is documentation that describes the system. The system was validated with performance tests.

The thesis is in Estonia and contains 60 pages of text, 7 chapters, 32 figures, 11 tables.

Lühendite ja mõistete sõnastik

POC	(*Proof of concept*) meetodi või lahenduse realiseerimine tõestamaks teostatavust
Logstash	elastic poolt loodud programm logide hoidmiseks.
Kibana	elastic poolt loodud veebirakendus, mis pakub graafilist kasutajaliidest, mille kaudu saab otsida logisid kasutades ElasticSearch päringuid.
REST	(*Representational State Transfer*) on suhtluskanal, mille kaudu saavad einevad süsteemid üksteisega suhelda üle võrgu.
Kafka	Kafka on laiendatav sõnumi-jagamise platvorm. [18]
ElasticSearch	elastic poolt loodud programm logide otsimiseks.
SOA	(*Service oriented architecture*) tarkvara disain, kus süsteem on jagatud erinevateks alamsüsteemideks.
Eureka	netflixi poolt loodud server, mis registreerib endasse mikroteenused.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Mobiiliseadmete kasutamine on viimaste aastatega kasvanud väga kiiresti. Tänu tehnoloogia arengule on muutunud mobiiliseadmed võimsamaks, kiiremaks ning mugavamaks. Tekkinud on palju erinevaid lahendusi ning raamistikke uute mobiilirakenduste loomiseks, tänu sellele on väga populaarseks muutunud ka erinevad mobiilirakendused, mida saab kasutada igapäeva elu lihtsustamiseks, näiteks Pocopay ja Taxify.

Populaarseks muutunud mobiilirakendused leiavad väga suurt kasutust. Infosüsteemid, mida need rakendused kasutavad, peavad pidama sellele juurele koormusele vastu ning suutma kiiresti kasutaja käske täita. Käesoleva töö eesmärk on välja töötada laiendatav infosüsteemi lahendus lähtudes Nutikassa äriprotsessidest.

Nutikassa on mobiilirakendus, mida saab kasutaja ise oma telefoni paigaldada ning osta kaupa registreeritud kauplustes kiirelt ilma järjekordadeta ka tööpäeva lõpus tipptunni ajal.

1.1 Taust ja probleemid

Eestis on kasutusel erinevaid kassasüsteemide lahendusi. Selveris saavad ainult partnerkaardi omanikud kasutada Selveri pulti, millega saab oma kauba saali peal ise ära registreerida ning hiljem lihtsalt kassas maksta. Prismas, Rimis ning Konsumis tuleb kliendil oma kaup iseteeninduskassas ise registreerida ning siis maksta. Mõlemal lähenemisel on omad head ja vead.

Probleemidena tuleb välja tuua:

- Iga kaubanduskett loob ise endale sama funktsionaalsusega IT süsteeme (aja kulu, raha kulu)
    - IT kiire arenguga kaasaskäimine tekitab lisakulusid

    - IT süsteemidesse uue funktsonaalsuse lisamine võib olla keeruline ja ebatöökindel

- Igal kaubandusketil on omad kliendikaardid (kliendi jaoks tülikas)
- Vähemalt korra peab kassast läbi käima
- Pidevalt peab rahakotist pangakaarte / kliendi kaarte otsima

1.2 Ülesande püstitus

- Kirjeldada äriprotsessid ning andmemudel lähtudes Zachmani raamistikust
- Uurida mikroteenuste arhitektuuri
- Realiseerida POC rakendus valitud arhitektuuril ning valideerida jõudlust vastavalt võimalikule koormusele

1.3 Metoodika

Ülesande püstituse saavutamiseks kirjeldab autor nutikassa äriprotsessid mahutades need Zachmani raamistikku. Seejärel uurib mikroteenustel põhinevat tarkvara arhitektuuri ning realiseerib infosüsteemi mikroteenuste.

1.4 Ülevaade tööst

Töö alguses antakse ülevaade Zachmani ettevõtte raamistikust ja mikroteenustest. Seejärel uuritakse ja kirjeldatakse süsteem lähtudes Zachmani raamistikust. Lõpuks valideeritakse töö tulemus näidisrakenduse peale tehtud jõudlustestidega.

2 Zahmani raamistik

Zachmani raamistik on John Zachmani poolt IBM tarbeks välja töötatud infosüsteemi arhitektuuri raamistik [1] . Zachmani raamistik 6 erinevat vaadet lähtudes 6 erinevast perspektiivist (tasandist). Zachmani raamistiku poolt kirjeldatud printsiibid on Planeerija vaade (*Executive perspective*), Omaniku vaade (*Business Management Perspective*), Projekteerija vaade (*Architect Perspective*), Ehitaja vaade (*Engineer Perspective*), Alltöövõtja vaade (*Technician perspective*), Toimiv organitatsioon (*Enterprise Perspective*) [2] .

Zachmani printsiipide lühikirjeldus [2] :

- Planeerija vaade (*Executive perspective*) – kõige üldisem vaade, kus kirjeldatakse üldised eesmärgid ning antakse üldistatud ülevaade süsteemist, skoobist ning maksumusest. [2]

- Omaniku vaade (*Business Management Perspective*) – arhitekt koostab läbi omaniku silmade arhitektuuri joonised [2] .

- Projekteerija vaade (*Architect Perspective*) – koostatakse detailsemad süsteemi nõuded. Need peavad olema piisavad, et süsteemi analüütik suudab nende põhjal koostada andmemudelid, põhiprotsessid ning fuktsioonid [2] .

- Ehitaja vaade (*Engineer Perspective*) – kirjeldatakse täpsemalt lahti arhitekti plaanid piisava täpsusega, et aru saada tehnoloogilistest piirangutest. Valitakse tehnoloogiad [2] .

- Alltöövõtja vaade (*Technician perspective*) – antakse spetsifikatsioon programmeerijatele, kes loovad etteantud funktsionaalsuse lähtudes spetsifikatsioonist [2] .

- Toimiv organisatsioon (*Enterprise Perspective*)

Tasand	**MIS** Andmed	**KUIDAS** Funktsioonid	**KUS** Paiknemine, võrk	**KES** Inimesed	**MILLAL** Aeg	**MIKS** Motivatsioon
Planeerija vaade	Oluliste mõistete ja objektide loend	Põhiliste toimimis-protsesside loend	Organisatsiooni eri üksuste asukohad	Võtmerühmad, töötajate loend	Olulisemate Sündmuste loend	Toimimis-eesmärgid ja -strateegiad
Omaniku vaade	Kontseptuaalne andmemudel	Toimimisprotsesside mudel	Logistika skeem	Töövoog, töötajate vastutused (*workflow*)	Protsesside stsenaariumid (master *plan*)	Toimimisplaan (äriplaan)
Projekteerija vaade	Loogiline andmemudel (ERD)	Rakenduste arhitektuur	Hajussüsteemi arhitektuur	Kasutajaliides-te arhitektuur	Protsesside struktuur	Toimimisreeglite mudelid
Ehitaja vaade	Füüsiline andmemudel	Süsteemi projekt (spetsifikatsioon)	Tehnoloogia arhitektuur	Kasutajaliides-te disain	Juhtimis-struktuur	Toimimisreeglite disain
Alltöövõtja vaade	Andmete struktuuri Kirjeldus	Programmi disain	Võrgu-arhitektuur	Turbe arhitektuur, kasutajaõigu-sed	Ajalise seotuse määratlus	Toimimisreeglite spetsifikatsioon
Toimiv organisatsioon	Tegelikud andmed	Programmi kood	Võrk, süsteemide paiknemine	Pädevad töötajad	Tegevustsüklid	Rakendatud toimimisreeglid

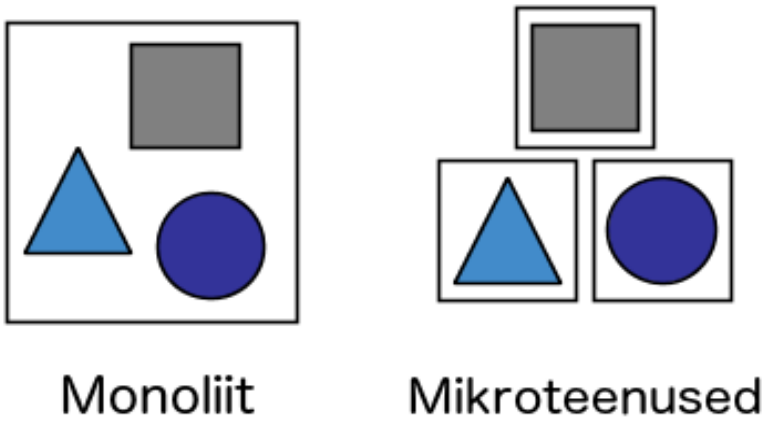
Tabel 21: Zachmani raamistik

3 Mikroteenused

Traditsiooniliselt on IT süsteemides monoliitset arhitektuuri. Monoliitsed rakendused on sellised rakendused, kus kogu funktsionaalsus asub ühes suures rakenduses. [9]

Monoliitsetest süsteemidest arenes edasi teenuse põhine arhitektuur (*SOA*). Teenuste põhises arhitektuuris jagati monoliitsed rakendused üksikuteks pisemateks alamsüsteemideks, mis igaüks hõlmab suurt osa funktsionaalsusest. [10]

Mikroteenused on teenuse põhise arhitektuuri alamliik, mis jagab süsteemi omavahel madala sõltuvustega väikesteks teenusteks.



Joonis 1: Monoliitrakendus ja mikroteenuste rakendus

3.1 Mikroteenuste laiendamine

Kaasaegsed süsteemid peavad suutma töötama suurel hulgal klientidega. Süsteemide laiendamisel on kaks viisi – vertikaalne laiendamine ja horisontaalne laiendamine. [19]

Joonis 2: Vertikaalne ja horisontaalne skaleerimine

Horisontaalne laiendamise korral dubleeritakse rakendust selliselt, et ühel aja hetkel töötab üks rakendus mitmes serveris üheaegselt. Koormust nende rakenduste vahel jagab koormuse tasakaalustaja (*Loan balancer*). Koormuse kasvades lisatakse rakendusi juurde. Horisontaalse laiendamise korral peavad rakendused olema sessioonivabad. [19]

Vertikaalne laiendamine on tähendab füüsiliste ressursside (protsessori võimsus, mälumaht) suurendamist. [19]

3.2 Mikroteenuste eelised [11]

- Mikroteenused on üksteisest sõltumatud, lihtsad.
- Mikroteenused valmivad kiirest tänu väikesele funktsionaalsusele.
- Mikroteenuseid on võimalik luua ning testida ilma, et oleks vaja teisi teisi teenuseid, ega tervet süsteemi.

- Mikroteenuseid on võimalik üksteisest sõltumatult laiendada vastavalt vajadusele.
- Mikroteenuseid on väga hästi testitavad.
- Veaolukorrad ei halva tervet süsteemi.
- Mikroteenuseid on võimalik üksteisest sõltumatult käivitada ning paralleelselt ja seeläbi kiirendada toodangu keskkonda uue funktsionaalsuse lisamist.
- Mikroteenused ei nõua võimast riistvara, ressurssi puuduse korral saab vastavat mikroteenust lihtsasti horisontaalselt laiendada.

3.3 Mikroteenuste puudused [11]

- Mikroteenused on üksteisest sõltumatud ning vajavad eraldi monitoorimist, käivitamist ning konfiguratsiooni haldamist.
- Igale mikroteenusele on vaja luua üksteisest sõltumatu elutsükkel (versioonihaldus, automaatsete testide käivitamine, rakendusse ehitamine, arhiveerimine, käivitamine)
- Suur võrgukoormus
- Andmebaasi transaktsiooni haldus
- Refaktoreerimine on keeruline, kui on vaja mingi osa funktsionaalsusest ühest teenusest teise tõsta.

3.4 Mikroteenuste parimad praktikad [12]

Mikroteenused pakuvad kiiret funktsionaalsuse realiseerimist. Iga mikroteenuse loomisega kasvab kogu süsteem ning muutub üha keerulisemaks. Keerukuse haldamiseks on vaja kindlalt piiritleda nõuded üksikutele mikroteenustele. [12]

- Mikroteenuseid tuleb luua ning käivitada üksteisest sõltumatult.

- Mikroteenuste andmed peavad olema privaatsed, see tähendab teised teenused peavad küsima infot läbi teenuse, mitte otse baasist.
- Mikroteenused peavad olema väiksed ning vastutama ühe kindla funktsionaalsuse eest.
- Andmed salvestada andmebaasi, mitte rakenduse mällu.
- Südmuspõhine suhtlus.
- Dokumenteerida kõiki teenuseid ühises kohas
- Jagada koormust mikroteenuste vahel
- Mikroteenuste vaheliseks suhtluseks kasutada universaalseid sõnumitüüpe (json, rest)
- Mikroteenuste registreerimine ühes keskses kohas, et lihtsustada konfiguratsiooni.
- Versioneerida mikroteenuseid, sellega tagatakse süsteemi töötamine uue funktsionaalsuse lisamisel.
- Teenuste vaheline suhtlus peab käima läbi dokumenteeritud kanalite.
- Mikroteenuste logisid tuleb hallata ühest kesksest kohast (näiteks Elastic Stack)
- Mikroteenuste monitoorimiseks kasutada ühte keskset kohta.
- Hoida mikroteenused sessioonivabana.

4 Zachmani raamistik nutikassa näitel

Antud töö käigus kirjeldatakse ära Joonis 3 punaseks märgitud osad. Zachmani raamistikust on lähemalt kirjeldatud peatükis 2.

Tasand	**MIS** Andmed	**KUIDAS** Funktsioonid	**KUS** Paiknemine, võrk	**KES** Inimesed	**MILLAL** Aeg	**MIKS** Motivatsioon
Planeerija vaade	Oluliste mõistete ja objektide loend	Põhiliste toimimis-protsesside loend	Organisatsiooni eri üksuste asukohad	Võtmerühmad, töötajate loend	Olulisemate Sündmuste loend	Toimimis-eesmärgid ja -strateegiad
Omaniku vaade	Kontseptuaalne andmemudel	Toimimisprotsessi de mudel	Logistika skeem	Töövoog, töötajate vastutused (*workflow*)	Protsesside stsenaariumid (master *plan*)	Toimimisplaan (äriplaan)
Projekteerija vaade	Loogiline andmemudel (ERD)	Rakenduste arhitektuur	Hajussüsteemi arhitektuur	Kasutajaliides-te arhitektuur	Protsesside struktuur	Toimimisreeglite mudelid
Ehitaja vaade	Füüsiline andmemudel	Süsteemi projekt (spetsifikatsioon)	Tehnoloogia arhitektuur	Kasutajaliides-te disain	Juhtimis-struktuur	Toimimisreeglite disain
Alltöövõtja vaade	Andmete struktuuri Kirjeldus	Programmi disain	Võrgu-arhitektuur	Turbe arhitektuur, kasutajaõigu-sed	Ajalise seotuse määratlus	Toimimisreeglite spetsifikatsioon
Toimiv organisatsioon	Tegelikud andmed	Programmi kood	Võrk, süsteemide paiknemine	Pädevad töötajad	Tegevustsüklid	Rakendatud toimimisreeglid

Joonis 3: Realiseeritavad Zachmani raamistiku osad

4.1 Planeerija vaade

4.1.1 Oluliste mõistete ja objektide loetelu

Selles vaates loetletakse süsteemi ärilised objektid kõige üldisemal tasemel.

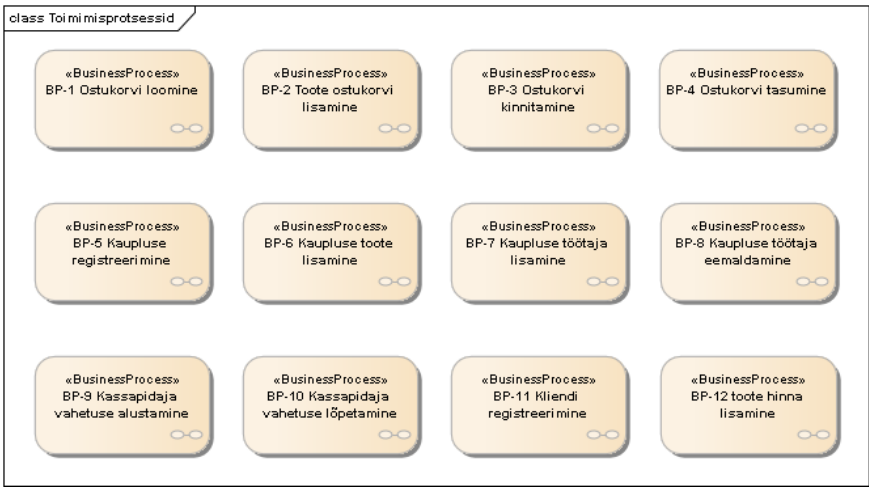
Põhilised äriobjektid:

- Kauplus
- Kaupluse toode
- Kaupluse kampaania
- Toote hind
- Ostukorv
- Tehing
- Klient
- Roll

4.1.2 Põhiliste toimimisprotsesside loend

Selles vaates loetletakse süsteemi põhilised äriprotsessid.

Süsteemis põhiprotsessioks on kaupluses toodete ostmise ning maksmisega seotud protsessid.



Joonis 4: Põhiliste toimimisprotsesside loend

4.1.3 Organisatsiooni eriüksuste asukohad

Nutikassa süsteem asub interneti võrgus Amazoni serveritest. Mobiilirakendus on kättesaadav Apple AppStorest.

4.1.4 Võtmerühmad, töötajate loend

- Kaupluse juhataja
- Kaupluse toodete haldaja
- Müüja

4.1.5 Olulisemate sündmuste loetelu

- Kliendi registreerimise käsk
- „Klient on registreeritud“ sündmus
- „Ostukorv on loodud“ sündmus

- Ostukorvi loomise käsk
- „Kassapidaja vahetus on lõpetatud“ sündmus
- Kassapidaja vahetuse lõpetamise käsk
- „Toode on ostukorvi lisatud“ sündmus
- Toote ostukorvi lisamise käsk
- „Ostukorv on kinnitatud“ sündmus
- Ostukorvi kinnitamise käsk
- „Ostukorv on makstud“ sündmus
- Kaupluse registreerimise käsk
- „Kauplus on registreeritud“ sündmus
- Kaupluse töötaja lisamise käsk
- „Kaupluse töötaja on registreeritud“ sündmus
- Kaupluse töötaja eemaldamise käsk
- „Kaupluse töötaja on eemaldatud“ sündmus

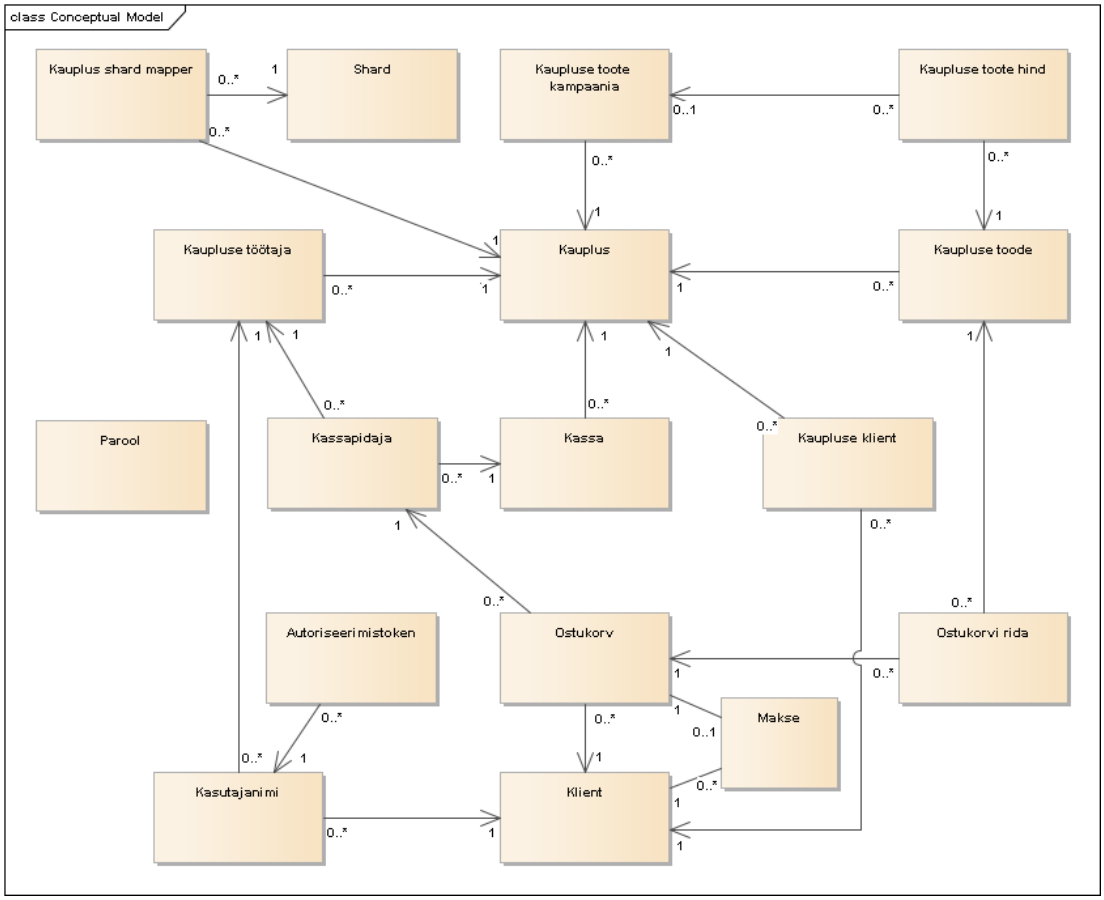
4.1.6 Toimimiseesmärgid

Nutikassa eesmärk on pakkuda mugavat ja kiiret lahendust kauplustes kauba eest tasumiseks. Nutikassa on suunatud väikestele ja uutele ettevõtetele, kes soovivad kiirelt ja mugavalt oma äri püsti panna.

4.2 Omaniku vaade

4.2.1 Kontseptuaalne andmemudel

Kontseptuaalne andmemudel on kõige üldisem mitte-tehniline mudel. Tänu mudeli lihtsusele kasutatakse seda vajalikele huvigruppidele ülevaate andmiseks. Lihtsuse pärast ei näidata siin spetsiifilisemaid atribuute aga protseduure. [3]



Joonis 5: Kontseptuaalne andmemudel

Kirjeldus

Kauplusel võib olla mitu kassat, toodet kampaaniat ning töötajat. Kaupluse tabelis hoitakse kaupluse infot. Kaupluse toote tabelis hoitakse infot toote kohta ning on seotud toote hinnaga. Kaupluse toote kampaania on seotud kaupluse toote ning hinnaga.

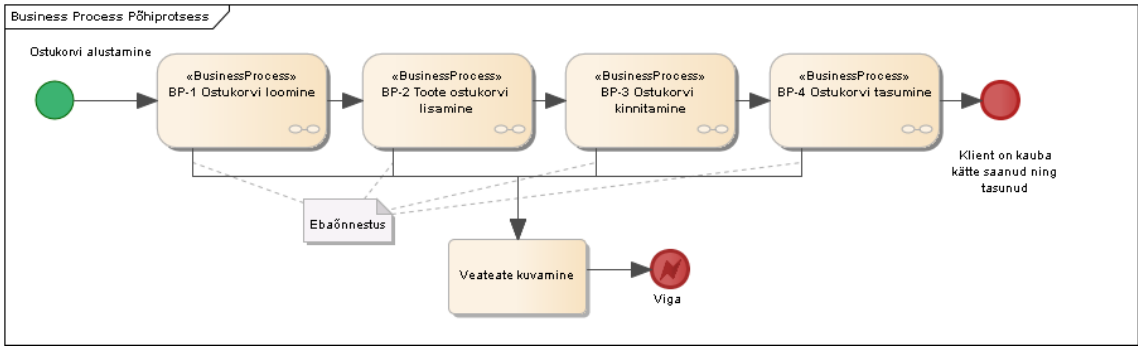
Kassa tabelis hoitakse infot kaupluses olevate kassade kohta. Kauplusel võib olla mitu kassat. Kassapidaja tabelis hoitakse infot töötaja vahetusest ning mis seob kassa ja töötaja. Töötaja tabelis hoitakse infot töötaja kohta. Ostukorvil on viide kassapidajale, kes kassas oli ning kliendile, kellele ostukorv kuulus. Kaupluse klient on vahetabel püsikliendi info hoidmiseks, mis seob kliendi ja kaupluse. Ostukorvi rida on vahetabel, mis seob iga ostetud kaupluse toote ostukorviga. Makse kirjeldab kliendi makseinfot ning omab viidet kliendile ning ostukorvile.

Kliendil on üks kasutajanimi süsteemi sisselogimiseks. Kasutajanimega on seotud autoriseerimistoken, mis hoiab süsteemi kasutaja sessiooni infot. Parool on tabel parooliräsi hoidmiseks. Parooliräsi ei ole seotud ühegi kindla kliendiga.

4.2.2 Toimimisprotsesside mudel

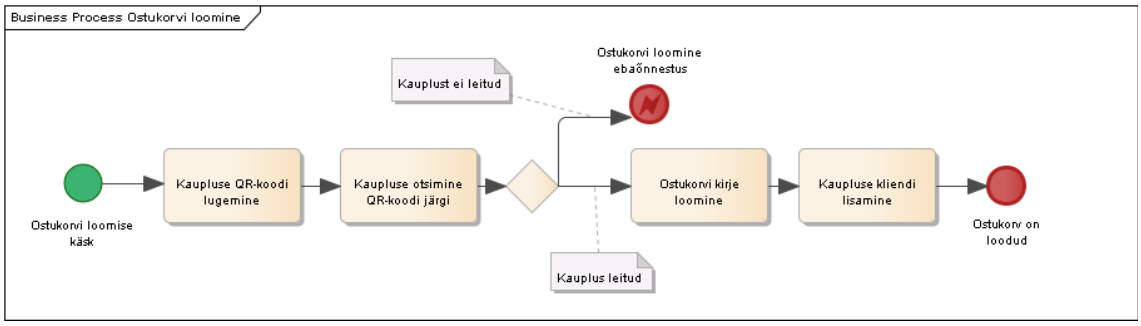
Toimimisprotsesside (äriprotsesside) mudel kirjeldab ettevõttes toimuvaid protsesse. Äriprotsess on tegevuste kogum, mis pakuvad kindlat teenust või toodet klientidele.

Nutikassa põhiprotsess on kliendi ostusoovitamine. Protsess saab alguse ostukorvi loomisest ning lõppeb ostukorvi tasumisega.



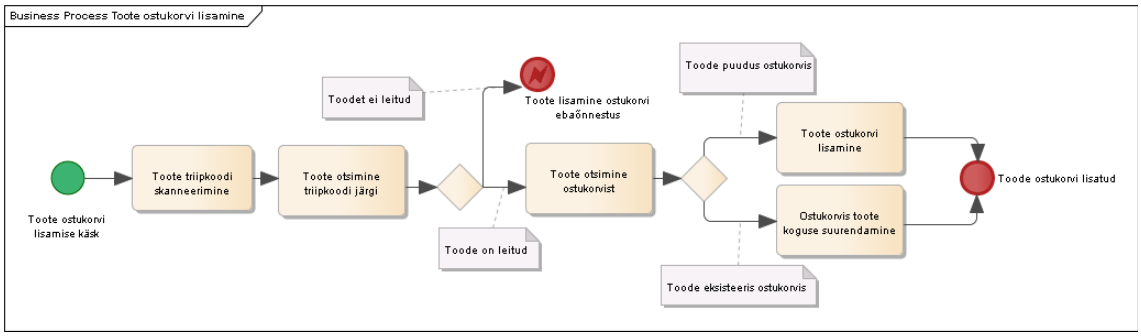
Joonis 6: Põhiprotsess

4.2.2.1 BP-1 Ostukorvi loomine



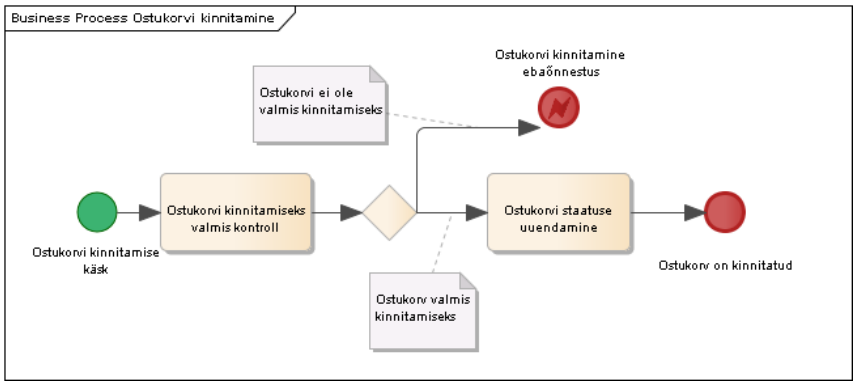
Joonis 7: BP-1 Ostukorvi loomine

4.2.2.2 BP-2 Toote ostukorvi lisamine



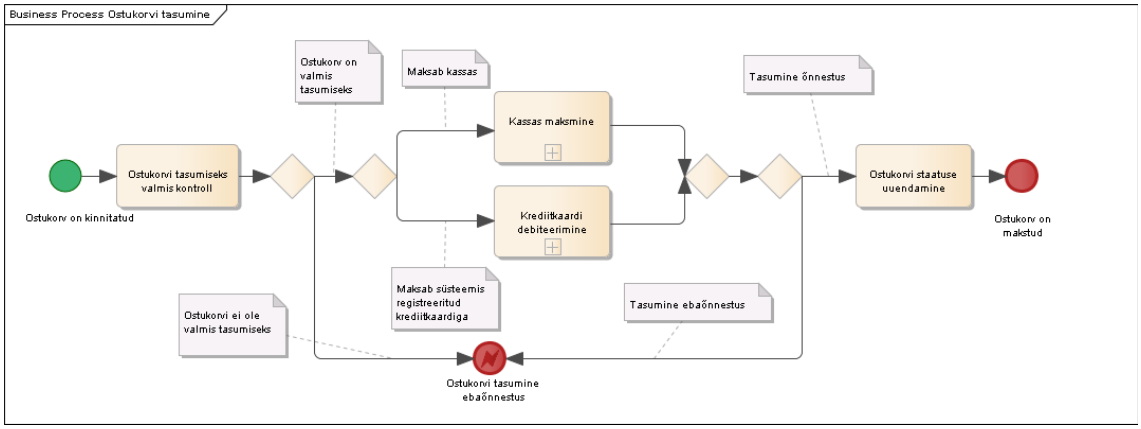
Joonis 8: BP-2 Toote ostukorvi lisamine

4.2.2.3 BP-3 Ostukorvi kinnitamine



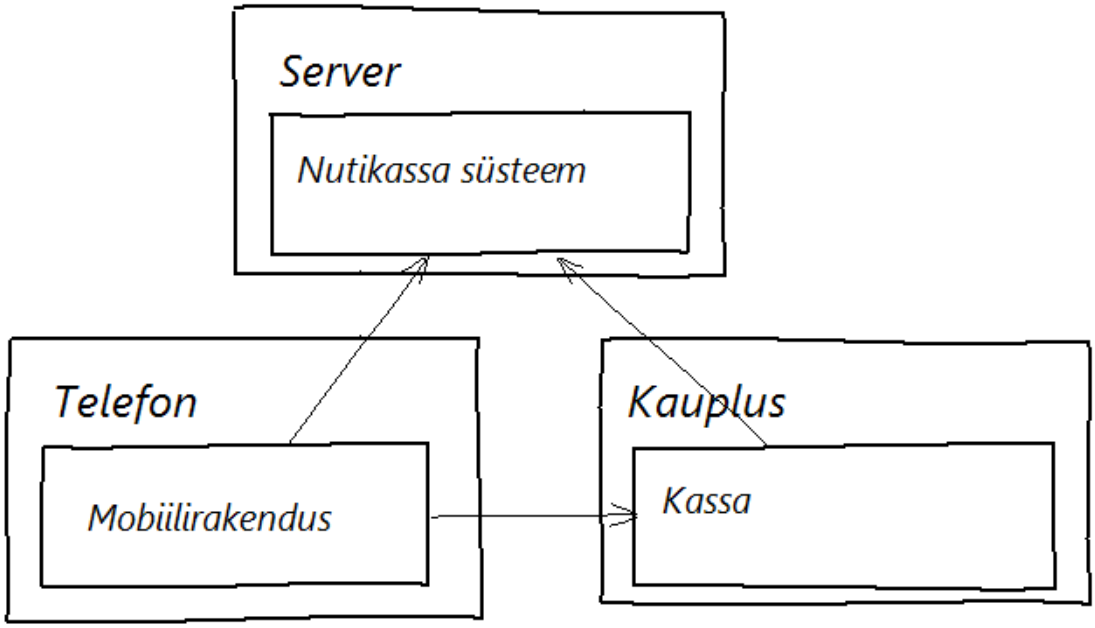
Joonis 9: BP-3 Ostukorvi kinnitamine

4.2.2.4 BP-4 Ostukorvi tasumine



Joonis 10: BP-4 Ostukorvi tasumine

4.2.3 Logistika skeem

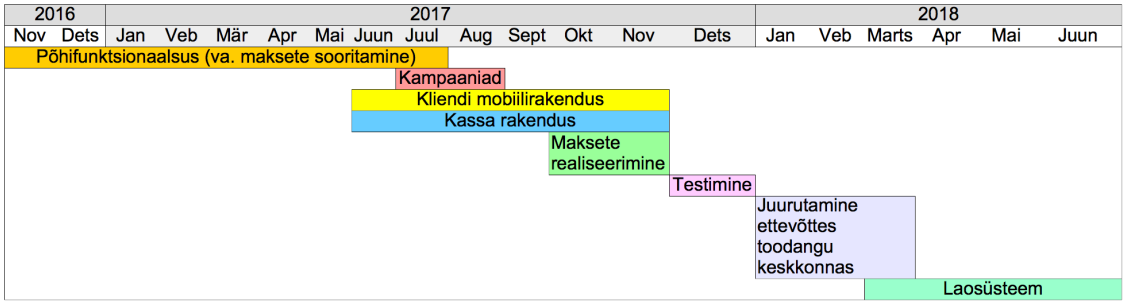


Joonis 11: Loogika skeem

4.2.4 Töötajate vastutused

- Kaupluse juhataja – kaupluses kõige suurema võimuga roll. Vastutab töötajate süsteemi registreerimise eest ning nendele õiguste andmise eest.
- Kaupluse toodete haldaja – vastutab kaupluses toodete lisamise, kustutamise ning hinnastamise eest.
- Müüja – vastutab toodete müügi eest.

4.2.5 Süsteemi ajaline plaan



Joonis 12: Süsteemi ajaline plaan

4.3 Turuanalüüs

Turuanalüüs on üks osa äriplaanist. Käesolev töö keskendub süsteemi arhitektuurile, mitte ettevõtte äriplaanile, ning sellepärast on äriplaanist lähemalt uuritud turgu, et kindlaks määrata võimalikku koormust süsteemile.

4.3.1 Turusegmendid

Eestis tegutseb hulgaliselt kaupade müügiga tegutsevaid kaupluseid. Kaupluste arv on tihedalt seotud elanikkonna suurusega ning nende maksejõuga.

Nutikassa on keskendunud kassasüsteemi pakkumisega jaemüügiga tegelevatele kauplustele nii Eestis kui välismaal.

Kassasüsteemi turu võib klientide järgi jagada järgmisteks segmentideks:

1. kaubandusketid
2. üksikud kauplused

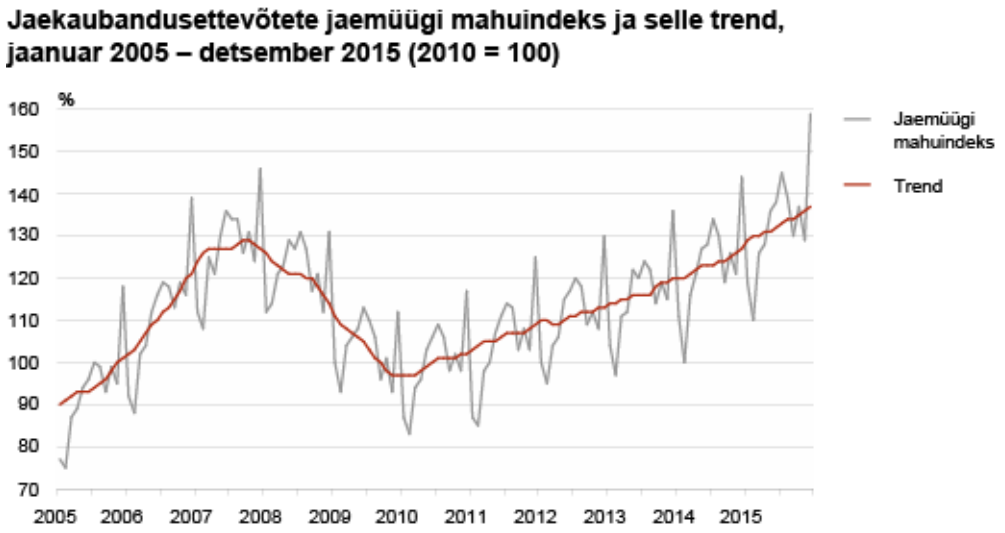
4.3.2 Sihtturu strateegia

Nutikassa peamiseks segmendiks on üksikud kauplused, kes soovivad kaasaegset kassasüsteemi. Klientide peamiseks mõjutajateks on selle lihtsus ning valmis lahendus. Mõjutada saab ka täieliku hinnainformatsiooniga, andes kindluse uue kassasüsteemi mõistlikkuse kohta.

4.3.3 Sihtturu kasvutempo

Eestis on jaekaubanduse müügitulu kasvanud alates 2010 aasta algusest.[4] Samamoodi on kasvanud ka euroopa jaemüük. [5] .

2016. aasta detsembris müüdi Eestis 510,7 miljoni euro eest kaupa, see teeb 390 euro inimese kohta kuus. [6] .



Joonis 13: Jaemüügi kasv

Seoses jaemüügi kasvuga on näha võimaliku kliendibaasi kasvu nii Eestis kui euroopas.

4.3.4 Turuanalüüsi mõju süsteemile

Turuanalüüsist tuli välja, et keskmiselt kulutab inimene 390 eurot kuus. Keskmise koormuse saamiseks oletame, et ühe ostu suurus oli keskmiselt 10 eurot, selle oletuse põhjal, et 100% turuosa suuruse korral peab süsteem olema võimeline teenindama keskmiselt 1173 ostukorvi minutis. Punkt 4.1.6 sõnastab, et nutikassa on suunatud just uutele ja väikestele ettevõtetele ja turuosa on väike, siis realistlik koormus süsteemile võib jääda 20-200 ostukorvi minutis vahele.

Turuanalüüsist tuli välja, et kõige suuremat koormust saab süsteemis ostukorvi teenus, toodete teenus ning maksete teenus.

4.4 Nõuded

4.4.1 Mittefunktsionaalsed nõuded

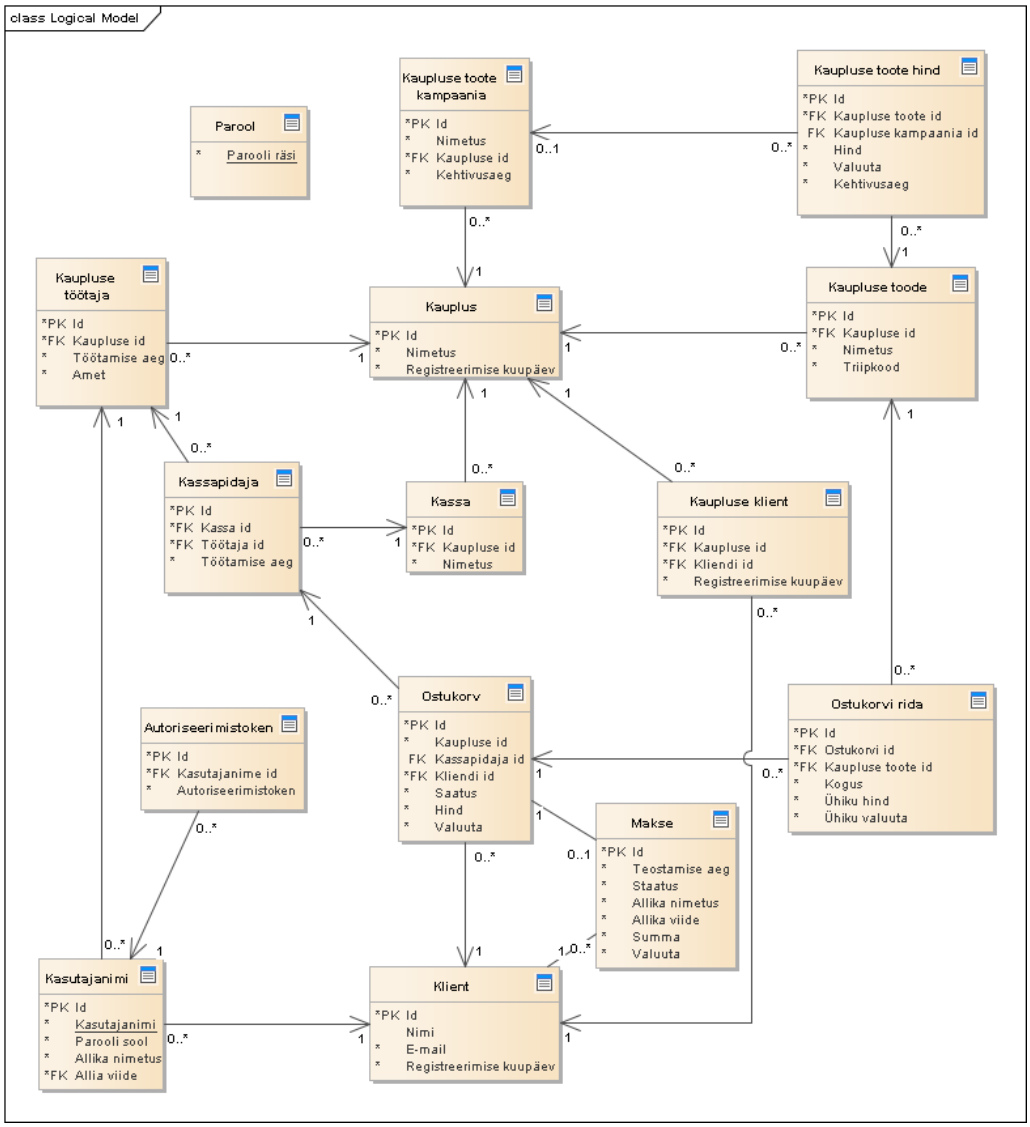
- Süsteem peab olema kättesaadav 100% ajast. Süsteemis ei tohi olla seisakuid, sest süsteemi kasutatakse erinevates ajatsoonides pidevalt.
- Süsteem peab töötama 7 päeva nädalas 24 tundi päevas.
- Süsteem peab tulema toime väiksemate vigadega (valed sisend parameetrid kasutajalt)

- Süsteem peab olema võimeline taluma kasutajate mitmekordset kasvu.
- Süsteemi peab olema võimalik laiendada uute funktsionaalsustega nii, et olemasolev funktsionaalsus säiliks.
- Süsteem peab tulema toime uuendamata mobiilirakendustega.
- Süsteemi tööd peab olema võimalik järgida ning tõrgetest tuleb teavitada vastutavaid isikuid.
- Süsteemile peab olema võimalik lisada ressursse ka kõige suuremal koormuse perioodil.
- Kasutajal peab olema ligipääs süsteemile üle interneti.
- Süsteem peab olema multikeelne.
- Süsteem ei tohi lasta kauplustel näha kliendi isikuandmeid.
- Süsteem ei tohi lubada klientidel näha teiste klientide isikuandmeid.
- Süsteem ei tohi lubada kolmandatel isikutel näha kliendi isikuandmeid.
- Süsteem peab toime tulema stabiilselt toime 1000 aktiivse kasutajaga.

4.5 Projekteerija vaade

4.5.1 Loogiline andmemudel

Loogiline andmemudel on kontseptuaalsest andmemudelist täpsem sisaldades endas tabeli atribuutide, sealhulgas primaarvõtmeid ja välisvõtmeid. Loogiline andmemudel ei sisalda andmebaasi spetsiifilisi kirjelduse, see tähendab, et puuduvad andmetüübid, triggerid, indeksid ning atribuutide nimetused ei ole tehnilised. [17]

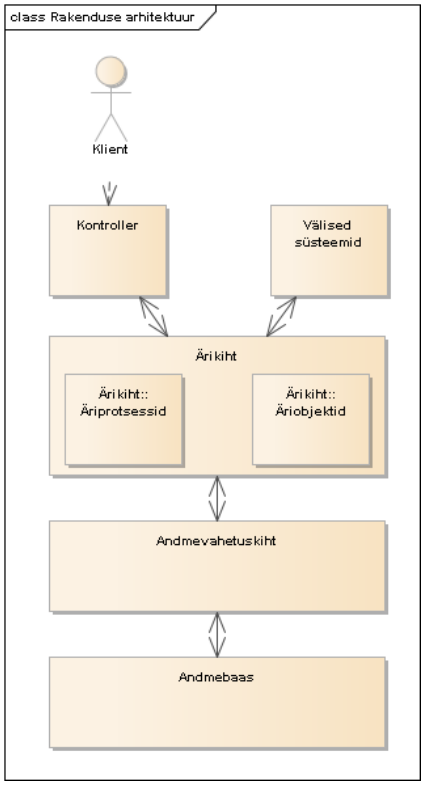


Joonis 14: Loogiline andmemudel

4.5.2 Rakenduste arhitektuur

Rakenduste arhitektuur koosneb 5 erinevast kihist:

- Kontroller – vastutab sissetulevaate päringute vastuvõtmise ning vastuse saatmise eest
- Ärikiht – vastutab äriprotsesside toimimise eest.
- Välised süsteemid – välised teenused, mida on vaja äriprotsesside toimimiseks.
- Andmevahetuskiht – vastutab andmete andmebaasi transpordi eest.
- Andmebaas – vastutab andmete talletamise eest.



Joonis 15: Rakenduse sisemine arhitektuur

4.5.3 Mikroteenuste hajussüsteemi arhitektuur

Turuanalüüsist selgus, et võimalik kasutajate arv võib tulevikus kasvada kiiresti ja ootamatult. Sellest tulenevalt on välistatud ühe suure monoliitrakenduse loomine. Süsteem peab olema laiendatav horisontaalselt, et jagada koormust suurema arvu serverite vahel.

Süsteem on jagatud üksteisest sõltumatuteks mikroteenusteks nii, et jääksid üksikud üksteisest eraldiseisvad rakendused.

Probleemis / pudelikaelad hajusarhitektuuris:

1. teenuste paljusus
2. konfiguratsiooni haldus
3. logide haldus
4. mikroteenuste vaheliste rest päringute paljusus
5. mikroteenuste sõltuvuste paljusus

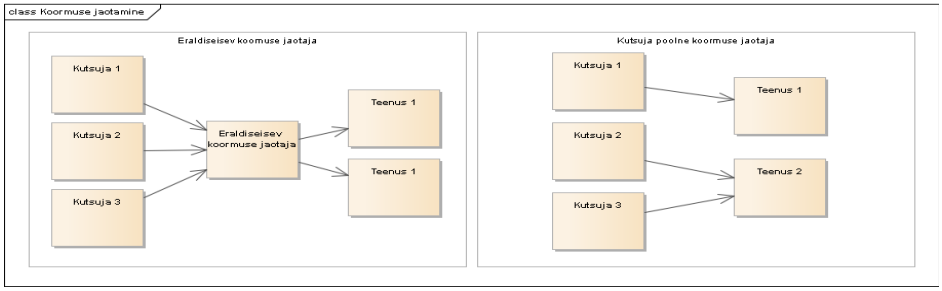
4.5.3.1 Teenuste register

Mikroteenuste paljusus teeb rakenduste haldamise keeruliseks. Selle lahenduseks on teenuste register. Kõik teenused, registreerivad ennast teenuste registris, sealhulgas ka kõik replitseeritud teenused. Teenuste register annab hea ülevaate teenustest ning nende aadressidest. [13]

Teenuste register võimaldab mikroteenustel küsida teiste mikroteenuste kohta infot ning saata omakorda nendesse päringuid. See lahendus lihtsustab konfigureerimist muutes selle dünaamilisemaks ning läbipaistvamaks. Teenuste register lihtsustab replitseeritud rakenduste juurde lisamist, sest uue replitseeritud rakenduse juurde lisamisel registreeritakse see automaatselt teenuste registris ning on teised mikroteenused saavad koheselt sellele päringuid saata. [13]

Joonis 16: Teenuste registri kontseptuaalne andmemudel

Teenuste register lahendab osaliselt ka rest päringute paljususe probleemi. Kuna teenuste register omab kõikide teenuse aadresse ning mikroteenused saavad neid küsida, siis on võimalik kasutada mikroteenustes päringu kutsuja poolset koormuse jaotamise (Client-side load balancing) lahendusi. See tähendab, et pole vaja eraldiseisvaid koormuse jaotamise proxy rakendusi (*Load balancer*) vaid see töö tehakse juba kutsuja pool ära ning päring saadetakse otse teisele mikroteenusele. [14]

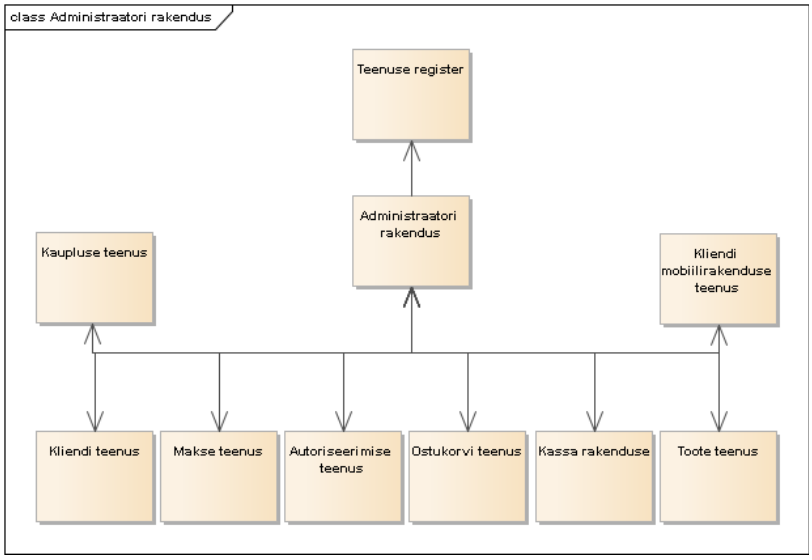


Joonis 17: Eraldiseisev koormusejaotaja vs kutsuja poolne koormuse jaotaja

4.5.3.2 Administraatori rakenduse (konfiguratsiooni rakendus)

Mikroteenuste paljusus muudab nende konfigureerimise keeruliseks. Eraldiseisvaid mikroteenuseid on palju, süsteemi keskkondasi on erinevaid (arenduskeskkond, testkeskkond, toodangu eelne keskkond, toodangu keskkond). Lisaks tulevad erinevad rajoonid (ida-euroopa, lääne-euroopa, ameerika), mis vajavad omakorda eraldi konfiguratsiooni.

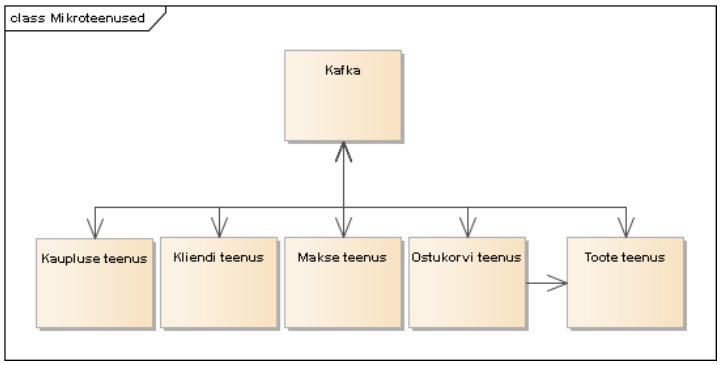
Konfiguratsiooni paljususe lahendust pakub administraatori rakendus, mis koondab kokku kõik konfiguratsiooni failid ning mikroteenused käivad küsimas neid ühest kesksest kohast. Mirkoteenused pöörduvad käivitudes administraatori rakenduse poole läbi *REST* liidese.



Joonis 18: Administraatori rakenduse hajusarhitektuur

4.5.3.3 Mikroteenuste hajusarhitektuur

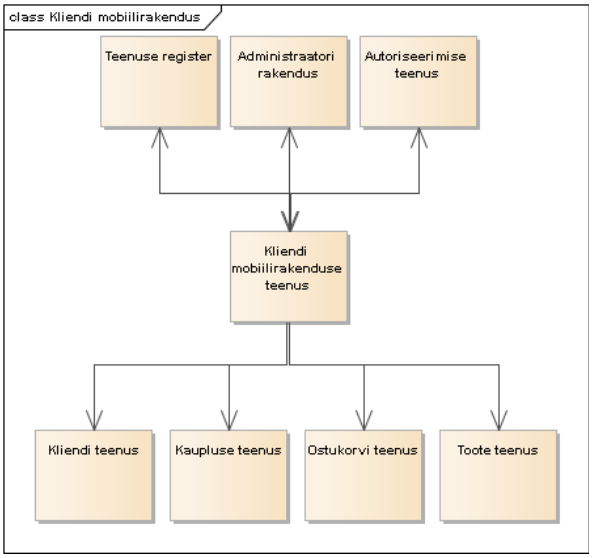
Mikroteenustel käib suhtlus tavaliselt läbi http rest päringute. See võib põhjustada suurel hulgal päringute rägastikke, millest võib olla raske hiljem aru saada. Seepärast on mikroteenustel heaks tavaks saanud sündmuse põhine lahendus. Sündmuste põhisel lahendusel kasutatakse avalda-kuula lähenemist, see tähendab, et kui ühes mikroteenuses toimub mingi sündmus, siis see sündmus saadetakse Kafka rakendusele. Teised mikroteenused kuulavad Kafka rakendusse saadetud sündmusi ning huvitatud mikroteenused saavad selle sündmuse kätte ning käivitavad omalt poolt vastavad protsessid. Error: Reference source not found



Joonis 19: Mikroteenuste hajusarhitektuur

4.5.3.4 Kliendi mobiilirakenduse hajusarhitektuur

Turvalisuse huvides ei ole mikroteenused välisvõrgust otse kättesaadavad. Välisvõrgust tulnud päringud käivad läbi kliendi mobiilirakenduse teenuse. Kliendi mobiilirakendus saadab päringud edasi nendele teenustele, kuhu vaja.



Joonis 20: Kliendi mobiilirakenduse hajusarhitektuur

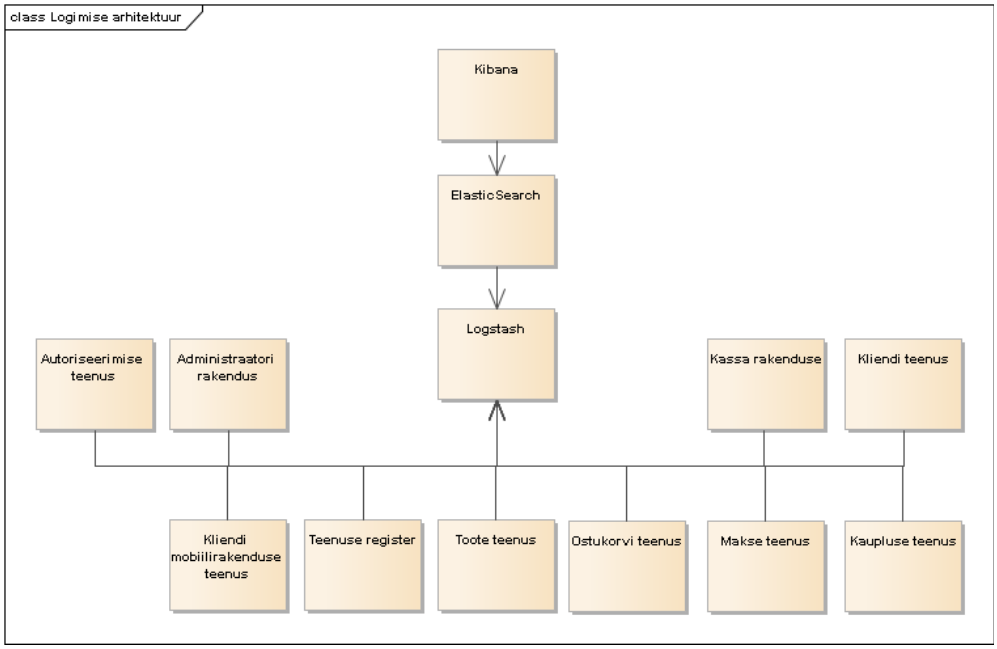
4.5.3.5 Rakenduse logide haldamine mikroteenuste arhitektuuris

Lahendusi rakenduse logide haldamiseks on mitmeid, käesolevas süsteemis kasutatakse firma elastic poolt loodud tarkvara logstash, elasticsearch ja kibana.

Logstash on süsteem, mis laeb kõikide rakenduste logid ühte kohta kokku ning indekseerib need.

Elasticsearch on otsingumootor, millega on võimalik otsida ja filtreerida logi kirjeid.

Kibana on graafiline veebirakendus, mille kaudu kasutaja saab sisestada elasticsearch päringuid ning kuvab tulemusi.



Joonis 21: Rakenduse logide haldamine

4.6 Ehitaja vaade

4.6.1 Füüsiline andmemudel

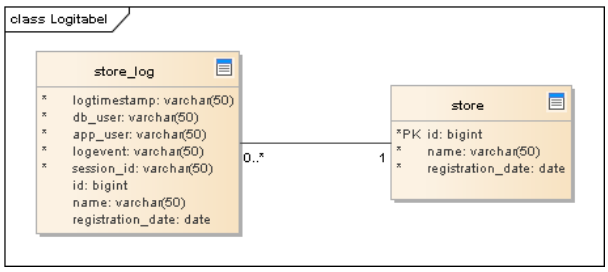
Füüsiline andmemudel on edasi arendatud loogilisest andmemudelist. Füüsiline andmemudel sisaldab tabeleid, väljasid, andmetüüpe, indekseid, välisvõtmeid, trigereid ning vaateid. Füüsiline andmemudel tuleb hoida kooskõlas loogilise andmemudeliga – nad ei tohi olla üksteisega vastuolus. Erinevalt loogilisest andmemudelist, on füüsiline andmemudel inglise keeles. [3]

Füüsiline andmemudel on igal mikroteenusel üksteisest lahus. Nutikassa süsteemis on potentsiaalne andmemaht suurt. Igal mikroteenuse andmebaasis on talle vajalik alamosa füüsilisest andmemudelist. Sellist alamosa võib nimetada ka alamsüsteemiks, aga antud töös on kasutatud andmebaasi mõistet, et rõhutada alamosade üksteisest sõltumatust.

4.6.1.1 Andmebaasi auditlogi

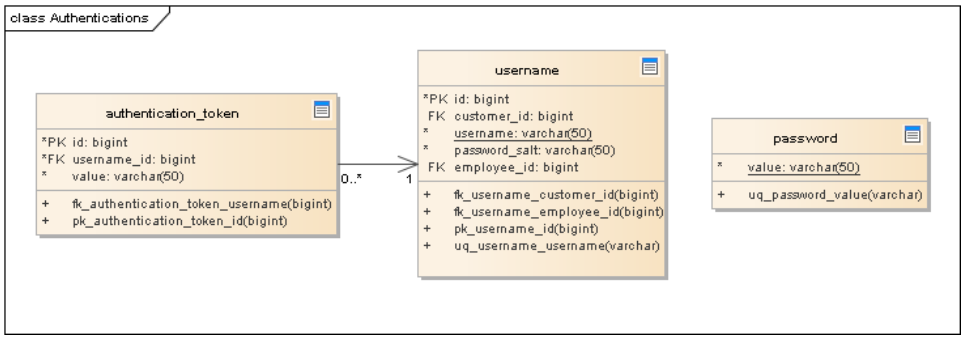
Levinud nõue kaasaegsetes andmebaasides on andmebaasi muudatuste logimine – mis info milleks muudeti, millal muudeti ning kes muutis. Selle saavutamiseks tehakse eraldi tabel (logitabel) ajaloo hoidmiseks. Logitabelid on süsteemi audittabelid ning nad ei osale äriprotsessides ja seepärast ei kuvata neid füüsilises andmemudelis. [7]

Nutikassa logitabelis on info lisaks kirje enda infole veel logimise aeg, muutja, muutuse liik ning päringu id. Päringu id kaudu saab siduda konkreetse muutuse rakenduse protsessiga ning annab võimaluse rakenduse logist leida üles vastava protsessi kirjed.



Joonis 22: Logitabeli näide

4.6.1.2 Autoriseerimise teenuse andmebaas



Joonis 23: Atentimise andmebaasi füüsiline mudel

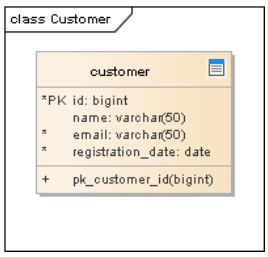
Tähistused
(PK) - primaarvõti

(FK) – välisvõti

Tabel 41. Autentimise teenuse andmebaasi kirjeldus

Nimetus loogilises andmemudelis	Nimetus füüsilises andmemudelis	Kirjeldus
Autoriseerimistoken	authentication_token	Tabel, kus hoitakse autoriseerimise võtmeid
Id	id (PK)	Automaatselt genereeritud unikaalne number
Kasutajanime id	username_id (FK)	Kasutajanimi, millega autoriseerimisevõti on tehtud
Autoriseerimistoken	value	Genereeritud autoriseerimise võti
Kasutajanimi		Tabel, kus hoitakse kasutajanimesid süsteemis autentimiseks
Id	id (PK)	Automaatselt genereeritud unikaalne number
Kliendi id	customer_id (FK)	Klient, kellele kasutajanimi kuulub
Kasutajanimi	username	Kasutajanimi, millega klient/töötaja süsteemis ennast tuvastab
Parooli sool	password_salt	Juhuslik väärtus, mis paroolile juurde lisatakse
Töötaja id	employee_id (FK)	Töötaja, kellele kasutajanimi kuulub
Parool	password	Tabel, kus hoitakse parooli räsisid
Parooli räsi	value	Parooli räsi

4.6.1.3 Kliendi teenuse andmebaas



Joonis 24: Kliendi andmebaasi füüsiline mudel

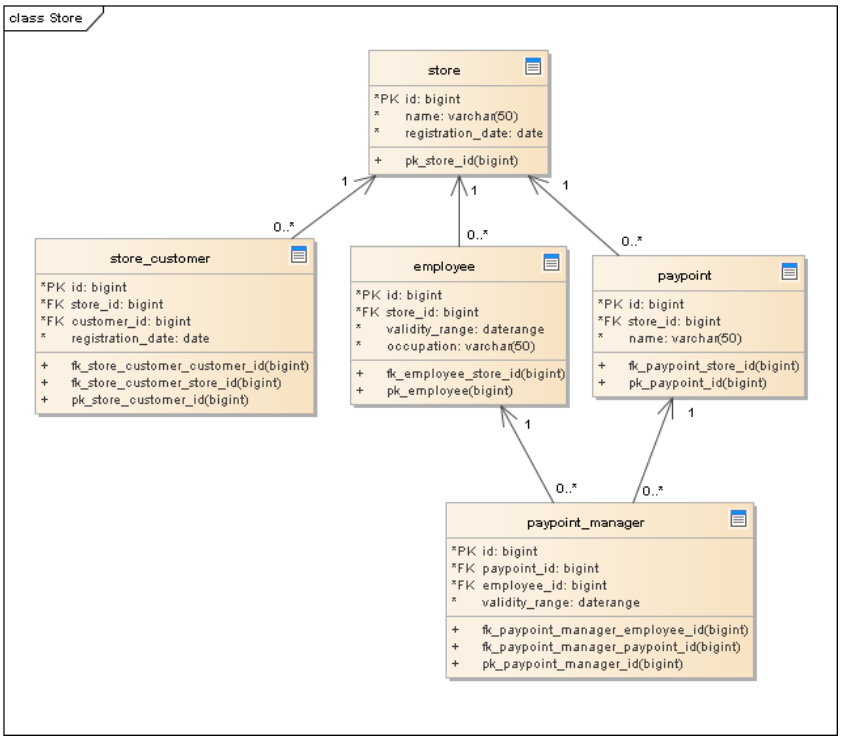
Tähistused
(PK) - primaarvõti

(FK) – välisvõti

Tabel 42. Kliendi teenuse andmebaasi kirjeldus

Nimetus loogilises andmemudelis	Nimetus füüsilises andmemudelis	Kirjeldus
Klient	customer	Tabel, kus hoitakse kliendi infot
Id	id (PK)	Automaatselt genereeritud unikaalne number
Nimi	name	Kliendi nimi
E-mail	email	Kliendi e-mail
Registreerimise kuupäev	registration_date	Kliendi registreerimise kuupäev

4.6.1.4 Kaupluse teenuse andmebaas



Joonis 25: Kaupluse andmebaasi füüsiline mudel

Tähistused
(PK) - primaarvõti

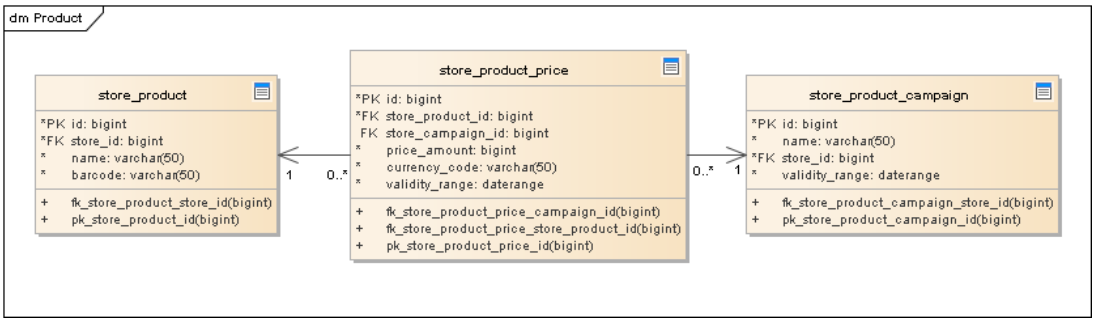
(FK) – välisvõti

Tabel 43. Kaupluse teenuse andmebaasi kirjeldus

Nimetus loogilises andmemudelis	Nimetus füüsilises andmemudelis	Kirjeldus
Kauplus	store	Tabel, kus hoitakse kaupluse infot
Id	id (PK)	Automaatselt genereeritud unikaalne number
Nimetus	name	Kaupluse nimetus
Registreerimise kuupäev	registration_date	Kaupluse registreerimise kuupäev
Kaupluse klient	store_customer	Tabel, kus hoitakse infot kaupluse püsikliendikliendi infot

Id	id (PK)	Automaatselt genereeritud unikaalne number
Kaupluse id	store_id (FK)	Viide kauplusele
Kliendi id	customer_id (FK)	Viide kliendile
Registreerimise kuupäev	registration_date	Püsikliendi registreerimise kuupäev
Töötaja	employee	Tabel, kus hoitakse töötaja infot
Id	id (PK)	Automaatselt genereeritud unikaalne number
Kaupluse id	store_id (FK)	Viide kauplusele
Töötamise aeg	validity_range	Töötaja töötamise aeg
Amet	occupation	Amet, milles töötaja töötas
Kassa	paypoint	Tabel, kus hoitakse kaupluses olevate kassade infot
Id	id (PK)	Automaatselt genereeritud unikaalne number
Kaupluse id	store_id (FK)	Viide kauplusele
Nimetus	name	Kassa nimetus
Kassapidaja	paypoint_manager	Tabel, kus hoitakse kassapidaja vahetuse infot
Id	id (PK)	Automaatselt genereeritud unikaalne number
Kassa id	paypoint_id (FK)	Viide kassale
Töötaja id	employee_id (FK)	Viide töötajale kes kassas töötas
Töötamise aeg	validity_range	Kassas töötamise aeg

4.6.1.5 Toote teenuse andmebaas

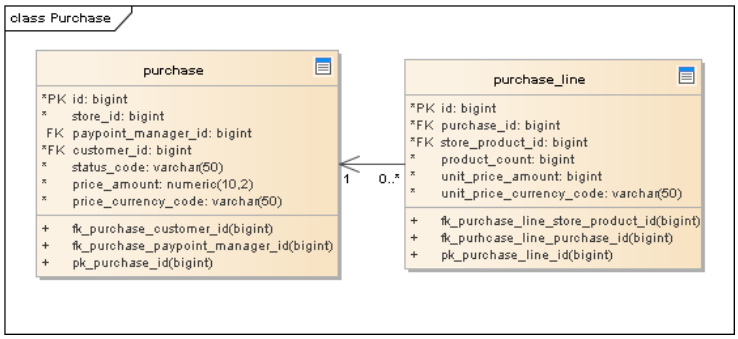


Joonis 26: Toote andmebaasi füüsiline mudel

Tabel 44. Toote teenuse andmebaasi kirjeldus

Nimetus loogilises and-memudelis	Nimetus füüsilises and-memudelis	Kirjeldus
Kaupluse toode	store_product	Tabel, kus hoitakse kaupluses müügil olevaid tooteid
Id	id (PK)	Automaatselt genereeritud unikaalne number
Kaupluse id	store_id (FK)	Viide kauplusele
Nimetus	name	Toote nimetus
Triipkood	barcode	Toote triipkood
Kaupluse toote hind	store_product_price	Tabel, kus hoitakse toote hinna infot
Id	id (PK)	Automaatselt genereeritud unikaalne number
Kaupluse toote id	store_product_id (FK)	Viide kaupluses olevale tootele
Kaupluse kampaania id	store_campaign_id (FK)	Viide kaupluses kehtivale kampaaniale
Hind	price_amount	Toote hind
Valuuta	currency_code	Toote valuuta
Kehtivusaeg	validity_range	Hinna kehtivusaeg
Kaupluse toote kampaania	store_product_campaign	Tabel, kus hoitakse kaupluses kehtivaid kampaaniaid
Id	id (PK)	Automaatselt genereeritud unikaalne number
Nimetus	name	Kampaania nimetus
Kaupluse id	store_id (FK)	Viide kauplusele
Kehtivusaeg	validity_range	Kampaania kehtivusaeg

4.6.1.6 Ostukorvi teenuse andmebaas



Joonis 27: Ostukorvi andmebaasi füüsiline mudel

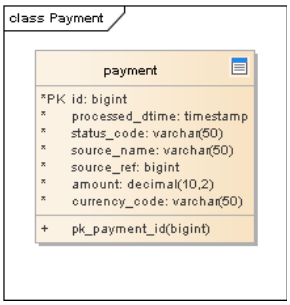
Tähistused
(PK) - primaarvõti

(FK) – välisvõti

Tabel 45. Ostukorvi teenuse andmebaasi kirjeldus

Nimetus loogilises andmemudelis	Nimetus füüsilises andmemudelis	Kirjeldus
Ostukorv	purchase	Tabel, kus hoitakse ostude infot
Id	id (PK)	Automaatselt genereeritud unikaalne number
Kaupluse id	store_id (FK)	Viide kauplusele, kus ostukorv loodud on
Kassa id	paypoint_manager_id (FK)	Viide kassale, kus ostukorvi eest tasuti
Kliendi id	customer_id (FK)	Viide kliendile, kes ostu eest tasus
Staatus	status_code	Ostukorvi olek
Hind	price_amount	Ostukorvi lõpphind
Valuuta	price_currency_code	Ostukorvi valuuta
Ostukorvi rida	purchase_line	Tabel, kus hoitakse infot ostukorvis olevate toodete kohta
Id	id (PK)	Automaatselt genereeritud unikaalne number
Ostukorvi id	purchase_id (FK)	Viide ostukorvile
Kaupluse toote id	store_product_id (FK)	Viide kaupluses olevale tootele
Kogus	product_count	Toote kogus
Ühiku hind	unit_price_amount	Toote ühikuhind
Ühiku valuuta	unit_price_currency_code	Valuuta

4.6.1.7 Tehingu teenuse andmebaas



Joonis 28: Tehingu andmebaasi füüsiline mudel

Tähistused
(PK) - primaarvõti

(FK) – välisvõti

Tabel 46. Tehingu teenise andmebaasi kirjeldus

Nimetus loogilises and-memudelis	Nimetus füüsilises and-memudelis	Kirjeldus
Makse	payment	Tabel, kus hoitakse infot maksete kohta
Id	id (PK)	Automaatselt genereeritud unikaalne number
Teostamise aeg	processed_dtime	Makse teostamise aeg
Staatus	status_code	Makse olek
Allika nimetus	source_name	Makse allikas
Allika viide	source_ref	Makse viide allikas
Summa	amount	Makse summa
Valuuta	currency_code	Makse valuuta

4.6.2 Süsteemi specifikatsioon

4.6.2.1 Süsteemis kasutatavad tehnoloogiad

Käesolevas töös on mikroteenused programmeeritud Javas. Andmebaasid on postgresql. Kasutatud on järgmisi raamistikke:

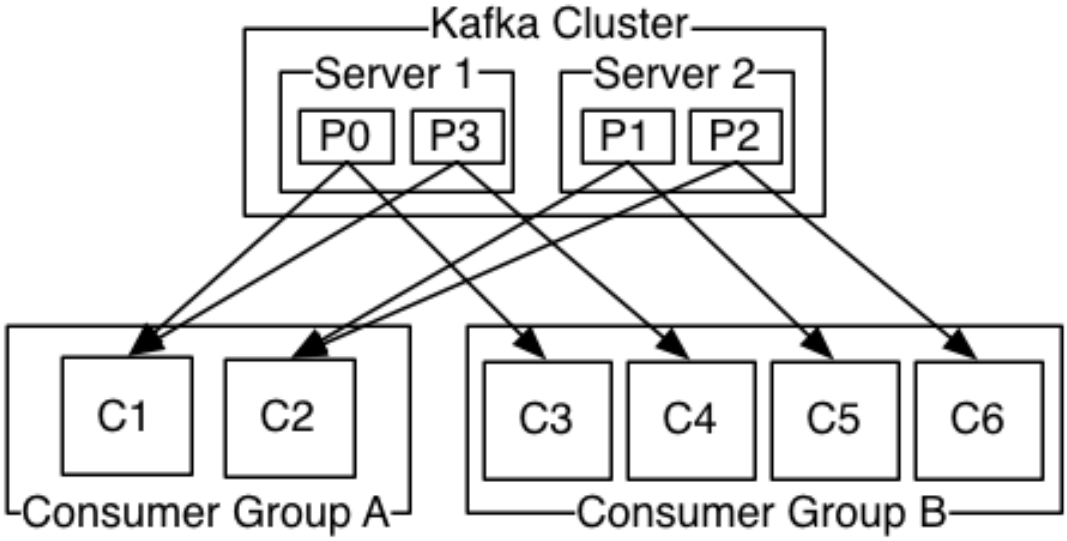
- Spring raamistik – võimas Java raamistik, mis pakub laiaulatusliku programmeerimise ja konfigureerimise mudelit.
- Spring boot – spring raamistiku moodul, mis paneb rakendusserveri (näiteks tomcat või jetty) rakendusega kaasa.
- Spring boot admin – andminrakendus springi rakenduste haldamiseks. [15]
- Spring cloud
    - spring-cloud-config – võimaldab rakendustel konfiguratsiooni küsida konfiguratsiooni serverist
    - spring-cloud-eureka – raamistik, mis annab võimekuse rakendustel registreerida ennast Eureka serveris
    - spring-cloud-feign – raamistik, mis lihtsustab rest päringute tegemist.
    - spring-clout-ribbon – kliendi poolne koormuse jagamise raamistik
- MyBatis – raamistik, mis suudab andmebaasi päringuid käivitada ning tulemused Java objektideks konverteerida
- Eureka – netflixi poolt loodud server, mis registreerib endasse mikroteenused. (teenuste register)
- Kafka – täpsem kirjeldus punktis 4.6.2.2.

4.6.2.2 Kafka

Kafka on laiendatav sõnumi-jagamise platvorm. Kafkat on võimalik kasutada mitmel viisik – sõnumi vahetus süsteemina, info talletamise süsteemina või voogude töötlemise süsteemina. Käesolevas töös on Kafkat kasutatud kui sõnumi vahetus süsteemina. [18]

Sõnumi vahetusel on kaks meetodit – ootejärjekord (queue) ja saada-ja-kuula (publish-subscribe). Ootejärjekord tähendab seda, et sõnum visatakse järjekorda ning järjekorra kuulaja töötleb sõnumeid ükshaaval. Sõnumis on järjekorras seni, kuni on töödeldud. Järjekorras olevaid sõnumeid saab töödelda ainult üks kuulaja. Saada-ja-kuula (publish-subscribe) korral saadetakse sõnum kõigile kuulajatele. Kafka seob need kaks meetodit. [18]

Kafka nimetab sisemiselt ootejärjekordi teemadeks (*topic*). Igal teemal võib olla mitu kuulajasgruppi, see võimaldab ühte sõnumit töödelda erinevate kuulajasgruppide (rakenduste) poolt üksteisest sõltumatult.[18]



Joonis 29: Kafka kuulajasgrupid [18]

4.6.2.3 Nutikassa rakenduste kirjeldus

Teenused:

- Autoriseerimise teenus – vastutab süsteemi kasutaja autoriseerimise eest. Füüsiline andmemudel on kirjeldatud punktis 4.6.1.2. Teenus salvestab endasse kasutajanime ja parooli. Kasutaja sisse logimisel kontrollib neid ning tagastab autoriseerimisetokeni (sessiooni tunnuse), millega süsteem edaspidi klienti tuvastab.

  Äriprotsessid, mille eest teenus vastutab:

  - Kasutaja autoriseerimine
  - Autoriseerimistokeni valideerimine

- Kaupluse teenus – vastutab kaupluste ning tema töötajate ja kassade eest. Füüsiline andmemudel on kirjeldatu punktis 4.6.1.4.

  Äriprotsessid, mille eest teenus vastutab:

  - BP-5 Kaupluse registreerimine
  - BP-7 Kaupluse töötaja lisamine
  - BP-8 Kaupluse töötaja eemaldamine
  - BP-9 Kassapidaja vahetuse alustamine
  - BP-10 Kassapidaja vahetuse lõpetamine

- Kliendi teenus – vastutab kliendi andmete salvestamise eest. Füüsiline andmemudel on kirjeldatud punktis 4.6.1.3.

  Äriprotsessid, mille eest teenus vastutab:

  - BP-11 Kliendi registreerimine

- Tehingute teenus – vastutab maksete teostamise eest. Füüsiline andemudel on kirjeldatud punktis 4.6.1.7.

Äriprotsessid, mille eest teenus vastutab:

- Maksete tegemine

- Ostukorvi teenus – vastutab ostukorvi loomise, ostukorvi toodete lisamise/eemaldamise ning ostukorvi eest tasumise eest. Füüsiline andmemudel on kirjeldatud puntkis 4.6.1.6.

  Äriprotsessid, mille eest teenus vastutab:

  - BP-1 Ostukorvi loomine
  - BP-2 Toote lisamine ostukorv
  - BP-3 Ostukorvi kinnitamine
  - BP-4 Ostukorvi eest tasumine

- Toote teenus – vastutab kaupluses olevate toodete ja toodete hindade eest. Füüsiline andmemudel on kirjeldatud punktis 4.6.1.5.

  Äriprotsessid, mille eest teenus vastutab:

  - Toodete otsimine
  - BP-6 Kaupluse toote lisamine

Rakendused

- Administraatori rakendus – kasutab Spring boot admin rakendust koos spring-cloud-config mooduliga. Tegemist on veebirakendusega, mille läbi on võimalik järgida rakenduste tööd. Tänu spring-cloud-config moodulile hoiab administraatori rakendus ka kõigi teiste mikroteenuste konfiguratsiooni faile.

- Kliendi mobiilirakendus – on proxy rakendus avaliku võrgu ja sisevõrgu vahel. Kõik kliendi mobiilirakenduse päringud käivad läbi selle.

- Kassa rakendus – on proxy rakendus avaliku võrgu ja sisevõrgu vahel. Tegemist on veebirakendusega, milles on veebiliides kassade jaoks.

- Teenuste register – kasutab spring-cloud-eureka moodulit, mis pakub teenuste registreerimise funktsionaalsust.
- Logstash – elastic poolt loodud programm logide hoidmiseks.
- ElasticSearch – elastic poolt loodud programm logide otsimiseks.
- Kibana – elastic poolt loodud veebirakendus, mis pakub graafilist kasutajaliidest, mille kaudu saab otsida logisid kasutades ElasticSearch päringuid.
- Kafka - on publitseeri ja kuula sõnumivahetus server.

4.7 Alltöövõtja vaade

4.7.1 Andmete kirjeldus

Andmete kirjeldused on füüsiliste andmemudelite järel peatükis 4.6.1.

4.7.2 Programmi disain

Programmi kood on lisatud lisadesse 1-9.

5 Jõudlustestid

Jõudlusteste viiakse antud töös läbi 4 korda. Iga test tõstetakse kasutajate arvu kahe kordseks (500, 1000, 2000, 4000).

Jõudlustestides kasutatud riistara:

Tabel 51.Jõudlustestide riistvara

Parameeter	Masin 1	Masin 2	Masin 3	Masin 4
Protsessor	4 Ghz	2.2 Ghz	2.2 Ghz	2.5 Ghz
Tuumade arv	8	8	8	4
Mälu	32 GB	16 GB	16 GB	8 GB
Kõvaketas	SSD	SSD	SSD	SSD
Rakendused	• 6 postgres andmebaasi • Autoriseerimise teenus • Kliendi teenus • Toodete teenus • Kaupluse teenus • Tehingute teenus • Kafka	• Ostukorvi teenus • Autoriseerimise teenus	• Toodete register • Admini rakendus • Logstash • Elastic-Search • Kibana	• Kliendi mobiili rakendus

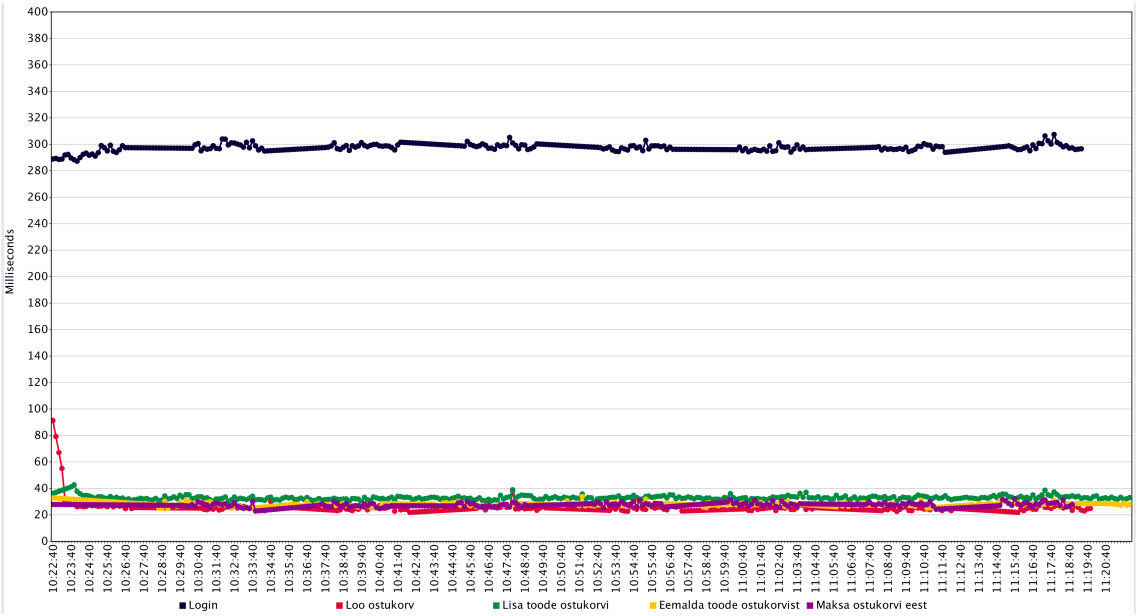
Jõudlustesti päringud

1. Login (1 kord)
2. Loo ostukorv (1 kord)
3. Lisa toode ostukorvi (9 korda)
4. Eemalda toode ostukorvist (3 korda)
5. Maksa ostukorvi eest (1 kord)

5.1 Jõudlustest 1

Testi pikkus – 60 minutit ning paralleelsete kasutajate arv testis – 500

Selles testis simuleeritakse süsteemi reaalseid kasutajaid. See tähendab, et päringute vahele on lisatud 30 sekundiline viide (aeg, kus klient poes ringi käib ja otsib järgmist toodet).



Joonis 30: Jõudlustest 1 - päringute kestuste graafik

Selle jõudlustesti vältel osteti 4184 ostukorvi, mis teeb keskmisel 70 ostukorvi minutis ning jääb peatükis 4.5.3 saadud koormuse vahemiku keskmisesse osasse. Kõik päringud selle testi vältel õnnestusid, vigu ei esinenud. Päringute kestused on stabiilsed, seda on näha nii päringute kestuse graafikust. Päringute kestuse väike standard hälve kinnitab, et päringud on stabiilsed ning ajas vähe muutuvad.

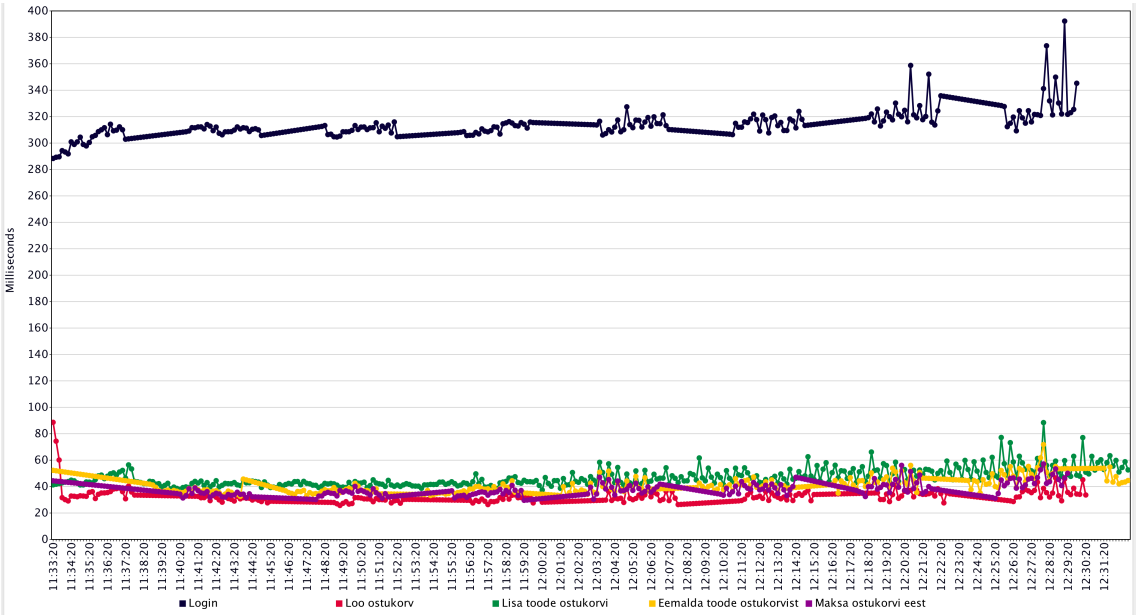
Tabel 52.Jõudlustesti 1 analüüs

Päring	Login	Loo ostukorv	Lisa toode ostukorvi	Eemalda toode ostukorvist	Ostukorvi tasumine
Minimaalne kestus (ms)	280	17	21	18	19
Maksimaalne kestus (ms)	411	250	121	100	120
Keskmine kestus (ms)	297	25	47	42	45
Keskmise kestuse standardhälve (ms)	8.41	7.26	6.34	6.32	6.33

5.2 Jõudlustest 2

Testi pikkus – 60 minutit ning paralleelsete kasutajate arv testis – 1000.

Selles testis simuleeritakse süsteemi reaalseid kasutajaid. See tähendab, et päringute vahele on lisatud 30 sekundiline viide (aeg, kus klient poes ringi käib ja otsib järgmist toodet).



Joonis 31: Jõudlustest 2 - päringute kestuste graafik

Selle jõudlustesti vältel osteti 8388 ostukorvi, mis teeb keskmisel 139 ostukorvi minutis ning jääb peatükis 4.5.3 saadud koormuse vahemiku teise kolmandikku. Kõik päringud selle testi vältel õnnestusid, vigu ei esinenud. Päringute kestused on stabiilsed, seda on näha nii päringute kestuse graafikust. Päringute kestuse väike standard hälve kinnitab, et päringud on stabiilsed ning ajas vähe muutuvad.

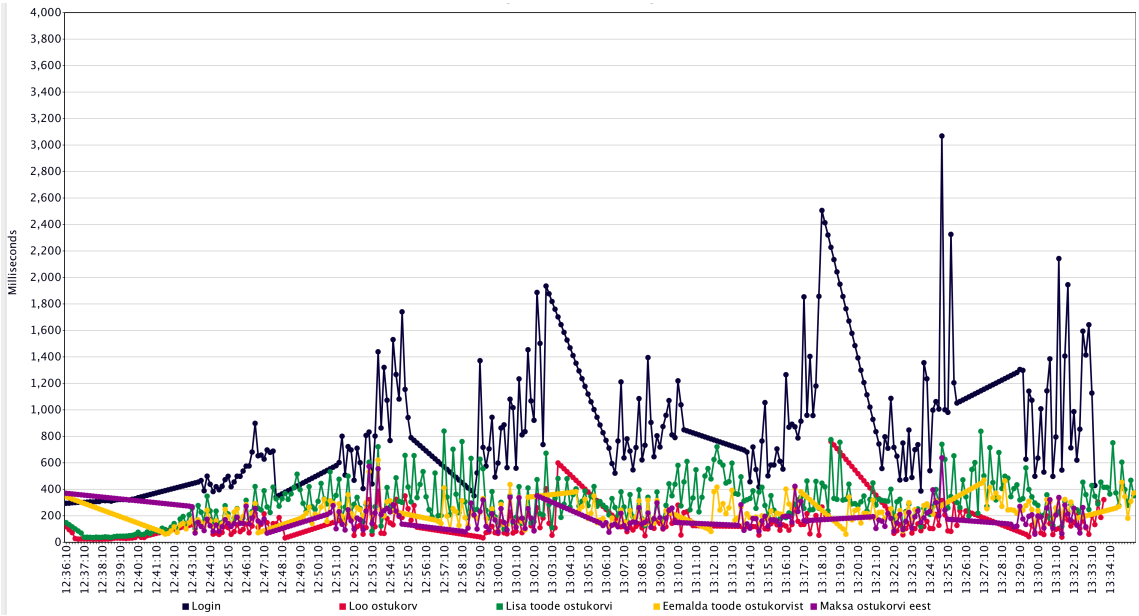
Tabel 53.Jõudlustesti 2 analüüs

Päring	Login	Loo ostukorv	Lisa toode ostukorvi	Eemalda toode ostukorvist	Ostukorvi tasumine
Minimaalne kestus (ms)	280	16	20	18	18
Maksimaalne kestus (ms)	804	161	134	146	167
Keskmine kestus (ms)	314	32	60	57	58
Keskmise kestuse standardhälve (ms)	26.12	13.25	15.63	14.98	15.04

5.3 Jõudlustest 3

Testi pikkus – 60 minutit ning paralleelsete kasutajate arv testis – 2000.

Selles testis simuleeritakse süsteemi reaalseid kasutajaid. See tähendab, et päringute vahele on lisatud 30 sekundiline viide (aeg, kus klient poes ringi käib ja otsib järgmist toodet).



Joonis 32: Jõudlustest 3 - päringute kestuste graafik

Selle jõudlustesti vältel osteti 16019 ostukorvi, mis teeb keskmisel 266 ostukorvi minutis ning ületab peatükis 4.5.3 saadud koormuse vahemiku. Kõik päringud selle testi vältel õnnestusid, vigu ei esinenud. Selles testis päringute kestused kasvasid märgatavalt ning olid väga kõikuvad, sest koormus oli suur. Ebastabiilsust kinnitas ka suur kestuse standardhälve.

Tabel 54.Jõudlustesti 3 analüüs

Päring	Login	Loo ostukorv	Lisa toode ostukorvi	Eemalda toode ostukorvist	Ostukorvi tasumine
Minimaalne kestus (ms)	280	17	21	19	20
Maksimaalne kestus (ms)	5333	1463	1732	1634	1980
Keskmine kestus (ms)	838	134	153	149	176
Keskmise kestuse standardhälve (ms)	643.25	153.41	174.00	169.59	184.51

5.4 Jõudlustest 4

Testi pikkus – 60 minutit ning paralleelsete kasutajate arv testis – 4000.

Käesolev test ebaõnnestus täielikult. Süsteem lakkas töötamast.

5.5 Jõudlustestide kokkuvõte

Jõudlustestidest tuli välja, et antud riistvaral suudab rakendus stabiilselt töötada 1000 kasutajaga, mis teeb umbes 139 ostukorvi minutis. Kahekordse koormuse kasvamise korral (266 ostukorvi minutis) on süsteem võimaline kliente teenindama, aga hea oleks selleljuhul rakendused laiendada. See katab ära turuanalüüsist tulnud mittefunktsionaalse nõude, mis sätestab, et süsteem peab toime tulema 1000 aktiivse kasutajaga ning 20-200 ostukorviga minutis. Suurema koormuse korral tuleks rakendused laiendada mitme serveri vahel, et jaotada koormust. Antud testidel oli autoriseerimise teenust kaks tükki ja ülejäänuid üks.

Uue funktsionaalsuse lisamine ei tekita probleemi mirkoteenuste laiendamises. Probleem võib tekkida teenuste registris, aga teenuste register ei saa suurt koormust. Mikroteenused pöörduvad teenuste registri pool perioodiliselt ja ei vaja seepärast laiendamist, kuna periood ei ole lühike.

6 Edasiarenduse võimalused

Loodud süsteem ei ole piisav, et pakkuda täisfunktsionaalsusega kassasüsteemi. Selleks, et pakkuda täisfunktsionaalset kassasüsteemi, tuleb süsteemi järgmiste iteratsioonidega täiendada. Järgnevalt on eristatud loetelu funktsionaalsustest, mida tuleks teha, et pakkuda täisfunktsionaalset kassasüsteemi.

- Automatiseerida mikroteenuste monitoorimist ning automaatset laiendatavust vastavalt koormuse kasvule.
- Realiseerida haldusrakendus kauplustele, mille kaudu kauplused saavad hallata töötajaid, tooteid, kampaaniaid ning hindasid.
- Realiseerida kassasüsteem kassatöötajatele, mille kaudu saab tooteid ostukorvi lisada ning nende eest tasuda sularahas, pangakaardiga või mobiilirakendusega.
- Realiseerida mobiilirakendus klientidele, mis kasutab kliendi mobiilirakenduse teenust ja mille läbi saavad kliendid ostukorvi luua, lisada ostukorvi tooteid ning tasuda.
- Realiseerida liidestused väliste süsteemidega, näiteks sendgrid (emaili saatmine), krediitkaardi maksete keskusega, kaardimakse terminaliga.
- Luua kauplade haldamiseks laosüsteem, mille kaudu parema ülevaate kaupluses olevatest kaupadest ning kogustest.
- Genereerida kauplustele raporteid.

7 Kokkuvõte

Antud töö eesmärgiks oli kirjeldada nutikassa süsteemi osad lähtudes Zachmani ettevõtte raamistikust ja tõestada äriprotsesside teostatavust mikroteenuste arhitektuuril ning valideerida neid jõudlustestidega. Skoobist jäid välja kasutajaliideste loomised ning väliste süsteemide integreerimised.

Töökäigus selgitati kuidas antud lahendus mikroteenustega saavutada. Selleks oli vaja uurida mikroteenuste eeliseid ja puuduseid. Seejärel pandi paika süsteemi põhiprotsessid ning nõuded süsteemile.

Töö tulemusena valmis poc rakendus, mis tõestab antud süsteemi teostatavust mikroteenustega, mis kasutab PostgreSQL andmebaase ja Java spring raamistikku, Kafkat ja Eureka süsteeme. Süsteem võimaldab kliente ja kaupluseid registreerida ning ostukorvi luua, tooteid lisada ning ostukorvi eest tasuda.

Loodud süsteem on esimese iteratsiooni tulemus, et tõestada mikroteenuste kasutatavust ning selle toimivust vastava koormuse all ning sellepärast realiseeriti ainult mikroteenused, mitte kasutajaliideste rakendused ega väliseid liidestusi.

Mikroteenused on kasutatavad ka laiemalt (pangad, uued rahvusvahelised firmad, startupid), mitte ainult nutikassa süsteemis.

Kasutatud kirjandus

[1] Zachman'i raamistik [WWW] https://www.tlu.ee/opmat/in/Arhitektuur/41_zachmani_raamistik.html (10.02.2017)

[2] Zachman Framework [WWW] https://en.wikipedia.org/wiki/Zachman_Framework (10.02.2017)

[3] Data Modeling – Conceptual, Logical And Physical Data Models [WWW] http://www.1keydata.com/datawarehousing/data-modeling-levels.html (11.02.2017)

[4] Jaekaubandusettevõtete müügitulu kasvas ka aasta viimasel kuul [WWW] https://www.stat.ee/pressiteade-2017-013 (29.04.2017)

[5] Retail trade volume index overview [WWW] http://ec.europa.eu/eurostat/statistics-explained/index.php/Retail_trade_volume_index_overview (29.04.2017)

[6] Detsembris jaemüük kasv kiirenes [WWW] http://www.stat.ee/pressiteade-2016-013?highlight=jaemüük (20.04.2017)

[7] Audit Trail – Tracing Data Changes in Database [WWW] https://www.codeproject.com/Articles/105768/Audit-Trail-Tracing-Data-Changes-in-Database (15.03.2017)

[8] What are microservices [WWW] http://microservices.io (15.03.2017)

[9] Monilothic Application [WWW] https://en.wikipedia.org/wiki/Monolithic_application (15.03.2017)

[10] Service oriented architecture [WWW] https://en.wikipedia.org/wiki/Service-oriented_architecture (15.02.2017)

[11] Microservices in a nutshell. Pros and Cons [WWW] https://blog.philipphauer.de/microservices-nutshell-pros-cons/ (15.02.2017)

[12] Irakli Nadareishvili, Ronnie Mitra, Matt McLarty & Mike Amundsen, "Microservice Architecture", O'Reilly Media, Inc, 2016

[13] Service Discovery in Microservices Architecture [WWW] https://www.nginx.com/blog/service-discovery-in-a-microservices-architecture/ (16.02.2017)

[14] Baker Street: Avoiding Bottlenecs with a Client-Slide Load Balancer for Microservices [WWW] https://thenewstack.io/baker-street-avoiding-bottlenecks-with-a-client-side-load-balancer-for-microservices/ (17.04.2017)

[15] Admin UI for administration of spring boot application [WWW] https://github.com/codecentric/spring-boot-admin (29.04.2017)

[16] Spring Framework [WWW] https://projects.spring.io/spring-framework/ (29.04.2017)

[17] Logical Data Model [WWW] http://www.1keydata.com/datawarehousing/logical-data-model.html (29.04.2017)

[18] Kafka documentation [WWW] http://kafka.apache.org/documentation.html (29.04.2017)

[19] Scalability [WWW] https://en.wikipedia.org/wiki/Scalability (01.05.2017)

Lisa 1 – Autoriseerimise teenuse programmikood

```
dependencies {
     compile project(":domain-common")
     compile project(":domain-common-database")
     compile project(":auth-service-api")
     compile project(":business-assert")
     compile "org.springframework.boot:spring-boot-starter-web"
     compile "org.springframework.boot:spring-boot-starter-actuator:$spring-
BootVersion"
     compile "org.springframework.cloud:spring-cloud-starter-config"
     compile "org.springframework.cloud:spring-cloud-starter-eureka"
     compile "org.springframework.cloud:spring-cloud-starter-hystrix"
     compile "org.postgresql:postgresql:$postgresqlVersion"
     compile "org.liquibase:liquibase-core:$liquibaseCoreVersion"
     compile "org.springframework:spring-jdbc"
     compile "com.zaxxer:HikariCP:2.6.1"
     compile "org.mybatis.spring.boot:mybatis-spring-boot-starter:1.3.0"
     compile "org.mybatis:mybatis-typehandlers-jsr310:1.0.2"
}
```

```
spring.application:
  name: auth-service
spring.cloud.config:
  uri: http://localhost:9999/config
  failFast: true
liquibase:
  changeLog: classpath:/changelog/master.xml
  enabled: true
mybatis:
  config-location: classpath:mybatis-config.xml
```

```
package ee.cashier.auth;
import org.springframework.boot.SpringApplication;
import org.springframework.cloud.client.SpringCloudApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.netflix.feign.EnableFeignClients;
@SpringCloudApplication
@EnableEurekaClient
@EnableFeignClients
public class AuthApplication {
    public static void main(String[] args) {
        SpringApplication.run(AuthApplication.class, args);
    }
}
```

```
package external.api;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import static org.springframework.http.MediaType.APPLICATION_JSON_UTF8_VALUE;
import static org.springframework.web.bind.annotation.RequestMethod.GET;
```

```java
import static org.springframework.web.bind.annotation.RequestMethod.POST;
public interface LoginApi {
    @RequestMapping(value = "/v1/login/{token}", method = GET,
            produces = APPLICATION_JSON_UTF8_VALUE)
    UsernameResponseJson findData(@PathVariable("token") String token);
    @RequestMapping(value = "/v1/credentials", method = POST,
            consumes = APPLICATION_JSON_UTF8_VALUE,
            produces = APPLICATION_JSON_UTF8_VALUE)
    CredentialsResponseJson createCredentials(@RequestBody CredentialsRe-
questJson json);
    @RequestMapping(value = "/v1/login", method = POST,
            consumes = APPLICATION_JSON_UTF8_VALUE,
            produces = APPLICATION_JSON_UTF8_VALUE)
    LoginResponseJson login(@RequestBody LoginRequestJson json);
}
```

```java
package external.api;
public class CredentialsRequestJson {
    private String username;
    private String password;
    private Long customerId;
    public static CredentialsRequestJson newLoginRequest(String username,
                                                         String password,
Long customerId) {
        CredentialsRequestJson json = new CredentialsRequestJson();
        json.setUsername(username);
        json.setPassword(password);
        json.setCustomerId(customerId);
        return json;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public Long getCustomerId() {
        return customerId;
    }
    public void setCustomerId(Long customerId) {
        this.customerId = customerId;
    }
}
```

```java
package external.api;
public class CredentialsResponseJson {
    private Long usernameId;
    private String username;
    public Long getUsernameId() {
        return usernameId;
    }
    public void setUsernameId(Long usernameId) {
        this.usernameId = usernameId;
    }
```

```
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
}
```

```
package external.api;
public class LoginRequestJson {
    private String username;
    private String password;
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

```
package external.api;
public class LoginResponseJson {
    private String token;
    public void setToken(String token) {
        this.token = token;
    }
    public String getToken() {
        return token;
    }
}
```

```
package external.api;
public class UsernameResponseJson {
    private String username;
    private Long customerId;
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public Long getCustomerId() {
        return customerId;
    }
    public void setCustomerId(Long customerId) {
        this.customerId = customerId;
    }
}
```

```
package ee.cashier.auth.api;
import ee.cashier.auth.service.FindUsernameService;
import ee.cashier.auth.service.LoginCreateService;
import ee.cashier.auth.service.LoginService;
```

```
import external.api.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;
@RestController
public class LoginApi implements external.api.LoginApi {
    @Autowired
    private LoginCreateService createLogin;
    @Autowired
    private LoginService login;
    @Autowired
    private FindUsernameService findUsername;
    @Override
    public UsernameResponseJson findData(@PathVariable String token) {
        return findUsername.findByToken(token).usernameResponse();
    }
    @Override
    public CredentialsResponseJson createCredentials(@RequestBody Credential-
sRequestJson json) {
        return createLogin.create(json).credentialsResponse();
    }
    @Override
    public LoginResponseJson login(@RequestBody LoginRequestJson json) {
        return login.login(json).loginResponse();
    }
}
```

```
package ee.cashier.auth.service;
import ee.cashier.auth.dao.PasswordDao;
import ee.cashier.auth.dao.TokenDao;
import ee.cashier.auth.model.Token;
import ee.cashier.auth.model.Username;
import external.api.LoginRequestJson;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;
import java.time.LocalDateTime;
import java.util.UUID;
import static ee.cashier.business.BusinessAssert.isTrue;
@Component
public class LoginService {
    @Autowired
    private PasswordDao passwordDao;
    @Autowired
    private TokenDao tokenDao;
    @Autowired
    private CryptService cryptService;
    @Autowired
    private FindUsernameService findUsername;
    @Transactional
    public Token login(LoginRequestJson json) {
        Username username = findUsername.find(json.getUsername());
        String hashedPassword = cryptService.generateHash(json.getPassword(),
username.getPasswordSalt());
        isTrue(passwordDao.exists(hashedPassword), "err.auth.credentialsNot-
Found");
        return saveToken(username);
    }
    private Token saveToken(Username username) {
        Token token = new Token();
        token.setUsernameId(username.getId());
```

```
        token.setValue(UUID.randomUUID().toString());
        token.setCreatedDTime(LocalDateTime.now());
        tokenDao.save(token);
        return token;
    }
}
```

```
package ee.cashier.auth.service;
import ee.cashier.auth.dao.PasswordDao;
import ee.cashier.auth.dao.UsernameDao;
import ee.cashier.auth.model.Username;
import external.api.CredentialsRequestJson;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;
import static ee.cashier.business.BusinessAssert.isFalse;
import static ee.cashier.business.BusinessAssert.isNotNull;
@Component
public class LoginCreateService {
    private static final Logger log = LoggerFactory.getLogger(LoginCreate-
Service.class);
    @Autowired
    private UsernameDao usernameDao;
    @Autowired
    private CryptService cryptService;
    @Autowired
    private PasswordDao passwordDao;
    @Transactional
    public Username create(CredentialsRequestJson json) {
        isNotNull(json.getCustomerId(), "err.auth.customerIdIsMissing");
        isFalse(usernameDao.existsByUsername(json.getUsername()), "er-
r.auth.usernameAlreadyExists");
        isNotNull(json.getPassword(), "err.auth.passwordIsMissing");
        Username username = new Username();
        username.setUsername(json.getUsername());
        username.setPasswordSalt(cryptService.generateSalt());
        username.setCustomerId(json.getCustomerId());
        usernameDao.save(username);
        passwordDao.save(cryptService.generateHash(json.getPassword(), user-
name.getPasswordSalt()));
        log.info("creating complete");
        return username;
    }
}
```

```
package ee.cashier.auth.service;
import ee.cashier.auth.dao.UsernameDao;
import ee.cashier.auth.model.Username;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import static ee.cashier.business.BusinessAssert.isNotNull;
@Component
public class FindUsernameService {
    @Autowired
    private UsernameDao usernameDao;
    @Autowired
    private FindTokenService findTokenService;
    public Username find(Long id) {
```

```
        return isNotNull(usernameDao.findOne(id), "err.auth.usernameNot-
Found");
    }
    public Username find(String username) {
        return isNotNull(usernameDao.findOneByUsername(username), "er-
r.auth.usernameNotFound");
    }
    public Username findByToken(String token) {
        return find(findTokenService.find(token).getUsernameId());
    }
}
```

```
package ee.cashier.auth.service;
import ee.cashier.auth.dao.TokenDao;
import ee.cashier.auth.model.Token;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import static ee.cashier.business.BusinessAssert.isNotNull;
@Component
public class FindTokenService {
    @Autowired
    private TokenDao tokenDao;
    public Token find(String token) {
        return isNotNull(tokenDao.findByToken(token), "err.auth.tokenNot-
Found");
    }
}
```

```
package ee.cashier.auth.service;
import org.springframework.security.crypto.bcrypt.BCrypt;
import org.springframework.stereotype.Service;
@Service
public class CryptService {
    private static final int LOG_ROUNDS = 12;
    private static final String HASH_PREFIX = "$2a$";
    private static final String SEPARATOR = "\\$";
    private static final int SALT_LENGTH = 28;
    public String generateSalt() {
        String generatedSalt = BCrypt.gensalt(LOG_ROUNDS);
        String[] parts = generatedSalt.split(SEPARATOR);
        return parts[parts.length - 1];
    }
    public String generateHash(String value, String plainSalt) {
        String saltPrefix = HASH_PREFIX + LOG_ROUNDS + "$";
        String salt = saltPrefix + plainSalt;
        String generatedHash = BCrypt.hashpw(value, salt);
        return generatedHash.substring(saltPrefix.length() + SALT_LENGTH);
    }
}
```

```
package ee.cashier.auth.model;
import external.api.CredentialsResponseJson;
import external.api.UsernameResponseJson;
public class Username {
    private Long id;
    private Long customerId;
    private String username;
    private String passwordSalt;
    public Long getId() {
```

```
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public Long getCustomerId() {
        return customerId;
    }
    public void setCustomerId(Long customerId) {
        this.customerId = customerId;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPasswordSalt() {
        return passwordSalt;
    }
    public void setPasswordSalt(String passwordSalt) {
        this.passwordSalt = passwordSalt;
    }
    public UsernameResponseJson usernameResponse() {
        UsernameResponseJson json = new UsernameResponseJson();
        json.setUsername(username);
        json.setCustomerId(customerId);
        return json;
    }
    public CredentialsResponseJson credentialsResponse() {
        CredentialsResponseJson json = new CredentialsResponseJson();
        json.setUsernameId(id);
        json.setUsername(username);
        return json;
    }
}
```

```
package ee.cashier.auth.model;
import external.api.LoginResponseJson;
import java.time.LocalDateTime;
public class Token {
    private Long id;
    private Long usernameId;
    private String value;
    private LocalDateTime createdDTime;
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public Long getUsernameId() {
        return usernameId;
    }
    public void setUsernameId(Long usernameId) {
        this.usernameId = usernameId;
    }
    public String getValue() {
        return value;
```

```
    }
    public void setValue(String value) {
        this.value = value;
    }
    public LocalDateTime getCreatedDTime() {
        return createdDTime;
    }
    public void setCreatedDTime(LocalDateTime createdDTime) {
        this.createdDTime = createdDTime;
    }
    public LoginResponseJson loginResponse() {
        LoginResponseJson json = new LoginResponseJson();
        json.setToken(value);
        return json;
    }
}
```

```
package ee.cashier.auth.dao;
import ee.cashier.auth.model.Username;
import org.apache.ibatis.annotations.Mapper;
@Mapper
public interface UsernameDao {
    void save(Username username);
    Username findOne(Long id);
    Username findOneByUsername(String username);
    boolean existsByUsername(String username);
}
```

```
package ee.cashier.auth.dao;
import ee.cashier.auth.model.Token;
import org.apache.ibatis.annotations.Mapper;
@Mapper
public interface TokenDao {
    void save(Token token);
    Token findByToken(String token);
}
```

```
package ee.cashier.auth.dao;
import org.apache.ibatis.annotations.Mapper;
@Mapper
public interface PasswordDao {
    void save(String passwordHash);
    boolean exists(String passwordHash);
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://my-
batis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="ee.cashier.auth.dao.UsernameDao">
    <resultMap id="UsernameResultMap" type="Username">
        <result column="id" property="id"/>
        <result column="username" property="username"/>
        <result column="password_salt" property="passwordSalt"/>
        <result column="customer_id" property="customerId"/>
    </resultMap>
    <insert id="save" useGeneratedKeys="true" keyColumn="id" keyProper-
ty="id">
        INSERT INTO auth.username (
```

```
            username,
            customer_id,
            password_salt
        ) VALUES (
            #{username},
            #{customerId},
            #{passwordSalt}
        )
    </insert>
    <select id="findOne" resultMap="UsernameResultMap">
        SELECT *
        FROM auth.username
        WHERE id = #{id}
    </select>
    <select id="findOneByUsername" resultMap="UsernameResultMap">
        SELECT *
        FROM auth.username
        WHERE username = #{username}
    </select>
    <select id="existsByUsername" resultType="boolean">
        SELECT count(*) > 0
        FROM auth.username
        WHERE username = #{username}
        LIMIT 1
    </select>
</mapper>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://my-
batis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="ee.cashier.auth.dao.TokenDao">
    <resultMap id="TokenResultMap" type="Token">
        <result column="id" property="id"/>
        <result column="username_id" property="usernameId"/>
        <result column="value" property="value"/>
        <result column="created_dtime" property="createdDTime"/>
    </resultMap>
    <insert id="save" useGeneratedKeys="true" keyColumn="id" keyProper-
ty="id">
        INSERT INTO auth.authentication_token (
            username_id,
            value,
            created_dtime
        ) VALUES (
            #{usernameId},
            #{value},
            #{createdDTime}
        )
    </insert>
    <select id="findByToken" resultMap="TokenResultMap">
       SELECT *
         FROM auth.authentication_token
        WHERE value = #{token}
    </select>
</mapper>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://my-
batis.org/dtd/mybatis-3-mapper.dtd">
```

```xml
<mapper namespace="ee.cashier.auth.dao.PasswordDao">
    <insert id="save">
        INSERT INTO auth.password (
            value
        ) VALUES (
            #{value}
        )
    </insert>
    <select id="exists" resultType="boolean">
        SELECT count(*) > 0
          FROM auth.password
         WHERE value = #{value}
    </select>
</mapper>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <settings>
        <setting name="cacheEnabled" value="false"/>
        <setting name="localCacheScope" value="STATEMENT"/>
    </settings>
    <typeAliases>
        <package name="ee.cashier.auth.model"/>
        <package name="ee.cashier.domain"/>
    </typeAliases>
    <typeHandlers>
    </typeHandlers>
    <mappers>
        <mapper resource="dao/PasswordDao.xml"/>
        <mapper resource="dao/TokenDao.xml"/>
        <mapper resource="dao/UsernameDao.xml"/>
    </mappers>
</configuration>
```

Lisa 2 – Kliendi teenuse programmikood

```
dependencies {
     compile project(":domain-common")
     compile project(":domain-common-database")
     compile project(":auth-service-api")
     compile project(":customer-service-api")
     compile project(":business-assert")
     compile "org.springframework.boot:spring-boot-starter-web"
     compile "org.springframework.boot:spring-boot-starter-actuator:$spring-
BootVersion"
     compile "org.springframework.cloud:spring-cloud-starter-config"
     compile "org.springframework.cloud:spring-cloud-starter-eureka"
     compile "org.springframework.cloud:spring-cloud-starter-feign"
     compile "org.springframework.cloud:spring-cloud-starter-hystrix"
     compile "org.springframework.cloud:spring-cloud-starter-stream-kafka"
     compile "com.fasterxml.jackson.datatype:jackson-datatype-jsr310:2.8.3"
     compile "org.postgresql:postgresql:$postgresqlVersion"
     compile "org.liquibase:liquibase-core:$liquibaseCoreVersion"
     compile "org.springframework:spring-jdbc"
     compile "com.zaxxer:HikariCP:2.6.1"
     compile "org.mybatis.spring.boot:mybatis-spring-boot-starter:1.3.0"
     compile "org.mybatis:mybatis-typehandlers-jsr310:1.0.2"
}
```

```
spring.application:
  name: customer-service
spring.cloud.config:
  uri: http://localhost:9999/config
  failFast: true
liquibase:
  changeLog: classpath:/changelog/master.xml
  enabled: true
mybatis:
  config-location: classpath:mybatis-config.xml
```

```
package ee.cashier.customer;
import org.springframework.boot.SpringApplication;
import org.springframework.cloud.client.SpringCloudApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.netflix.feign.EnableFeignClients;
import org.springframework.kafka.annotation.EnableKafka;
@EnableKafka
@EnableEurekaClient
@EnableFeignClients
@SpringCloudApplication
public class CustomerApplication {
     public static void main(String[] args) {
          SpringApplication.run(CustomerApplication.class, args);
     }
}
```

```java
package external.api;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import static org.springframework.http.MediaType.APPLICATION_JSON_UTF8_VALUE;
import static org.springframework.web.bind.annotation.RequestMethod.GET;
import static org.springframework.web.bind.annotation.RequestMethod.POST;
public interface CustomerApi {
    @RequestMapping(value = "/v1/customer/registration", method = POST,
            consumes = APPLICATION_JSON_UTF8_VALUE,
            produces = APPLICATION_JSON_UTF8_VALUE)
    CustomerResponseJson register(@RequestBody CustomerRegistrationRequestJ-
son json);
    @RequestMapping(value = "/v1/customer/{customerId}", method = GET,
            produces = APPLICATION_JSON_UTF8_VALUE)
    CustomerResponseJson findCustomerInfo(@PathVariable("customerId") Long
customerId);
}
```

```java
package external.api;
public class CustomerRegistrationRequestJson {
    private String name;
    private String email;
    private String password;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

```java
package external.api;
import java.time.LocalDate;
public class CustomerResponseJson {
    private Long id;
    private String name;
    private String email;
    private LocalDate registrationDate;
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getName() {
```

```
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public LocalDate getRegistrationDate() {
        return registrationDate;
    }
    public void setRegistrationDate(LocalDate registrationDate) {
        this.registrationDate = registrationDate;
    }
}
```

```
package ee.cashier.customer.api;
import ee.cashier.customer.service.CreateCustomerService;
import ee.cashier.customer.service.FindCustomerService;
import external.api.CustomerRegistrationRequestJson;
import external.api.CustomerResponseJson;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import static org.springframework.http.MediaType.APPLICATION_JSON_UTF8_VALUE;
import static org.springframework.web.bind.annotation.RequestMethod.GET;
import static org.springframework.web.bind.annotation.RequestMethod.POST;
@RestController
public class CustomerApi implements external.api.CustomerApi {
    @Autowired
    private CreateCustomerService registerCustomer;
    @Autowired
    private FindCustomerService findCustomer;
    @RequestMapping(value = "/v1/customer/registration", method = POST,
            consumes = APPLICATION_JSON_UTF8_VALUE,
            produces = APPLICATION_JSON_UTF8_VALUE)
    public CustomerResponseJson register(@RequestBody CustomerRegistrationRe-
questJson json) {
        return registerCustomer.register(json).toResponse();
    }
    @RequestMapping(value = "/v1/customer/{customerId}", method = GET,
            produces = APPLICATION_JSON_UTF8_VALUE)
    public CustomerResponseJson findCustomerInfo(@PathVariable("customerId")
Long customerId) {
        return findCustomer.find(customerId).toResponse();
    }
}
```

```
package ee.cashier.customer.service;
import ee.cashier.customer.dao.CustomerDao;
import ee.cashier.customer.external.auth.LoginApiClient;
import ee.cashier.customer.model.Customer;
import ee.cashier.customer.publish.CustomerRegisteredPublisher;
import external.api.CustomerRegistrationRequestJson;
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;
import java.time.LocalDate;
import static ee.cashier.business.BusinessAssert.isFalse;
import static ee.cashier.business.BusinessAssert.isNotEmpty;
import static external.api.CredentialsRequestJson.newLoginRequest;
@Component
public class CreateCustomerService {
    @Autowired
    private FindCustomerService findCustomer;
    @Autowired
    private CustomerDao customerDao;
    @Autowired
    private LoginApiClient loginApi;
    @Autowired
    private CustomerRegisteredPublisher customerRegisteredPublisher;
    @Transactional
    public Customer register(CustomerRegistrationRequestJson json) {
        isNotEmpty(json.getEmail(), "err.customer.emailIsMissing");
        isNotEmpty(json.getPassword(), "err.customer.passwordIsMissing");
        isFalse(findCustomer.existsByEmail(json.getEmail()), "err.cus-
tomer.emailAlreadyExists");
        Customer customer = new Customer();
        customer.setName(json.getName());
        customer.setEmail(json.getEmail());
        customer.setRegistrationDate(LocalDate.now());
        customerDao.save(customer);
        loginApi.createCredentials(
                newLoginRequest(json.getEmail(), json.getPassword(), cus-
tomer.getId()));
        customerRegisteredPublisher.publish(customer.getId());
        return customer;
    }
}
```

```
package ee.cashier.customer.service;
import ee.cashier.customer.dao.CustomerDao;
import ee.cashier.customer.model.Customer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import static ee.cashier.business.BusinessAssert.isNotNull;
@Component
public class FindCustomerService {
    @Autowired
    private CustomerDao customerDao;
    public Customer findByEmail(String email) {
        return isNotNull(customerDao.findByEmail(email), "err.customer.cus-
tomerNotFound");
    }
    public boolean existsByEmail(String email) {
        return customerDao.existsByEmail(email);
    }
    public Customer find(Long customerId) {
        return isNotNull(customerDao.findById(customerId), "err.customer.-
customerNotFound");
    }
}
```

```
package ee.cashier.customer.publish;
import java.time.LocalDate;
```

```
public class CustomerRegisteredEvent {
     private Long customerId;
     private String customerName;
     private String customerEmail;
     private LocalDate customerRegistrationDate;
     public Long getCustomerId() {
          return customerId;
     }
     public void setCustomerId(Long customerId) {
          this.customerId = customerId;
     }
     public String getCustomerName() {
          return customerName;
     }
     public void setCustomerName(String customerName) {
          this.customerName = customerName;
     }
     public String getCustomerEmail() {
          return customerEmail;
     }
     public void setCustomerEmail(String customerEmail) {
          this.customerEmail = customerEmail;
     }
     public LocalDate getCustomerRegistrationDate() {
          return customerRegistrationDate;
     }
     public void setCustomerRegistrationDate(LocalDate customerRegistra-
tionDate) {
          this.customerRegistrationDate = customerRegistrationDate;
     }
}
```

```
package ee.cashier.customer.publish;
import ee.cashier.customer.model.Customer;
import ee.cashier.customer.service.FindCustomerService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
@Component
public class CustomerRegisteredPublisher {
     private final String eventName = "customer-registered";
     @Autowired
     private EventPublisher eventPublisher;
     @Autowired
     private FindCustomerService findCustomerService;
     public void publish(Long customerId) {
          Customer customer = findCustomerService.find(customerId);
          CustomerRegisteredEvent event = new CustomerRegisteredEvent();
          event.setCustomerId(customerId);
          event.setCustomerName(customer.getName());
          event.setCustomerEmail(customer.getEmail());
          event.setCustomerRegistrationDate(customer.getRegistrationDate());
          eventPublisher.publish(eventName, event);
     }
}
```

```
package ee.cashier.customer.publish;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.beans.factory.annotation.Autowired;
```

```java
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Component;
@Component
public class EventPublisher {
    @Autowired
    private KafkaTemplate<Object, Object> kafkaTemplate;
    @Autowired
    private ObjectMapper objectMapper;
    protected void publish(String topic, Object event) {
        kafkaTemplate.send(topic, convertToJson(event)).addCallback(
                o -> System.out.println("Success " + o),
                throwable -> {
                    System.err.println(throwable.getMessage());
                    throwable.printStackTrace();
                });
    }
    private String convertToJson(Object event) {
        try {
            return objectMapper.writeValueAsString(event);
        } catch (JsonProcessingException e) {
            throw new RuntimeException(e);
        }
    }
}
```

```java
package ee.cashier.customer.model;
import external.api.CustomerResponseJson;
import java.time.LocalDate;
public class Customer {
    private Long id;
    private String name;
    private String email;
    private LocalDate registrationDate;
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public LocalDate getRegistrationDate() {
        return registrationDate;
    }
    public void setRegistrationDate(LocalDate registrationDate) {
        this.registrationDate = registrationDate;
    }
    public CustomerResponseJson toResponse() {
        CustomerResponseJson json = new CustomerResponseJson();
```

```
        json.setId(id);
        json.setName(name);
        json.setEmail(email);
        json.setRegistrationDate(registrationDate);
        return json;
    }
}
```

```
package ee.cashier.customer.external.auth;
import external.api.LoginApi;
import org.springframework.cloud.netflix.feign.FeignClient;
@FeignClient("auth-service")
public interface LoginApiClient extends LoginApi {}
```

```
package ee.cashier.customer.dao;
import ee.cashier.customer.model.Customer;
import org.apache.ibatis.annotations.Mapper;
@Mapper
public interface CustomerDao {
    Customer findById(Long customerId);
    Customer findByEmail(String email);
    boolean existsByEmail(String email);
    void save(Customer customer);}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://my-
batis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="ee.cashier.customer.dao.CustomerDao">
    <resultMap id="CustomerResultMap" type="Customer">
        <result column="id" property="id"/>
        <result column="name" property="name"/>
        <result column="email" property="email"/>
        <result column="registration_date" property="registrationDate"/>
    </resultMap>
    <insert id="save" useGeneratedKeys="true" keyColumn="id" keyProper-
ty="id">
        INSERT INTO customer.customer (
            name,
            email,
            registration_date
        ) VALUES (
            #{name},
            #{email},
            #{registrationDate}
        )
    </insert>
    <select id="findByEmail" resultMap="CustomerResultMap">
        SELECT *
          FROM customer.customer
         WHERE email = #{email}
    </select>
    <select id="findById" resultMap="CustomerResultMap">
        SELECT *
          FROM customer.customer
         WHERE id = #{customerId}
    </select>
    <select id="existsByEmail" resultType="boolean">
        SELECT count(*) > 0
          FROM customer.customer
         WHERE email = #{email}
         LIMIT 1
```

```
    </select>
</mapper>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <settings>
        <setting name="cacheEnabled" value="false"/>
        <setting name="localCacheScope" value="STATEMENT"/>
    </settings>
    <typeAliases>
        <package name="ee.cashier.customer.model"/>
        <package name="ee.cashier.domain"/>
    </typeAliases>
    <mappers>
        <mapper resource="dao/CustomerDao.xml"/>
    </mappers>
</configuration>
```

Lisa 3 – Kliendi mobiilirakenduse teenuse programmikood

```
dependencies {
     compile project(":domain-common")
     compile project(":business-assert")
     compile project(":auth-service-api")
     compile project(":customer-service-api")
     compile project(":product-service-api")
     compile project(":purchase-service-api")
     compile project(":store-service-api")
     compile "org.springframework.boot:spring-boot-starter-web"
     compile "org.springframework.boot:spring-boot-starter-actuator:$spring-
BootVersion"
     compile "org.springframework.cloud:spring-cloud-starter-config"
     compile "org.springframework.cloud:spring-cloud-starter-eureka"
     compile "org.springframework.cloud:spring-cloud-starter-feign"
     compile "org.springframework.cloud:spring-cloud-starter-hystrix"
     compile "com.fasterxml.jackson.datatype:jackson-datatype-jsr310:2.8.3"
}
```

```
spring.application:
  name: mobile-service
spring.cloud.config:
  uri: http://localhost:9999/config
  failFast: true
```

```
package ee.cashier.mobile;
import ee.cashier.mobile.auth.SecurityFilter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.web.servlet.FilterRegistrationBean;
import org.springframework.cloud.client.SpringCloudApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.netflix.feign.EnableFeignClients;
import org.springframework.context.annotation.Bean;
@EnableEurekaClient
@EnableFeignClients
@SpringCloudApplication
public class MobileApplication {
     @Autowired
     private SecurityFilter securityFilter;
     public static void main(String[] args) {
          SpringApplication.run(MobileApplication.class, args);
     }
     @Bean
     public FilterRegistrationBean requestLogFilterRegistration() {
          FilterRegistrationBean bean = new FilterRegistrationBean();
          bean.setFilter(securityFilter);
          bean.addUrlPatterns("/v1/*");
          bean.setOrder(0);
          return bean;
     }
```

```
}
```

```
package ee.cashier.mobile.api;
public class BarcodeJson {
     private String barcode;
     public String getBarcode() {
          return barcode;
     }
     public void setBarcode(String barcode) {
          this.barcode = barcode;
     }
}
```

```
package ee.cashier.mobile.api;
import ee.cashier.mobile.external.CustomerApi;
import ee.cashier.mobile.external.LoginApi;
import ee.cashier.mobile.external.ProductApi;
import ee.cashier.mobile.external.PurchaseApi;
import ee.cashier.mobile.external.StoreApi;
import external.api.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import java.util.List;
import static ee.cashier.business.BusinessAssert.isNotNull;
import static ee.cashier.mobile.auth.AuthenticationContextHolder.get;
import static org.springframework.http.MediaType.APPLICATION_JSON_UTF8_VALUE;
import static org.springframework.web.bind.annotation.RequestMethod.GET;
import static org.springframework.web.bind.annotation.RequestMethod.POST;
@RestController
public class MobileController {
     @Autowired
     private LoginApi loginApi;
     @Autowired
     private CustomerApi customerApi;
     @Autowired
     private ProductApi productApi;
     @Autowired
     private PurchaseApi purchaseApi;
     @Autowired
     private StoreApi storeApi;
     @RequestMapping(value = "/login", method = POST, consumes =
APPLICATION_JSON_UTF8_VALUE,
               produces = APPLICATION_JSON_UTF8_VALUE)
     public LoginResponseJson login(@RequestBody LoginRequestJson json) {
          return loginApi.login(json);
     }
     @RequestMapping(value = "/v1/profile", method = GET, produces = APPLICA-
TION_JSON_UTF8_VALUE)
     public CustomerResponseJson findCustomerInfo() {
          return customerApi.findCustomerInfo(get().getCustomerId());
     }
     @RequestMapping(value = "/v1/purchases/create", method = POST, produces =
APPLICATION_JSON_UTF8_VALUE,
               consumes = APPLICATION_JSON_UTF8_VALUE)
     public PurchaseResponseJson create(@RequestBody BarcodeJson json) {
          StoreResponseJson store =
isNotNull(storeApi.search(json.getBarcode()), "err.mobile.storeNotFound");
```

```
        return purchaseApi.create(store.getStoreId());
    }
    @RequestMapping(value = "/v1/purchases", method = GET, produces = APPLI-
CATION_JSON_UTF8_VALUE)
    public List<PurchaseResponseJson> find() {
        return purchaseApi.findByCustomer(get().getCustomerId());
    }
    @RequestMapping(value = "/v1/purchases/{purchaseId}", method = GET, pro-
duces = APPLICATION_JSON_UTF8_VALUE)
    public PurchaseResponseJson find(@PathVariable("purchaseId") Long purcha-
seId) {
        return purchaseApi.find(purchaseId);
    }
    @RequestMapping(value = "/v1/purchases/{purchaseId}/products/add", method
= POST,
            produces = APPLICATION_JSON_UTF8_VALUE)
    public PurchaseResponseJson addProduct(@PathVariable("purchaseId") Long
purchaseId,
                                           @RequestBody BarcodeJson
json) {
        PurchaseResponseJson purchase = find(purchaseId);
        ProductResponseJson product =
productApi.search(purchase.getStoreId(), json.getBarcode());
        return purchaseApi.addProduct(purchaseId, product.getProductId());
    }
    @RequestMapping(value = "/v1/purchases/{purchaseId}/products/
{productId}/remove", method = POST,
            produces = APPLICATION_JSON_UTF8_VALUE)
    public PurchaseResponseJson removeProduct(@PathVariable("purchaseId")
Long purchaseId,
                                              @PathVariable("productId")
Long productId) {
        return purchaseApi.removeProduct(purchaseId, productId);
    }
    @RequestMapping(value = "/v1/purchases/{purchaseId}/pay", method = POST,
            produces = APPLICATION_JSON_UTF8_VALUE)
    public PurchaseResponseJson pay(@PathVariable("purchaseId") Long purcha-
seId) {
        return purchaseApi.pay(purchaseId);
    }
}
```

```
package ee.cashier.mobile.auth;
import external.api.UsernameResponseJson;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class AuthenticationContextHolder {
    private static final Logger log = LoggerFactory.getLogger(Authentication-
ContextHolder.class);
    private static final ThreadLocal<UsernameResponseJson> authentications =
new ThreadLocal<>();
    public static void put(UsernameResponseJson shardContext) {
        log.debug("Setting context");
        authentications.set(shardContext);
    }
    public static UsernameResponseJson get() {
        log.debug("Getting");
        return authentications.get();
    }
    public static void clear() {
```

```
        log.debug("Clearing");
        authentications.remove();
    }
}
```

```
package ee.cashier.mobile.auth;
import ee.cashier.mobile.external.LoginApi;
import external.api.UsernameResponseJson;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import javax.servlet.*;
import javax.servlet.http.HttpServletRequest;
import java.io.IOException;
@Component
public class SecurityFilter implements Filter {
    @Autowired
    private LoginApi loginApi;
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
    }
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain)
            throws IOException, ServletException {
        String authenticationToken = ((HttpServletRequest)
servletRequest).getHeader("x-auth");
        if (authenticationToken == null) {
            throw new RuntimeException("Authentication token is missing");
        }
        UsernameResponseJson tokenData = loginApi.findData(authenticationTo-
ken);
        if (tokenData == null) {
            throw new RuntimeException("Authentication token is not valid");
        }
        AuthenticationContextHolder.put(tokenData);
        filterChain.doFilter(servletRequest, servletResponse);
    }
    @Override
    public void destroy() {
    }
}
```

```
package ee.cashier.mobile.external;
import org.springframework.cloud.netflix.feign.FeignClient;
@FeignClient("customer-service")
public interface CustomerApi extends external.api.CustomerApi {
}
```

```
package ee.cashier.mobile.external;
import org.springframework.cloud.netflix.feign.FeignClient;
@FeignClient("auth-service")
public interface LoginApi extends external.api.LoginApi {
}
```

```
package ee.cashier.mobile.external;
import org.springframework.cloud.netflix.feign.FeignClient;
@FeignClient("product-service")
public interface ProductApi extends external.api.ProductApi {
```

```
}
```

```
package ee.cashier.mobile.external;
import org.springframework.cloud.netflix.feign.FeignClient;
@FeignClient("purchase-service")
public interface PurchaseApi extends external.api.PurchaseApi {
}
```

```
package ee.cashier.mobile.external;
import org.springframework.cloud.netflix.feign.FeignClient;
@FeignClient("store-service")
public interface StoreApi extends external.api.StoreApi {
}
```

Lisa 4 – Tehingu teenuse programmikood

```
dependencies {
     compile project(":domain-common")
     compile project(":domain-common-database")
     compile project(":business-assert")
     compile project(":auth-service-api")
     compile "org.springframework.boot:spring-boot-starter-web"
     compile "org.springframework.boot:spring-boot-starter-actuator:$spring-
BootVersion"
     compile "org.springframework.cloud:spring-cloud-starter-config"
     compile "org.springframework.cloud:spring-cloud-starter-eureka"
     compile "org.springframework.cloud:spring-cloud-starter-feign"
     compile "org.springframework.cloud:spring-cloud-starter-hystrix"
     compile "org.springframework.cloud:spring-cloud-starter-stream-kafka"
     compile "com.fasterxml.jackson.datatype:jackson-datatype-jsr310:2.8.3"
     compile "org.postgresql:postgresql:$postgresqlVersion"
     compile "org.liquibase:liquibase-core:$liquibaseCoreVersion"
     compile "org.springframework:spring-jdbc"
     compile "com.zaxxer:HikariCP:2.6.1"
     compile "org.mybatis.spring.boot:mybatis-spring-boot-starter:1.3.0"
     compile "org.mybatis:mybatis-typehandlers-jsr310:1.0.2"
}
```

```
spring.application:
  name: payment-service
spring.cloud.config:
  uri: http://localhost:9999/config
  failFast: true
liquibase:
  changeLog: classpath:/changelog/master.xml
  enabled: true
mybatis:
  config-location: classpath:mybatis-config.xml
```

```
package ee.cashier.payment;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.cloud.client.SpringCloudApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.netflix.feign.EnableFeignClients;
import org.springframework.context.annotation.Bean;
import org.springframework.kafka.annotation.EnableKafka;
@EnableKafka
@EnableEurekaClient
@EnableFeignClients
@SpringCloudApplication
public class PaymentApplication {
     public static void main(String[] args) {
          SpringApplication.run(PaymentApplication.class, args);
     }
}
```

```java
package ee.cashier.payment.subscribe;
import java.math.BigDecimal;
public class PurchaseConfirmedEvent {
    private Long purchaseId;
    private BigDecimal priceAmount;
    private String currencyCode;
    public Long getPurchaseId() {
        return purchaseId;
    }
    public void setPurchaseId(Long purchaseId) {
        this.purchaseId = purchaseId;
    }
    public BigDecimal getPriceAmount() {
        return priceAmount;
    }
    public void setPriceAmount(BigDecimal priceAmount) {
        this.priceAmount = priceAmount;
    }
    public String getCurrencyCode() {
        return currencyCode;
    }
    public void setCurrencyCode(String currencyCode) {
        this.currencyCode = currencyCode;
    }
}
```

```java
package ee.cashier.payment.subscribe;
import com.fasterxml.jackson.databind.ObjectMapper;
import ee.cashier.payment.publish.PurchasePaidEventPublisher;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Component;
import java.io.IOException;
@Component
public class PurchaseConfirmedSubscriber {
    @Autowired
    private ObjectMapper objectMapper;
    @Autowired
    private PurchasePaidEventPublisher purchasePaidPublisher;
    @KafkaListener(topics = "purchase-confirmed")
    public void consume(String message) throws IOException {
        PurchaseConfirmedEvent messageObject = objectMapper.readValue(mes-
sage, PurchaseConfirmedEvent.class);
        purchasePaidPublisher.publish(messageObject);
    }
}
```

```java
package ee.cashier.payment.publish;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Component;
@Component
public class EventPublisher {
    @Autowired
    private KafkaTemplate<Object, Object> kafkaTemplate;
    @Autowired
    private ObjectMapper objectMapper;
    protected void publish(String topic, Object event) {
```

```
        kafkaTemplate.send(topic, convertToJson(event)).addCallback(
                o -> System.out.println("Success " + o),
                throwable -> {
                    System.err.println(throwable.getMessage());
                    throwable.printStackTrace();
                });
    }
    private String convertToJson(Object event) {
        try {
            return objectMapper.writeValueAsString(event);
        } catch (JsonProcessingException e) {
            throw new RuntimeException(e);
        }
    }
}
```

```
package ee.cashier.payment.publish;
import ee.cashier.domain.Source;
public class PurchasePaidEvent {
    private Source source;
    public Source getSource() {
        return source;
    }
    public void setSource(Source source) {
        this.source = source;
    }
}
```

```
package ee.cashier.payment.publish;
import ee.cashier.domain.Source;
import ee.cashier.payment.subscribe.PurchaseConfirmedEvent;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
@Component
public class PurchasePaidEventPublisher {
    private final String eventName = "purchase-paid";
    @Autowired
    private EventPublisher eventPublisher;
    public void publish(PurchaseConfirmedEvent message) {
        PurchasePaidEvent event = new PurchasePaidEvent();
        event.setSource(Source.source("PURCHASE", message.getPurchaseId()));
        eventPublisher.publish(eventName, event);
    }
}
```

Lisa 5 – Toote teenuse programmikood

```
dependencies {
    compile project(":domain-common")
    compile project(":domain-common-database")
    compile project(":business-assert")
    compile project(":auth-service-api")
    compile project(":product-service-api")
    compile "org.springframework.boot:spring-boot-starter-web"
    compile "org.springframework.boot:spring-boot-starter-actuator:$spring-
BootVersion"
    compile "org.springframework.cloud:spring-cloud-starter-config"
    compile "org.springframework.cloud:spring-cloud-starter-eureka"
    compile "org.springframework.cloud:spring-cloud-starter-feign"
    compile "org.springframework.cloud:spring-cloud-starter-hystrix"
    compile "org.springframework.cloud:spring-cloud-starter-stream-kafka"
    compile "com.fasterxml.jackson.datatype:jackson-datatype-jsr310:2.8.3"
    compile "org.postgresql:postgresql:$postgresqlVersion"
    compile "org.liquibase:liquibase-core:$liquibaseCoreVersion"
    compile "org.springframework:spring-jdbc"
    compile "com.zaxxer:HikariCP:2.6.1"
    compile "org.mybatis.spring.boot:mybatis-spring-boot-starter:1.3.0"
    compile "org.mybatis:mybatis-typehandlers-jsr310:1.0.2"
}
```

```
spring.application:
  name: product-service
spring.cloud.config:
  uri: http://localhost:9999/config
  failFast: true
liquibase:
  changeLog: classpath:/changelog/master.xml
  enabled: true
mybatis:
  config-location: classpath:mybatis-config.xml
```

```
package ee.cashier.product;
import ee.cashier.product.auth.SecurityFilter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.web.servlet.FilterRegistrationBean;
import org.springframework.cloud.client.SpringCloudApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.netflix.feign.EnableFeignClients;
import org.springframework.context.annotation.Bean;
import org.springframework.kafka.annotation.EnableKafka;
@EnableKafka
@EnableEurekaClient
@EnableFeignClients
@SpringCloudApplication
public class ProductApplication {
    @Autowired
    private SecurityFilter securityFilter;
    public static void main(String[] args) {
```

```
        SpringApplication.run(ProductApplication.class, args);
    }
    @Bean
    public FilterRegistrationBean requestLogFilterRegistration() {
        FilterRegistrationBean bean = new FilterRegistrationBean();
        bean.setFilter(securityFilter);
        bean.addUrlPatterns("/v1/*");
        bean.setOrder(0);
        return bean;
    }
}
```

```
package external.api;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import static org.springframework.http.MediaType.APPLICATION_JSON_UTF8_VALUE;
import static org.springframework.web.bind.annotation.RequestMethod.GET;
import static org.springframework.web.bind.annotation.RequestMethod.POST;
public interface ProductApi {
    @RequestMapping(value = "/v1/stores/{storeId}/products/create", method =
POST,
            consumes = APPLICATION_JSON_UTF8_VALUE, produces =
APPLICATION_JSON_UTF8_VALUE)
    ProductResponseJson create(@PathVariable("storeId") Long storeId, @Re-
questBody ProductCreateRequestJson json);
    @RequestMapping(value = "/v1/stores/{storeId}/products/{productId}",
method = GET,
            produces = APPLICATION_JSON_UTF8_VALUE)
    ProductResponseJson find(@PathVariable("storeId") Long storeId, @Path-
Variable("productId") Long productId);
    @RequestMapping(value = "/v1/stores/{storeId}/products/", method = GET,
produces = APPLICATION_JSON_UTF8_VALUE)
    ProductResponseJson search(@PathVariable("storeId") Long storeId, @Re-
questParam("barcode") String barcode);
    @RequestMapping(value = "/v1/stores/{storeId}/products/{productId}/price",
method = POST,
            consumes = APPLICATION_JSON_UTF8_VALUE, produces =
APPLICATION_JSON_UTF8_VALUE)
    ProductResponseJson changePrice(@PathVariable("storeId") Long storeId,
@PathVariable("productId") Long productId,
                                    @RequestBody ProductPriceChang-
eRequestJson json);
}
```

```
package external.api;
public class ProductCreateRequestJson {
    private String barcode;
    private String name;
    public String getBarcode() {
        return barcode;
    }
    public void setBarcode(String barcode) {
        this.barcode = barcode;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
```

```
        this.name = name;
    }
}
```

```
package external.api;
import ee.cashier.lang.daterange.DateRange;
import java.math.BigDecimal;
public class ProductPriceChangeRequestJson {
    private BigDecimal priceAmount;
    private String currencyCode;
    private DateRange validityRange;
    public BigDecimal getPriceAmount() {
        return priceAmount;
    }
    public void setPriceAmount(BigDecimal priceAmount) {
        this.priceAmount = priceAmount;
    }
    public String getCurrencyCode() {
        return currencyCode;
    }
    public void setCurrencyCode(String currencyCode) {
        this.currencyCode = currencyCode;
    }
    public DateRange getValidityRange() {
        return validityRange;
    }
    public void setValidityRange(DateRange validityRange) {
        this.validityRange = validityRange;
    }
}
```

```
package external.api;
import java.math.BigDecimal;
public class ProductResponseJson {
    private Long productId;
    private Long storeId;
    private String productName;
    private String currencyCode;
    private BigDecimal priceAmount;
    public Long getProductId() {
        return productId;
    }
    public void setProductId(Long productId) {
        this.productId = productId;
    }
    public Long getStoreId() {
        return storeId;
    }
    public void setStoreId(Long storeId) {
        this.storeId = storeId;
    }
    public String getProductName() {
        return productName;
    }
    public void setProductName(String productName) {
        this.productName = productName;
    }
    public String getCurrencyCode() {
        return currencyCode;
```

```
    }
    public void setCurrencyCode(String currencyCode) {
        this.currencyCode = currencyCode;
    }
    public BigDecimal getPriceAmount() {
        return priceAmount;
    }
    public void setPriceAmount(BigDecimal priceAmount) {
        this.priceAmount = priceAmount;
    }
}
```

```
package ee.cashier.product.api;
import ee.cashier.product.model.StoreProduct;
import ee.cashier.product.service.ChangeProductPriceService;
import ee.cashier.product.service.CreateProductService;
import ee.cashier.product.service.FindProductService;
import external.api.ProductCreateRequestJson;
import external.api.ProductPriceChangeRequestJson;
import external.api.ProductResponseJson;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import java.util.Optional;
@RestController
public class ProductApi implements external.api.ProductApi {
    @Autowired
    private CreateProductService productCreate;
    @Autowired
    private FindProductService productFind;
    @Autowired
    private ChangeProductPriceService priceChange;
    @Override
    public ProductResponseJson create(@PathVariable("storeId") Long storeId,
                                      @RequestBody ProductCreateRequestJ-
son json) {
        return productCreate.create(storeId, json).toProductResponseJson();
    }
    @Override
    public ProductResponseJson find(@PathVariable("storeId") Long storeId,
                                    @PathVariable("productId") Long pro-
ductId) {
        return Optional.ofNullable(productFind.find(storeId, productId))
                .map(StoreProduct::toProductResponseJson).orElse(null);
    }
    @Override
    public ProductResponseJson search(@PathVariable("storeId") Long storeId,
@RequestParam("barcode") String barcode) {
        return Optional.ofNullable(productFind.find(storeId, barcode))
                .map(StoreProduct::toProductResponseJson).orElse(null);
    }
    @Override
    public ProductResponseJson changePrice(@PathVariable("storeId") Long
storeId,
                                           @PathVariable("productId")
Long productId,
                                           @RequestBody Product-
PriceChangeRequestJson json) {
```

```
        return priceChange.change(storeId, productId, json).toProductRespon-
seJson();
    }
}
```

```
package ee.cashier.product.auth;
import external.api.UsernameResponseJson;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class AuthenticationContextHolder {
    private static final Logger log = LoggerFactory.getLogger(Authentication-
ContextHolder.class);
    private static final ThreadLocal<UsernameResponseJson> authentications =
new ThreadLocal<>();
    public static void put(UsernameResponseJson shardContext) {
        log.debug("Setting context");
        authentications.set(shardContext);
    }
    public static UsernameResponseJson get() {
        log.debug("Getting");
        return authentications.get();
    }
    public static void clear() {
        log.debug("Clearing");
        authentications.remove();
    }
}
```

```
package ee.cashier.product.auth;
import org.springframework.cloud.netflix.feign.FeignClient;
@FeignClient("auth-service")
public interface LoginApi extends external.api.LoginApi {
}
```

```
package ee.cashier.product.auth;
import external.api.UsernameResponseJson;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import javax.servlet.*;
import javax.servlet.http.HttpServletRequest;
import java.io.IOException;
@Component
public class SecurityFilter implements Filter {
    @Autowired
    private LoginApi loginApi;
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
    }
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain)
            throws IOException, ServletException {
        String authenticationToken = ((HttpServletRequest)
servletRequest).getHeader("x-auth");
        if (authenticationToken == null) {
            throw new RuntimeException("Authentication token is missing");
        }
        UsernameResponseJson tokenData = loginApi.findData(authenticationTo-
ken);
```

```
        if (tokenData == null) {
            throw new RuntimeException("Authentication token is not valid");
        }
        AuthenticationContextHolder.put(tokenData);
        filterChain.doFilter(servletRequest, servletResponse);
    }
    @Override
    public void destroy() {
    }
}
```

```
package ee.cashier.product.dao;
import ee.cashier.product.model.StoreProduct;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Param;
@Mapper
public interface StoreProductDao {
    void save(StoreProduct product);
    StoreProduct find(@Param("storeId") Long storeId, @Param("productId")
Long productId);
    StoreProduct findByBarcode(@Param("storeId") Long storeId, @Param("bar-
code") String barcode);
}
```

```
package ee.cashier.product.dao;
import ee.cashier.product.model.StoreProductPrice;
import org.apache.ibatis.annotations.Mapper;
@Mapper
public interface StoreProductPriceDao {
    void save(StoreProductPrice price);
}
```

```
package ee.cashier.product.model;
import external.api.ProductResponseJson;
import java.math.BigDecimal;
public class StoreProduct {
    private Long id;
    private Long storeId;
    private String name;
    private String barcode;
    private BigDecimal priceAmount;
    private String currencyCode;
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public Long getStoreId() {
        return storeId;
    }
    public void setStoreId(Long storeId) {
        this.storeId = storeId;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
```

```
    }
    public String getBarcode() {
        return barcode;
    }
    public void setBarcode(String barcode) {
        this.barcode = barcode;
    }
    public BigDecimal getPriceAmount() {
        return priceAmount;
    }
    public void setPriceAmount(BigDecimal priceAmount) {
        this.priceAmount = priceAmount;
    }
    public String getCurrencyCode() {
        return currencyCode;
    }
    public void setCurrencyCode(String currencyCode) {
        this.currencyCode = currencyCode;
    }
    public ProductResponseJson toProductResponseJson() {
        ProductResponseJson json = new ProductResponseJson();
        json.setProductId(id);
        json.setProductName(name);
        json.setStoreId(storeId);
        json.setPriceAmount(priceAmount);
        json.setCurrencyCode(currencyCode);
        return json;
    }
}
```

```
package ee.cashier.product.model;
import ee.cashier.lang.daterange.DateRange;
import java.math.BigDecimal;
public class StoreProductPrice {
    private Long id;
    private Long storeProductId;
    private BigDecimal priceAmount;
    private String currencyCode;
    private DateRange validityRange;
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public Long getStoreProductId() {
        return storeProductId;
    }
    public void setStoreProductId(Long storeProductId) {
        this.storeProductId = storeProductId;
    }
    public BigDecimal getPriceAmount() {
        return priceAmount;
    }
    public void setPriceAmount(BigDecimal priceAmount) {
        this.priceAmount = priceAmount;
    }
    public String getCurrencyCode() {
        return currencyCode;
```

```
    }
    public void setCurrencyCode(String currencyCode) {
        this.currencyCode = currencyCode;
    }
    public DateRange getValidityRange() {
        return validityRange;
    }
    public void setValidityRange(DateRange validityRange) {
        this.validityRange = validityRange;
    }
}
```

```
package ee.cashier.product.service;
import ee.cashier.product.dao.StoreProductPriceDao;
import ee.cashier.product.model.StoreProduct;
import ee.cashier.product.model.StoreProductPrice;
import external.api.ProductPriceChangeRequestJson;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;
import static ee.cashier.business.BusinessAssert.*;
import static java.math.BigDecimal.ZERO;
@Component
public class ChangeProductPriceService {
    @Autowired
    private StoreProductPriceDao priceDao;
    @Autowired
    private FindProductService findProductService;
    @Transactional
    public StoreProduct change(Long storeId, Long productId, Product-
PriceChangeRequestJson json) {
        //TODO: add base currency to store
        isEquals("EUR", json.getCurrencyCode(), "err.product.onlyEURAl-
lowed");
        isNotNull(json.getPriceAmount(), "err.product.priceIsMissing");
        isTrue(ZERO.compareTo(json.getPriceAmount()) < 0, "err.product.pri-
ceIsNegative");
        findProductService.assertExists(storeId, productId);
        //TODO: what to do if one already exists
        StoreProductPrice price = new StoreProductPrice();
        price.setStoreProductId(productId);
        price.setPriceAmount(json.getPriceAmount());
        price.setCurrencyCode(json.getCurrencyCode());
        price.setValidityRange(json.getValidityRange());
        priceDao.save(price);
        return findProductService.find(storeId, productId);
    }
}
```

```
package ee.cashier.product.service;
import ee.cashier.product.dao.StoreProductDao;
import ee.cashier.product.model.StoreProduct;
import external.api.ProductCreateRequestJson;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;
import static ee.cashier.business.BusinessAssert.isFalse;
import static ee.cashier.business.BusinessAssert.isNotEmpty;
@Component
public class CreateProductService {
```

```
    @Autowired
    private StoreProductDao productDao;
    @Autowired
    private FindProductService findProductService;
    @Transactional
    public StoreProduct create(Long storeId, ProductCreateRequestJson json) {
        isNotEmpty(json.getName(), "err.product.nameIsMissing");
        isNotEmpty(json.getBarcode(), "err.product.barcodeIsMissing");
        isFalse(findProductService.exists(storeId, json.getBarcode()), "er-
r.product.barcodeAlreadyExists");
        StoreProduct product = new StoreProduct();
        product.setStoreId(storeId);
        product.setName(json.getName());
        product.setBarcode(json.getBarcode());
        productDao.save(product);
        return product;
    }
}
```

```
package ee.cashier.product.service;
import ee.cashier.product.dao.StoreProductDao;
import ee.cashier.product.model.StoreProduct;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import static ee.cashier.business.BusinessAssert.isNotNull;
@Component
public class FindProductService {
    @Autowired
    private StoreProductDao productDao;
    public StoreProduct assertExists(Long storeId, Long productId) {
        return isNotNull(productDao.find(storeId, productId), "err.produc-
t.notFound");
    }
    public StoreProduct find(Long storeId, Long productId) {
        return productDao.find(storeId, productId);
    }
    public StoreProduct find(Long storeId, String barcode) {
        return productDao.findByBarcode(storeId, barcode);
    }
    public boolean exists(Long storeId, String barcode) {
        return find(storeId, barcode) != null;
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <settings>
        <setting name="cacheEnabled" value="false"/>
        <setting name="localCacheScope" value="STATEMENT"/>
    </settings>
    <typeAliases>
        <package name="ee.cashier.product.model"/>
        <package name="ee.cashier.domain"/>
    </typeAliases>
    <typeHandlers>
        <typeHandler handler="ee.cashier.lang.daterange.DateRangeTypeHan-
```

```
dler"/>
     </typeHandlers>
     <mappers>
          <mapper resource="dao/StoreProductDao.xml"/>
          <mapper resource="dao/StoreProductPriceDao.xml"/>
     </mappers>
</configuration>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://my-
batis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="ee.cashier.product.dao.StoreProductDao">
     <resultMap id="StoreProductResultMap" type="StoreProduct">
          <result column="id" property="id"/>
          <result column="store_id" property="storeId"/>
          <result column="name" property="name"/>
          <result column="barcode" property="barcode"/>
          <result column="price_amount" property="priceAmount"/>
          <result column="currency_code" property="currencyCode"/>
     </resultMap>
     <insert id="save" useGeneratedKeys="true" keyColumn="id" keyProper-
ty="id">
          INSERT INTO product.store_product (
               store_id,
               name,
               barcode
          ) VALUES (
               #{storeId},
               #{name},
               #{barcode}
          )
     </insert>
     <select id="find" resultMap="StoreProductResultMap">
          SELECT
               product.*,
               price.price_amount,
               price.currency_code
          FROM product.store_product product
               LEFT JOIN product.store_product_price price ON price.store_prod-
uct_id = product.id
                                                                  AND price.-
validity_range @> current_date
          WHERE product.store_id = #{storeId}
                AND product.id = #{productId}
     </select>
     <select id="findByBarcode" resultMap="StoreProductResultMap">
          SELECT
               product.*,
               price.price_amount,
               price.currency_code
          FROM product.store_product product
               LEFT JOIN product.store_product_price price ON price.store_prod-
uct_id = product.id
                                                                  AND price.-
validity_range @> current_date
          WHERE product.store_id = #{storeId}
                AND product.barcode = #{barcode}
     </select>
```

```
</mapper>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://my-
batis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="ee.cashier.product.dao.StoreProductPriceDao">
    <resultMap id="StoreProductPriceResultMap" type="StoreProductPrice">
        <result column="id" property="id"/>
        <result column="store_product_id" property="storeProductId"/>
        <result column="price_amount" property="priceAmount"/>
        <result column="currency_code" property="currencyCode"/>
        <result column="validity_range" property="validityRange"/>
    </resultMap>
    <insert id="save" useGeneratedKeys="true" keyColumn="id" keyProper-
ty="id">
        INSERT INTO product.store_product_price (
            store_product_id,
            price_amount,
            currency_code,
            validity_range
        ) VALUES (
            #{storeProductId},
            #{priceAmount},
            #{currencyCode},
            #{validityRange} :: DATERANGE
        )
    </insert>
</mapper>
```

Lisa 6 – Toote teenuse programmikood

```
dependencies {
     compile project(":domain-common")
     compile project(":domain-common-database")
     compile project(":business-assert")
     compile project(":purchase-service-api")
     compile project(":auth-service-api")
     compile project(":product-service-api")
     compile "org.springframework.boot:spring-boot-starter-web"
     compile "org.springframework.boot:spring-boot-starter-actuator:$spring-
BootVersion"
     compile "org.springframework.cloud:spring-cloud-starter-config"
     compile "org.springframework.cloud:spring-cloud-starter-eureka"
     compile "org.springframework.cloud:spring-cloud-starter-feign"
     compile "org.springframework.cloud:spring-cloud-starter-hystrix"
     compile "org.springframework.cloud:spring-cloud-starter-stream-kafka"
     compile "com.fasterxml.jackson.datatype:jackson-datatype-jsr310:2.8.3"
     compile "org.postgresql:postgresql:$postgresqlVersion"
     compile "org.liquibase:liquibase-core:$liquibaseCoreVersion"
     compile "org.springframework:spring-jdbc"
     compile "com.zaxxer:HikariCP:2.6.1"
     compile "org.mybatis.spring.boot:mybatis-spring-boot-starter:1.3.0"
     compile "org.mybatis:mybatis-typehandlers-jsr310:1.0.2"
}
```

```
spring.application:
  name: purchase-service
spring.cloud.config:
  uri: http://localhost:9999/config
  failFast: true
liquibase:
  changeLog: classpath:/changelog/master.xml
  enabled: true
mybatis:
  config-location: classpath:mybatis-config.xml
```

```
package ee.cashier.purchase;
import ee.cashier.purchase.auth.SecurityFilter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.web.servlet.FilterRegistrationBean;
import org.springframework.cloud.client.SpringCloudApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.netflix.feign.EnableFeignClients;
import org.springframework.context.annotation.Bean;
import org.springframework.kafka.annotation.EnableKafka;
@EnableKafka
@EnableEurekaClient
@EnableFeignClients
@SpringCloudApplication
public class PurchaseApplication {
```

```
    @Autowired
    private SecurityFilter securityFilter;
    public static void main(String[] args) {
        SpringApplication.run(PurchaseApplication.class, args);
    }
    @Bean
    public FilterRegistrationBean requestLogFilterRegistration() {
        FilterRegistrationBean bean = new FilterRegistrationBean();
        bean.setFilter(securityFilter);
        bean.addUrlPatterns("/v1/*");
        bean.setOrder(0);
        return bean;
    }
}
```

```
package external.api;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import java.util.List;
import static org.springframework.http.MediaType.APPLICATION_JSON_UTF8_VALUE;
import static org.springframework.web.bind.annotation.RequestMethod.GET;
import static org.springframework.web.bind.annotation.RequestMethod.POST;
public interface PurchaseApi {
    @RequestMapping(value = "/v1/stores/{storeId}/purchases/create", method =
POST,
            produces = APPLICATION_JSON_UTF8_VALUE)
    PurchaseResponseJson create(@PathVariable("storeId") Long storeId);
    @RequestMapping(value = "/v1/customers/{customerId}/purchases", method =
GET,
            produces = APPLICATION_JSON_UTF8_VALUE)
    List<PurchaseResponseJson> findByCustomer(@PathVariable("customerId")
Long customerId);
    @RequestMapping(value = "/v1/purchases/{purchaseId}", method = GET,
            produces = APPLICATION_JSON_UTF8_VALUE)
    PurchaseResponseJson find(@PathVariable("purchaseId") Long purchaseId);
    @RequestMapping(value = "/v1/purchases/{purchaseId}/products/
{productId}/add", method = POST,
            produces = APPLICATION_JSON_UTF8_VALUE)
    PurchaseResponseJson addProduct(@PathVariable("purchaseId") Long purcha-
seId,
                                    @PathVariable("productId") Long pro-
ductId);
    @RequestMapping(value = "/v1/purchases/{purchaseId}/products/
{productId}/remove", method = POST,
            produces = APPLICATION_JSON_UTF8_VALUE)
    PurchaseResponseJson removeProduct(@PathVariable("purchaseId") Long pur-
chaseId,
                                       @PathVariable("productId") Long
productId);
    @RequestMapping(value = "/v1/purchases/{purchaseId}/pay", method = POST,
            produces = APPLICATION_JSON_UTF8_VALUE)
    PurchaseResponseJson pay(@PathVariable("purchaseId") Long purchaseId);
}
```

```
package external.api;
import java.math.BigDecimal;
public class PurchaseLineResponseJson {
    private Long id;
    private Long productId;
    private int productCount;
```

```
    private BigDecimal unitPriceAmount;
    private String unitCurrencyCode;
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public Long getProductId() {
        return productId;
    }
    public void setProductId(Long productId) {
        this.productId = productId;
    }
    public int getProductCount() {
        return productCount;
    }
    public void setProductCount(int productCount) {
        this.productCount = productCount;
    }
    public BigDecimal getUnitPriceAmount() {
        return unitPriceAmount;
    }
    public void setUnitPriceAmount(BigDecimal unitPriceAmount) {
        this.unitPriceAmount = unitPriceAmount;
    }
    public String getUnitCurrencyCode() {
        return unitCurrencyCode;
    }
    public void setUnitCurrencyCode(String unitCurrencyCode) {
        this.unitCurrencyCode = unitCurrencyCode;
    }
}
```

```
package external.api;
import java.math.BigDecimal;
import java.util.List;
import static java.util.Collections.emptyList;
public class PurchaseResponseJson {
    private Long purchaseId;
    private Long storeId;
    private Long paypointManagerId;
    private Long customerId;
    private String statusCode;
    private BigDecimal priceAmount;
    private String currencyCode;
    private List<PurchaseLineResponseJson> lines = emptyList();
    public Long getPurchaseId() {
        return purchaseId;
    }
    public void setPurchaseId(Long purchaseId) {
        this.purchaseId = purchaseId;
    }
    public Long getStoreId() {
        return storeId;
    }
    public void setStoreId(Long storeId) {
        this.storeId = storeId;
    }
```

```
    public Long getPaypointManagerId() {
        return paypointManagerId;
    }
    public void setPaypointManagerId(Long paypointManagerId) {
        this.paypointManagerId = paypointManagerId;
    }
    public Long getCustomerId() {
        return customerId;
    }
    public void setCustomerId(Long customerId) {
        this.customerId = customerId;
    }
    public String getStatusCode() {
        return statusCode;
    }
    public void setStatusCode(String statusCode) {
        this.statusCode = statusCode;
    }
    public BigDecimal getPriceAmount() {
        return priceAmount;
    }
    public void setPriceAmount(BigDecimal priceAmount) {
        this.priceAmount = priceAmount;
    }
    public String getCurrencyCode() {
        return currencyCode;
    }
    public void setCurrencyCode(String currencyCode) {
        this.currencyCode = currencyCode;
    }
    public List<PurchaseLineResponseJson> getLines() {
        return lines;
    }
    public void setLines(List<PurchaseLineResponseJson> lines) {
        this.lines = lines;
    }
}
```

```
package ee.cashier.purchase.api;
import ee.cashier.purchase.model.Purchase;
import ee.cashier.purchase.service.*;
import external.api.PurchaseResponseJson;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
import java.util.List;
import static ee.cashier.purchase.auth.AuthenticationContextHolder.get;
import static java.util.stream.Collectors.toList;
@RestController
public class PurchaseApi implements external.api.PurchaseApi {
    @Autowired
    private CreatePurchaseService createPurchase;
    @Autowired
    private CreatePurchaseLineService createPurchaseLine;
    @Autowired
    private RemovePurchaseLineService removePurchaseLine;
    @Autowired
    private FindPurchaseService findPurchase;
    @Autowired
    private PayPurchaseService payPurchase;
```

```
    @Override
    public PurchaseResponseJson create(@PathVariable("storeId") Long storeId)
{
        return createPurchase.create(storeId, get().getCustomerId()).toPur-
chaseResponseJson();
    }
    @Override
    public List<PurchaseResponseJson> findByCustomer(@PathVariable("cus-
tomerId") Long customerId) {
        return findPurchase.findByCustomer(customerId).stream().map(Pur-
chase::toPurchaseResponseJson).collect(toList());
    }
    @Override
    public PurchaseResponseJson find(@PathVariable("purchaseId") Long purcha-
seId) {
        return findPurchase.find(get().getCustomerId(), purchaseId).toPur-
chaseResponseJson();
    }
    @Override
    public PurchaseResponseJson addProduct(@PathVariable("purchaseId") Long
purchaseId,
                                           @PathVariable("productId")
Long productId) {
        return createPurchaseLine.create(get().getCustomerId(), purchaseId,
productId).toPurchaseResponseJson();
    }
    @Override
    public PurchaseResponseJson removeProduct(@PathVariable("purchaseId")
Long purchaseId,
                                              @PathVariable("productId")
Long productId) {
        return removePurchaseLine.remove(get().getCustomerId(), purchaseId,
productId).toPurchaseResponseJson();
    }
    @Override
    public PurchaseResponseJson pay(@PathVariable("purchaseId") Long purcha-
seId) {
        return payPurchase.pay(get().getCustomerId(), purchaseId).toPur-
chaseResponseJson();
    }
}
```

```
package ee.cashier.purchase.auth;
import external.api.UsernameResponseJson;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class AuthenticationContextHolder {
    private static final Logger log = LoggerFactory.getLogger(Authentication-
ContextHolder.class);
    private static final ThreadLocal<UsernameResponseJson> authentications =
new ThreadLocal<>();
    public static void put(UsernameResponseJson shardContext) {
        log.debug("Setting context");
        authentications.set(shardContext);
    }
    public static UsernameResponseJson get() {
        log.debug("Getting");
        return authentications.get();
    }
    public static void clear() {
```

```
            log.debug("Clearing");
            authentications.remove();
        }
}
```

```
package ee.cashier.purchase.auth;
import org.springframework.cloud.netflix.feign.FeignClient;
@FeignClient("auth-service")
public interface LoginApi extends external.api.LoginApi {
}
```

```
package ee.cashier.purchase.auth;
import external.api.UsernameResponseJson;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import javax.servlet.*;
import javax.servlet.http.HttpServletRequest;
import java.io.IOException;
@Component
public class SecurityFilter implements Filter {
    @Autowired
    private LoginApi loginApi;
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
    }
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain)
            throws IOException, ServletException {
        String authenticationToken = ((HttpServletRequest)
servletRequest).getHeader("x-auth");
        if (authenticationToken == null) {
            throw new RuntimeException("Authentication token is missing");
        }
        UsernameResponseJson tokenData = loginApi.findData(authenticationTo-
ken);
        if (tokenData == null) {
            throw new RuntimeException("Authentication token is not valid");
        }
        AuthenticationContextHolder.put(tokenData);
        filterChain.doFilter(servletRequest, servletResponse);
    }
    @Override
    public void destroy() {
    }
}
```

```
package ee.cashier.purchase.dao;
import ee.cashier.purchase.model.Purchase;
import ee.cashier.purchase.model.Purchase.StatusCode;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Param;
import java.math.BigDecimal;
import java.util.List;
@Mapper
public interface PurchaseDao {
    void save(Purchase purchase);
    Purchase find(Long purchaseId);
    int updateStatus(@Param("purchaseId") Long purchaseId, @Param("status-
```

```
Code") StatusCode pendingPayment);
    int updatePrice(@Param("purchaseId") Long purchaseId,
@Param("priceAmount") BigDecimal priceAmount);
    List<Purchase> findByCustomer(Long customerId);
}
```

```
package ee.cashier.purchase.dao;
import ee.cashier.purchase.model.PurchaseLine;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Param;
import java.util.List;
@Mapper
public interface PurchaseLineDao {
    void save(PurchaseLine line);
    List<PurchaseLine> findByPurchase(Long purchaseId);
    PurchaseLine findByPurchaseAndProduct(@Param("purchaseId") Long purcha-
seId, @Param("productId") Long productId);
    int updateCount(@Param("purchaseLineId") Long purchaseLineId,
@Param("productCount") int productCount);
    int remove(Long purchaseLineId);
}
```

```
package ee.cashier.purchase.external;
import org.springframework.cloud.netflix.feign.FeignClient;
@FeignClient("product-service")
public interface ProductApi extends external.api.ProductApi {
}
```

```
package ee.cashier.purchase.model;
import external.api.PurchaseResponseJson;
import java.math.BigDecimal;
import java.util.List;
import static java.util.Collections.emptyList;
import static java.util.stream.Collectors.toList;
public class Purchase {
    private Long id;
    private Long storeId;
    private Long paypointManagerId;
    private Long customerId;
    private StatusCode statusCode;
    private BigDecimal priceAmount;
    private String currencyCode;
    private List<PurchaseLine> lines = emptyList();
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public Long getStoreId() {
        return storeId;
    }
    public void setStoreId(Long storeId) {
        this.storeId = storeId;
    }
    public Long getPaypointManagerId() {
        return paypointManagerId;
    }
    public void setPaypointManagerId(Long paypointManagerId) {
```

```
        this.paypointManagerId = paypointManagerId;
    }
    public Long getCustomerId() {
        return customerId;
    }
    public void setCustomerId(Long customerId) {
        this.customerId = customerId;
    }
    public StatusCode getStatusCode() {
        return statusCode;
    }
    public void setStatusCode(StatusCode statusCode) {
        this.statusCode = statusCode;
    }
    public BigDecimal getPriceAmount() {
        return priceAmount;
    }
    public void setPriceAmount(BigDecimal priceAmount) {
        this.priceAmount = priceAmount;
    }
    public String getCurrencyCode() {
        return currencyCode;
    }
    public void setCurrencyCode(String currencyCode) {
        this.currencyCode = currencyCode;
    }
    public List<PurchaseLine> getLines() {
        return lines;
    }
    public void setLines(List<PurchaseLine> lines) {
        this.lines = lines;
    }
    public PurchaseResponseJson toPurchaseResponseJson() {
        PurchaseResponseJson json = new PurchaseResponseJson();
        json.setPurchaseId(id);
        json.setStoreId(storeId);
        json.setPaypointManagerId(paypointManagerId);
        json.setCustomerId(customerId);
        json.setStatusCode(statusCode.name());
        json.setPriceAmount(priceAmount);
        json.setCurrencyCode(currencyCode);

json.setLines(lines.stream().map(PurchaseLine::toPurchaseLineJson).collect(to
List()));
        return json;
    }
    public enum StatusCode {
        INPROGRESS, PENDING_PAYMENT, PAID
    }
}
```

```
package ee.cashier.purchase.model;
import external.api.PurchaseLineResponseJson;
import java.math.BigDecimal;
public class PurchaseLine {
    private Long id;
    private Long purchaseId;
    private Long storeProductId;
    private int productCount;
```

```
    private BigDecimal unitPriceAmount;
    private String unitCurrencyCode;
    public PurchaseLineResponseJson toPurchaseLineJson() {
        PurchaseLineResponseJson json = new PurchaseLineResponseJson();
        json.setId(id);
        json.setProductId(storeProductId);
        json.setProductCount(productCount);
        json.setUnitPriceAmount(unitPriceAmount);
        json.setUnitCurrencyCode(unitCurrencyCode);
        return json;
    }
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public Long getPurchaseId() {
        return purchaseId;
    }
    public void setPurchaseId(Long purchaseId) {
        this.purchaseId = purchaseId;
    }
    public Long getStoreProductId() {
        return storeProductId;
    }
    public void setStoreProductId(Long storeProductId) {
        this.storeProductId = storeProductId;
    }
    public int getProductCount() {
        return productCount;
    }
    public void setProductCount(int productCount) {
        this.productCount = productCount;
    }
    public BigDecimal getUnitPriceAmount() {
        return unitPriceAmount;
    }
    public void setUnitPriceAmount(BigDecimal unitPriceAmount) {
        this.unitPriceAmount = unitPriceAmount;
    }
    public String getUnitCurrencyCode() {
        return unitCurrencyCode;
    }
    public void setUnitCurrencyCode(String unitCurrencyCode) {
        this.unitCurrencyCode = unitCurrencyCode;
    }
}
```

```
package ee.cashier.purchase.publish;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Component;
@Component
public class EventPublisher {
    @Autowired
    private KafkaTemplate<Object, Object> kafkaTemplate;
```

```
    @Autowired
    private ObjectMapper objectMapper;
    protected void publish(String topic, Object event) {
        kafkaTemplate.send(topic, convertToJson(event)).addCallback(
                o -> System.out.println("Success " + o),
                throwable -> {
                    System.err.println(throwable.getMessage());
                    throwable.printStackTrace();
                });
    }
    private String convertToJson(Object event) {
        try {
            return objectMapper.writeValueAsString(event);
        } catch (JsonProcessingException e) {
            throw new RuntimeException(e);
        }
    }
}
```

```
package ee.cashier.purchase.publish;
import java.math.BigDecimal;
public class PurchaseConfirmedEvent {
    private Long purchaseId;
    private BigDecimal priceAmount;
    private String currencyCode;
    public Long getPurchaseId() {
        return purchaseId;
    }
    public void setPurchaseId(Long purchaseId) {
        this.purchaseId = purchaseId;
    }
    public BigDecimal getPriceAmount() {
        return priceAmount;
    }
    public void setPriceAmount(BigDecimal priceAmount) {
        this.priceAmount = priceAmount;
    }
    public String getCurrencyCode() {
        return currencyCode;
    }
    public void setCurrencyCode(String currencyCode) {
        this.currencyCode = currencyCode;
    }
}
```

```
package ee.cashier.purchase.publish;
import ee.cashier.purchase.model.Purchase;
import ee.cashier.purchase.service.FindPurchaseService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
@Component
public class PurchaseConfirmedEventPublisher {
    private final String eventName = "purchase-confirmed";
    @Autowired
    private EventPublisher eventPublisher;
    @Autowired
    private FindPurchaseService findPurchase;
    public void publish(Long customerId, Long purchaseId) {
        Purchase purchase = findPurchase.find(customerId, purchaseId);
```

```
        PurchaseConfirmedEvent event = new PurchaseConfirmedEvent();
        event.setPurchaseId(purchase.getId());
        event.setPriceAmount(purchase.getPriceAmount());
        event.setCurrencyCode(purchase.getCurrencyCode());
        eventPublisher.publish(eventName, event);
    }
}
```

```
package ee.cashier.purchase.service;
import ee.cashier.purchase.dao.PurchaseDao;
import ee.cashier.purchase.dao.PurchaseLineDao;
import ee.cashier.purchase.external.ProductApi;
import ee.cashier.purchase.model.Purchase;
import ee.cashier.purchase.model.PurchaseLine;
import external.api.ProductResponseJson;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;
import java.math.BigDecimal;
import static ee.cashier.business.BusinessAssert.isEquals;
import static ee.cashier.business.BusinessAssert.isNotNull;
import static ee.cashier.business.UpdateResult.updateResult;
import static ee.cashier.purchase.model.Purchase.StatusCode.INPROGRESS;
@Component
public class CreatePurchaseLineService {
    @Autowired
    private FindPurchaseService findPurchase;
    @Autowired
    private FindPurchaseLineService findPurchaseLine;
    @Autowired
    private ProductApi productApi;
    @Autowired
    private PurchaseDao purchaseDao;
    @Autowired
    private PurchaseLineDao purchaseLineDao;
    @Transactional
    public Purchase create(Long customerId, Long purchaseId, Long productId)
{
        Purchase purchase = findPurchase.find(customerId, purchaseId);
        isEquals(INPROGRESS, purchase.getStatusCode(), "err.purchase.notAl-
lowed");
        PurchaseLine line = findPurchaseLine.find(purchase.getId(), produc-
tId);
        ProductResponseJson product = isNotNull(productApi.find(pur-
chase.getStoreId(), productId),
                "err.purchase.productNotFound");
        if (line != null) {
            updateResult(purchaseLineDao.updateCount(line.getId(), line.get-
ProductCount() + 1))
                    .assertSingle();
        } else {
            line = new PurchaseLine();
            line.setPurchaseId(purchase.getId());
            line.setStoreProductId(product.getProductId());
            line.setProductCount(1);
            line.setUnitPriceAmount(product.getPriceAmount());
            line.setUnitCurrencyCode(product.getCurrencyCode());
            purchaseLineDao.save(line);
        }
        BigDecimal newTotalPrice = purchase.getPriceAmount().add(produc-
t.getPriceAmount());
```

```
        updateResult(purchaseDao.updatePrice(purchase.getId(), newTotal-
Price)).assertSingle();
        return findPurchase.find(customerId, purchaseId);
    }
}
```

```
package ee.cashier.purchase.service;
import ee.cashier.purchase.dao.PurchaseDao;
import ee.cashier.purchase.model.Purchase;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;
import static ee.cashier.purchase.model.Purchase.StatusCode.INPROGRESS;
import static java.math.BigDecimal.ZERO;
@Component
public class CreatePurchaseService {
    @Autowired
    private PurchaseDao purchaseDao;
    @Transactional
    public Purchase create(Long storeId, Long customerId) {
        Purchase purchase = new Purchase();
        purchase.setStoreId(storeId);
        purchase.setCustomerId(customerId);
        purchase.setPriceAmount(ZERO);
        purchase.setCurrencyCode("EUR");
        purchase.setStatusCode(INPROGRESS);
        purchaseDao.save(purchase);
        return purchase;
    }
}
```

```
package ee.cashier.purchase.service;
import ee.cashier.purchase.dao.PurchaseLineDao;
import ee.cashier.purchase.model.PurchaseLine;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import static ee.cashier.business.BusinessAssert.isNotNull;
@Component
public class FindPurchaseLineService {
    @Autowired
    private PurchaseLineDao purchaseLineDao;
    public PurchaseLine find(Long purchaseId, Long productId) {
        return purchaseLineDao.findByPurchaseAndProduct(purchaseId, produc-
tId);
    }
    public PurchaseLine require(Long purchaseId, Long productId) {
        return isNotNull(find(purchaseId, productId), "err.purchase.pur-
chaseLineNotFound");
    }
}
```

```
package ee.cashier.purchase.service;
import ee.cashier.purchase.dao.PurchaseDao;
import ee.cashier.purchase.model.Purchase;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import java.util.List;
import static ee.cashier.business.BusinessAssert.isEquals;
@Component
public class FindPurchaseService {
```

```
    @Autowired
    private PurchaseDao purchaseDao;
    public Purchase find(Long customerId, Long purchaseId) {
        Purchase purchase = purchaseDao.find(purchaseId);
        isEquals(customerId, purchase.getCustomerId(), "err.purchase.does-
NotBelongToCustomer");
        return purchase;
    }
    public List<Purchase> findByCustomer(Long customerId) {
        return purchaseDao.findByCustomer(customerId);
    }
}
```

```
package ee.cashier.purchase.service;
import ee.cashier.purchase.dao.PurchaseDao;
import ee.cashier.purchase.model.Purchase;
import ee.cashier.purchase.publish.PurchaseConfirmedEventPublisher;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import static ee.cashier.business.BusinessAssert.isEquals;
import static ee.cashier.business.UpdateResult.updateResult;
import static ee.cashier.purchase.model.Purchase.StatusCode.INPROGRESS;
import static ee.cashier.purchase.model.Purchase.StatusCode.PENDING_PAYMENT;
@Component
public class PayPurchaseService {
    @Autowired
    private FindPurchaseService findPurchase;
    @Autowired
    private PurchaseDao purchaseDao;
    @Autowired
    private PurchaseConfirmedEventPublisher eventPublisher;
    public Purchase pay(Long customerId, Long purchaseId) {
        Purchase purchase = findPurchase.find(customerId, purchaseId);
        isEquals(INPROGRESS, purchase.getStatusCode(), "err.purchase.notAl-
lowed");
        updateResult(purchaseDao.updateStatus(purchase.getId(), PENDING_PAY-
MENT)).assertSingle();
        eventPublisher.publish(customerId, purchaseId);
        return findPurchase.find(customerId, purchaseId);
    }
}
```

```
package ee.cashier.purchase.service;
import ee.cashier.purchase.dao.PurchaseDao;
import ee.cashier.purchase.dao.PurchaseLineDao;
import ee.cashier.purchase.model.Purchase;
import ee.cashier.purchase.model.PurchaseLine;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;
import java.math.BigDecimal;
import static ee.cashier.business.BusinessAssert.isEquals;
import static ee.cashier.business.UpdateResult.updateResult;
import static ee.cashier.purchase.model.Purchase.StatusCode.INPROGRESS;
@Component
public class RemovePurchaseLineService {
    @Autowired
    private FindPurchaseService findPurchase;
    @Autowired
    private FindPurchaseLineService findPurchaseLine;
```

```
    @Autowired
    private PurchaseDao purchaseDao;
    @Autowired
    private PurchaseLineDao purchaseLineDao;
    @Transactional
    public Purchase remove(Long customerId, Long purchaseId, Long productId)
{
        Purchase purchase = findPurchase.find(customerId, purchaseId);
        isEquals(INPROGRESS, purchase.getStatusCode(), "err.purchase.notAl-
lowed");
        PurchaseLine line = findPurchaseLine.require(purchase.getId(), pro-
ductId);
        if (line.getProductCount() > 1) {
            updateResult(purchaseLineDao.updateCount(line.getId(), line.get-
ProductCount() - 1))
                    .assertSingle();
        } else {

updateResult(purchaseLineDao.remove(line.getId())).assertSingle();
        }
        BigDecimal newTotalPrice =
purchase.getPriceAmount().subtract(line.getUnitPriceAmount());
        updateResult(purchaseDao.updatePrice(purchase.getId(), newTotal-
Price)).assertSingle();
        return findPurchase.find(customerId, purchaseId);
    }
}
```

```
package ee.cashier.purchase.subscribe;
import ee.cashier.domain.Source;
public class PurchasePaidEvent {
    private Source source;
    public Source getSource() {
        return source;
    }
    public void setSource(Source source) {
        this.source = source;
    }
}
```

```
package ee.cashier.purchase.subscribe;
import com.fasterxml.jackson.databind.ObjectMapper;
import ee.cashier.purchase.dao.PurchaseDao;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;
import java.io.IOException;
import static ee.cashier.business.UpdateResult.updateResult;
import static ee.cashier.purchase.model.Purchase.StatusCode.PAID;
@Component
public class PurchasePaidSubscriber {
    @Autowired
    private ObjectMapper objectMapper;
    @Autowired
    private PurchaseDao purchaseDao;
    @Transactional
    @KafkaListener(topics = "purchase-paid")
    public void consume(String message) throws IOException {
        PurchasePaidEvent messageObject = objectMapper.readValue(message,
```

```
PurchasePaidEvent.class);
        updateResult(purchaseDao.updateStatus(messageObject.getSource().ge-
tRef(), PAID)).assertSingle();
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <settings>
        <setting name="cacheEnabled" value="false"/>
        <setting name="localCacheScope" value="STATEMENT"/>
    </settings>
    <typeAliases>
        <package name="ee.cashier.purchase.model"/>
        <package name="ee.cashier.domain"/>
    </typeAliases>
    <typeHandlers>
        <typeHandler handler="ee.cashier.lang.daterange.DateRangeTypeHan-
dler"/>
    </typeHandlers>
    <mappers>
        <mapper resource="dao/PurchaseDao.xml"/>
        <mapper resource="dao/PurchaseLineDao.xml"/>
    </mappers>
</configuration>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://my-
batis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="ee.cashier.purchase.dao.PurchaseDao">
    <resultMap id="PurchaseResultMap" type="Purchase">
        <result column="id" property="id"/>
        <result column="store_id" property="storeId"/>
        <result column="paypoint_manager_id" property="paypointManagerId"/>
        <result column="customer_id" property="customerId"/>
        <result column="status_code" property="statusCode"/>
        <result column="price_amount" property="priceAmount"/>
        <result column="currency_code" property="currencyCode"/>
        <collection property="lines"
                    column="id"
                    select="ee.cashier.purchase.dao.PurchaseLineDao.findBy-
Purchase"
        />
    </resultMap>
    <insert id="save" useGeneratedKeys="true" keyColumn="id" keyProper-
ty="id">
        INSERT INTO purchase.purchase (
            store_id,
            paypoint_manager_id,
            customer_id,
            status_code,
            price_amount,
            currency_code
        ) VALUES (
            #{storeId},
            #{paypointManagerId},
```

```
            #{customerId},
            #{statusCode},
            #{priceAmount},
            #{currencyCode}
        )
    </insert>
    <select id="find" resultMap="PurchaseResultMap">
        SELECT *
        FROM purchase.purchase
        WHERE id = #{purchaseId}
    </select>
    <select id="findByCustomer" resultMap="PurchaseResultMap">
        SELECT *
        FROM purchase.purchase
        WHERE customer_id = #{customerId}
    </select>
    <update id="updateStatus">
        UPDATE purchase.purchase
        SET status_code = #{statusCode}
        WHERE id = #{purchaseId}
    </update>
    <update id="updatePrice">
        UPDATE purchase.purchase
        SET price_amount = #{priceAmount}
        WHERE id = #{purchaseId}
    </update>
</mapper>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://my-
batis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="ee.cashier.purchase.dao.PurchaseLineDao">
    <resultMap id="PurchaseLineResultMap" type="PurchaseLine">
        <result column="id" property="id"/>
        <result column="purchase_id" property="purchaseId"/>
        <result column="store_product_id" property="storeProductId"/>
        <result column="product_count" property="productCount"/>
        <result column="unit_price_amount" property="unitPriceAmount"/>
        <result column="unit_currency_code" property="unitCurrencyCode"/>
    </resultMap>
    <insert id="save" useGeneratedKeys="true" keyProperty="id" keyCol-
umn="id">
        INSERT INTO purchase.purchase_line (
            purchase_id,
            store_product_id,
            product_count,
            unit_price_amount,
            unit_currency_code
        ) VALUES (
            #{purchaseId},
            #{storeProductId},
            #{productCount},
            #{unitPriceAmount},
            #{unitCurrencyCode}
        )
    </insert>
    <select id="findByPurchase" resultMap="PurchaseLineResultMap">
        SELECT *
        FROM purchase.purchase_line
```

```
        WHERE purchase_id = #{purchaseId}
    </select>
    <select id="findByPurchaseAndProduct" resultMap="PurchaseLineResultMap">
        SELECT *
        FROM purchase.purchase_line
        WHERE purchase_id = #{purchaseId}
              AND store_product_id = #{productId}
    </select>
    <update id="updateCount">
        UPDATE purchase.purchase_line
        SET product_count = #{productCount}
        WHERE id = #{purchaseLineId}
    </update>
    <delete id="remove">
        DELETE
        FROM purchase.purchase_line
        WHERE id = #{purchaseLineId}
    </delete>
</mapper>
```

Lisa 7 – Kaupluse teenuse programmikood

```
dependencies {
    compile project(":domain-common")
    compile project(":domain-common-database")
    compile project(":business-assert")
    compile project(":store-service-api")
    compile project(":auth-service-api")
    compile "org.springframework.boot:spring-boot-starter-web"
    compile "org.springframework.boot:spring-boot-starter-actuator:$spring-
BootVersion"
    compile "org.springframework.cloud:spring-cloud-starter-config"
    compile "org.springframework.cloud:spring-cloud-starter-eureka"
    compile "org.springframework.cloud:spring-cloud-starter-feign"
    compile "org.springframework.cloud:spring-cloud-starter-hystrix"
    compile "org.springframework.cloud:spring-cloud-starter-stream-kafka"
    compile "com.fasterxml.jackson.datatype:jackson-datatype-jsr310:2.8.3"
    compile "org.postgresql:postgresql:$postgresqlVersion"
    compile "org.liquibase:liquibase-core:$liquibaseCoreVersion"
    compile "org.springframework:spring-jdbc"
    compile "com.zaxxer:HikariCP:2.6.1"
    compile "org.mybatis.spring.boot:mybatis-spring-boot-starter:1.3.0"
    compile "org.mybatis:mybatis-typehandlers-jsr310:1.0.2"
}
```

```
spring.application:
  name: store-service
spring.cloud.config:
  uri: http://localhost:9999/config
  failFast: true
liquibase:
  changeLog: classpath:/changelog/master.xml
  enabled: true
mybatis:
  config-location: classpath:mybatis-config.xml
```

```
package ee.cashier.store;
import ee.cashier.store.auth.SecurityFilter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.web.servlet.FilterRegistrationBean;
import org.springframework.cloud.client.SpringCloudApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.netflix.feign.EnableFeignClients;
import org.springframework.context.annotation.Bean;
import org.springframework.kafka.annotation.EnableKafka;
@EnableKafka
@EnableEurekaClient
@EnableFeignClients
@SpringCloudApplication
public class StoreApplication {
    @Autowired
```

```
    private SecurityFilter securityFilter;
    public static void main(String[] args) {
        SpringApplication.run(StoreApplication.class, args);
    }
    @Bean
    public FilterRegistrationBean requestLogFilterRegistration() {
        FilterRegistrationBean bean = new FilterRegistrationBean();
        bean.setFilter(securityFilter);
        bean.addUrlPatterns("/v1/*");
        bean.setOrder(0);
        return bean;
    }
}
```

```
package external.api;
public class CreateStoreRequestJson {
    private String name;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

```
package external.api;
public class StoreResponseJson {
    private Long storeId;
    private String barcode;
    private String storeName;
    public Long getStoreId() {
        return storeId;
    }
    public void setStoreId(Long storeId) {
        this.storeId = storeId;
    }
    public String getBarcode() {
        return barcode;
    }
    public void setBarcode(String barcode) {
        this.barcode = barcode;
    }
    public String getStoreName() {
        return storeName;
    }
    public void setStoreName(String storeName) {
        this.storeName = storeName;
    }
}
```

```
package external.api;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import static org.springframework.http.MediaType.APPLICATION_JSON_UTF8_VALUE;
import static org.springframework.web.bind.annotation.RequestMethod.GET;
import static org.springframework.web.bind.annotation.RequestMethod.POST;
```

```
public interface StoreApi {
    @RequestMapping(value = "/v1/stores/create", method = POST, consumes =
APPLICATION_JSON_UTF8_VALUE)
    StoreResponseJson create(@RequestBody CreateStoreRequestJson json);
    @RequestMapping(value = "/v1/stores/search", method = GET, produces = AP-
PLICATION_JSON_UTF8_VALUE)
    StoreResponseJson search(@RequestParam("barcode") String barcode);
    @RequestMapping(value = "/v1/stores/{storeId}", method = GET, produces =
APPLICATION_JSON_UTF8_VALUE)
    StoreResponseJson find(@PathVariable("storeId") Long storeId);
}
```

```
package ee.cashier.store.api;
import ee.cashier.store.model.Store;
import ee.cashier.store.service.CreateStoreService;
import ee.cashier.store.service.FindStoreService;
import external.api.CreateStoreRequestJson;
import external.api.StoreResponseJson;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import static ee.cashier.store.auth.AuthenticationContextHolder.get;
import static java.util.Optional.ofNullable;
@RestController
public class StoreApi implements external.api.StoreApi {
    @Autowired
    private CreateStoreService createStore;
    @Autowired
    private FindStoreService findStore;
    @Override
    public StoreResponseJson create(@RequestBody CreateStoreRequestJson json)
{
        return createStore.create(get().getCustomerId(), json).toStoreRe-
sponseJson();
    }
    @Override
    public StoreResponseJson search(@RequestParam("barcode") String barcode)
{
        return ofNullable(findStore.findByBarcode(barcode)).map(Store::to-
StoreResponseJson).orElse(null);
    }
    @Override
    public StoreResponseJson find(@PathVariable("storeId") Long storeId) {
        return ofNullable(findStore.find(storeId)).map(Store::toStoreRespon-
seJson).orElse(null);
    }
}
```

```
package ee.cashier.store.auth;
import external.api.UsernameResponseJson;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class AuthenticationContextHolder {
    private static final Logger log = LoggerFactory.getLogger(Authentication-
ContextHolder.class);
    private static final ThreadLocal<UsernameResponseJson> authentications =
new ThreadLocal<>();
    public static void put(UsernameResponseJson shardContext) {
```

```
          log.debug("Setting context");
          authentications.set(shardContext);
     }
     public static UsernameResponseJson get() {
          log.debug("Getting");
          return authentications.get();
     }
     public static void clear() {
          log.debug("Clearing");
          authentications.remove();
     }
}
```

```
package ee.cashier.store.auth;
import org.springframework.cloud.netflix.feign.FeignClient;
@FeignClient("auth-service")
public interface LoginApi extends external.api.LoginApi {
}
```

```
package ee.cashier.store.auth;
import external.api.UsernameResponseJson;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import javax.servlet.*;
import javax.servlet.http.HttpServletRequest;
import java.io.IOException;
@Component
public class SecurityFilter implements Filter {
     @Autowired
     private LoginApi loginApi;
     @Override
     public void init(FilterConfig filterConfig) throws ServletException {
     }
     @Override
     public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain)
               throws IOException, ServletException {
          String authenticationToken = ((HttpServletRequest)
servletRequest).getHeader("x-auth");
          if (authenticationToken == null) {
               throw new RuntimeException("Authentication token is missing");
          }
          UsernameResponseJson tokenData = loginApi.findData(authenticationTo-
ken);
          if (tokenData == null) {
               throw new RuntimeException("Authentication token is not valid");
          }
          AuthenticationContextHolder.put(tokenData);
          filterChain.doFilter(servletRequest, servletResponse);
     }
     @Override
     public void destroy() {
     }
}
```

```
package ee.cashier.store.dao;
import ee.cashier.store.model.Employee;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Param;
```

```
@Mapper
public interface EmployeeDao {
     void save(Employee employee);
     Employee find(Long employeeId);
     Employee findByCustomer(@Param("storeId") Long storeId, @Param("cus-
tomerId") Long customerId);
}
```

```
package ee.cashier.store.dao;
import ee.cashier.store.model.Store;
import org.apache.ibatis.annotations.Mapper;
@Mapper
public interface StoreDao {
     void save(Store store);
     Store find(Long storeId);
     Store findByBarcode(String barcode);
}
```

```
package ee.cashier.store.model;
import external.api.StoreResponseJson;
import java.time.LocalDate;
public class Store {
     private Long id;
     private String name;
     private String barcode;
     private LocalDate registrationDate;
     public Long getId() {
          return id;
     }
     public void setId(Long id) {
          this.id = id;
     }
     public String getName() {
          return name;
     }
     public void setName(String name) {
          this.name = name;
     }
     public LocalDate getRegistrationDate() {
          return registrationDate;
     }
     public void setRegistrationDate(LocalDate registrationDate) {
          this.registrationDate = registrationDate;
     }
     public String getBarcode() {
          return barcode;
     }
     public void setBarcode(String barcode) {
          this.barcode = barcode;
     }
     public StoreResponseJson toStoreResponseJson() {
          StoreResponseJson json = new StoreResponseJson();
          json.setStoreId(id);
          json.setBarcode(barcode);
          json.setStoreName(name);
          return json;
     }
}
```

```java
package ee.cashier.store.publisher;
public class EmployeeRegisteredEvent {
     private Long employeeId;
     private Long storeId;
     private Long customerId;
     private String occupation;
     public Long getEmployeeId() {
          return employeeId;
     }
     public void setEmployeeId(Long employeeId) {
          this.employeeId = employeeId;
     }
     public Long getStoreId() {
          return storeId;
     }
     public void setStoreId(Long storeId) {
          this.storeId = storeId;
     }
     public Long getCustomerId() {
          return customerId;
     }
     public void setCustomerId(Long customerId) {
          this.customerId = customerId;
     }
     public String getOccupation() {
          return occupation;
     }
     public void setOccupation(String occupation) {
          this.occupation = occupation;
     }
}
```

```java
package ee.cashier.store.publisher;
import ee.cashier.store.model.Employee;
import ee.cashier.store.service.FindEmployeeService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
@Component
public class EmployeeRegisteredPublisher {
     private final String eventName = "employee-registered";
     @Autowired
     private EventPublisher eventPublisher;
     @Autowired
     private FindEmployeeService findEmployeeService;
     public void publish(Long employeeId) {
          Employee employee = findEmployeeService.find(employeeId);
          EmployeeRegisteredEvent event = new EmployeeRegisteredEvent();
          event.setEmployeeId(employee.getId());
          event.setStoreId(employee.getStoreId());
          event.setCustomerId(employee.getCustomerId());
          event.setOccupation(employee.getOccupation().name());
          eventPublisher.publish(eventName, event);
     }
}
```

```java
package ee.cashier.store.model;
import ee.cashier.lang.daterange.DateRange;
public class Employee {
     private Long id;
```

```
    private Long storeId;
    private Long customerId;
    private EmployeeOccupation occupation;
    private DateRange validityRange;
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public Long getStoreId() {
        return storeId;
    }
    public void setStoreId(Long storeId) {
        this.storeId = storeId;
    }
    public EmployeeOccupation getOccupation() {
        return occupation;
    }
    public void setOccupation(EmployeeOccupation occupation) {
        this.occupation = occupation;
    }
    public DateRange getValidityRange() {
        return validityRange;
    }
    public void setValidityRange(DateRange validityRange) {
        this.validityRange = validityRange;
    }
    public Long getCustomerId() {
        return customerId;
    }
    public void setCustomerId(Long customerId) {
        this.customerId = customerId;
    }
}
```

```
package ee.cashier.store.publisher;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Component;
@Component
public class EventPublisher {
    @Autowired
    private KafkaTemplate<Object, Object> kafkaTemplate;
    @Autowired
    private ObjectMapper objectMapper;
    protected void publish(String topic, Object event) {
        kafkaTemplate.send(topic, convertToJson(event)).addCallback(
                o -> System.out.println("Success " + o),
                throwable -> {
                    System.err.println(throwable.getMessage());
                    throwable.printStackTrace();
                });
    }
    private String convertToJson(Object event) {
        try {
            return objectMapper.writeValueAsString(event);
```

```
        } catch (JsonProcessingException e) {
            throw new RuntimeException(e);
        }
    }
}
```

```
package ee.cashier.store.publisher;
import java.time.LocalDate;
public class StoreRegisteredEvent {
    private Long storeId;
    private String storeName;
    private String storeBarcode;
    private LocalDate registrationDate;
    public Long getStoreId() {
        return storeId;
    }
    public void setStoreId(Long storeId) {
        this.storeId = storeId;
    }
    public String getStoreName() {
        return storeName;
    }
    public void setStoreName(String storeName) {
        this.storeName = storeName;
    }
    public String getStoreBarcode() {
        return storeBarcode;
    }
    public void setStoreBarcode(String storeBarcode) {
        this.storeBarcode = storeBarcode;
    }
    public LocalDate getRegistrationDate() {
        return registrationDate;
    }
    public void setRegistrationDate(LocalDate registrationDate) {
        this.registrationDate = registrationDate;
    }
}
```

```
package ee.cashier.store.publisher;
import ee.cashier.store.model.Store;
import ee.cashier.store.service.FindStoreService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
@Component
public class StoreRegisteredPublisher {
    private final String eventName = "store-registered";
    @Autowired
    private EventPublisher eventPublisher;
    @Autowired
    private FindStoreService findStoreService;
    public void publish(Long storeId) {
        Store store = findStoreService.find(storeId);
        StoreRegisteredEvent event = new StoreRegisteredEvent();
        event.setStoreId(store.getId());
        event.setStoreName(store.getName());
        event.setStoreBarcode(store.getBarcode());
        event.setRegistrationDate(store.getRegistrationDate());
        eventPublisher.publish(eventName, event);
```

```
    }
}
```

```
package ee.cashier.store.service;
import ee.cashier.store.dao.EmployeeDao;
import ee.cashier.store.model.Employee;
import ee.cashier.store.model.EmployeeOccupation;
import ee.cashier.store.publisher.EmployeeRegisteredPublisher;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import static ee.cashier.business.BusinessAssert.isFalse;
import static ee.cashier.lang.daterange.DateRange.dateRange;
import static java.time.LocalDate.now;
@Component
public class CreateEmployeeService {
    private static final Logger log = LoggerFactory.getLogger(CreateEmploy-
eeService.class);
    @Autowired
    private EmployeeDao employeeDao;
    @Autowired
    private FindEmployeeService findEmployeeService;
    @Autowired
    private EmployeeRegisteredPublisher employeeRegisteredPublisher;
    public Long create(Long storeId, Long customerId, EmployeeOccupation oc-
cupation) {
        log.debug("Creating employer for store-{} with customer-{} and occu-
pation", storeId, customerId, occupation);
        isFalse(findEmployeeService.exists(storeId, customerId), "ee.s-
tore.employeeAlreadyExists");
        Employee employee = new Employee();
        employee.setStoreId(storeId);
        employee.setValidityRange(dateRange(now()));
        employee.setOccupation(occupation);
        employee.setCustomerId(customerId);
        employeeDao.save(employee);
        employeeRegisteredPublisher.publish(employee.getId());
        return employee.getId();
    }
}
```

```
package ee.cashier.store.service;
import ee.cashier.store.dao.StoreDao;
import ee.cashier.store.model.Store;
import ee.cashier.store.publisher.StoreRegisteredPublisher;
import external.api.CreateStoreRequestJson;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;
import static ee.cashier.business.BusinessAssert.isNotEmpty;
import static ee.cashier.store.model.EmployeeOccupation.OWNER;
import static java.time.LocalDate.now;
import static java.util.UUID.randomUUID;
@Component
public class CreateStoreService {
    private static final Logger log = LoggerFactory.getLogger(CreateStoreSer-
vice.class);
    @Autowired
```

```
    private StoreDao storeDao;
    @Autowired
    private CreateEmployeeService createEmployee;
    @Autowired
    private StoreRegisteredPublisher storeRegisteredPublisher;
    @Transactional
    public Store create(Long customerId, CreateStoreRequestJson json) {
        isNotEmpty(json.getName(), "err.store.nameIsMissing");
        Store store = new Store();
        store.setName(json.getName());
        store.setBarcode(randomUUID().toString());
        store.setRegistrationDate(now());
        storeDao.save(store);
        createEmployee.create(store.getId(), customerId, OWNER);
        storeRegisteredPublisher.publish(store.getId());
        return store;
    }
}
```

```
package ee.cashier.store.service;
import ee.cashier.store.dao.EmployeeDao;
import ee.cashier.store.model.Employee;
import ee.cashier.store.model.Store;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
@Component
public class FindEmployeeService {
    private static final Logger log = LoggerFactory.getLogger(FindEmploy-
eeService.class);
    @Autowired
    private EmployeeDao employeeDao;
    public Employee find(Long storeId, Long customerId) {
        return employeeDao.findByCustomer(storeId, customerId);
    }
    public boolean exists(Long storeId, Long customerId) {
        return employeeDao.findByCustomer(storeId, customerId) != null;
    }
    public Employee find(Long employeeId) {
        return employeeDao.find(employeeId);
    }
}
```

```
package ee.cashier.store.service;
import ee.cashier.store.dao.StoreDao;
import ee.cashier.store.model.Store;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
@Component
public class FindStoreService {
    private static final Logger log = LoggerFactory.getLogger(FindStoreSer-
vice.class);
    @Autowired
    private StoreDao storeDao;
    public Store findByBarcode(String barcode) {
        return storeDao.findByBarcode(barcode);
    }
```

```
    public Store find(Long storeId) {
        return storeDao.find(storeId);
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <settings>
        <setting name="cacheEnabled" value="false"/>
        <setting name="localCacheScope" value="STATEMENT"/>
    </settings>
    <typeAliases>
        <package name="ee.cashier.store.model"/>
        <package name="ee.cashier.domain"/>
    </typeAliases>
    <typeHandlers>
        <typeHandler handler="ee.cashier.lang.daterange.DateRangeTypeHan-
dler"/>
    </typeHandlers>
    <mappers>
        <mapper resource="dao/EmployeeDao.xml"/>
        <mapper resource="dao/PaypointDao.xml"/>
        <mapper resource="dao/PaypointManagerDao.xml"/>
        <mapper resource="dao/StoreCustomerDao.xml"/>
        <mapper resource="dao/StoreDao.xml"/>
    </mappers>
</configuration>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://my-
batis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="ee.cashier.store.dao.EmployeeDao">
    <resultMap id="EmployeeResultMap" type="Employee">
        <result column="id" property="id"/>
        <result column="store_id" property="storeId"/>
        <result column="occupation" property="occupation"/>
        <result column="validity_range" property="validityRange"/>
    </resultMap>
    <insert id="save" useGeneratedKeys="true" keyProperty="id" keyCol-
umn="id">
        INSERT INTO store.employee (
        store_id,
        customer_id,
        validity_range,
        occupation
        ) VALUES (
        #{storeId},
        #{customerId},
        #{validityRange}::daterange,
        #{occupation}
        )
    </insert>
    <select id="find" resultMap="EmployeeResultMap">
        SELECT *
        FROM store.employee
        WHERE id = #{find}
```

```
    </select>
    <select id="findByCustomer" resultMap="EmployeeResultMap">
        SELECT *
        FROM store.employee
        WHERE store_id = #{storeId}
               AND customer_id = #{customerId}
    </select>
</mapper>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://my-
batis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="ee.cashier.store.dao.StoreDao">
    <resultMap id="StoreResultMap" type="Store">
        <result column="id" property="id"/>
        <result column="name" property="name"/>
        <result column="barcode" property="barcode"/>
        <result column="registration_date" property="registrationDate"/>
    </resultMap>
    <insert id="save" useGeneratedKeys="true" keyColumn="id" keyProper-
ty="id">
        INSERT INTO store.store (
            name,
            barcode,
            registration_date
        ) VALUES (
            #{name},
            #{barcode},
            #{registrationDate}
        )
    </insert>
    <select id="find" resultMap="StoreResultMap">
        SELECT *
        FROM store.store
        WHERE id = #{storeId}
    </select>
    <select id="findByBarcode" resultMap="StoreResultMap">
        SELECT *
        FROM store.store
        WHERE barcode = #{barcode}
    </select>
</mapper>
```

Lisa 8 – Administraatori rakenduse programmikood

```
dependencies {
     compile "de.codecentric:spring-boot-admin-server:1.4.2"
     compile "de.codecentric:spring-boot-admin-server-ui:1.4.2"
     compile('org.springframework.cloud:spring-cloud-starter-eureka')
     compile "org.springframework.cloud:spring-cloud-config-server"
}
```

```
package ee.cashier.admin;
import de.codecentric.boot.admin.config.EnableAdminServer;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.config.server.EnableConfigServer;
@EnableAutoConfiguration
@EnableAdminServer
@EnableDiscoveryClient
@EnableConfigServer
public class AdminApplication {
     public static void main(String[] args) {
          SpringApplication.run(AdminApplication.class, args);
     }
}
```

```
server.port: 9999
spring.application.name: admin-service
spring.profiles.active: native
management.security.enabled: false
eureka.client:
  registryFetchIntervalSeconds: 5
  registerWithEureka: false
  fetchRegistry: true
  serviceUrl.defaultZone: http://localhost:8761/eureka/
spring.boot.admin:
  contextPath: /admin
spring.cloud.config.server:
  prefix: /config
  native.search-locations: classpath:config/
```

Lisa 9 – Teenuste registri rakenduse programmikood

```
dependencies {
     compile "de.codecentric:spring-boot-admin-server:1.4.2"
     compile "de.codecentric:spring-boot-admin-server-ui:1.4.2"
     compile('org.springframework.cloud:spring-cloud-starter-eureka-server')
     compile('org.springframework.cloud:spring-cloud-starter-eureka')
     compile "org.springframework.cloud:spring-cloud-config-server"
}
```

```
server.port: 8761
spring.application.name: discovery-service
management.security.enabled: false
eureka.client:
  registerWithEureka: false
  fetchRegistry: true
eureka.server:
  waitTimeInMsWhenSyncEmpty: 0
```

```
package ee.cashier.discovery;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;
@EnableEurekaServer
@SpringBootApplication
public class DiscoveryApplication {
     public static void main(String[] args) {
          SpringApplication.run(DiscoveryApplication.class, args);
     }
}
```