TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Kristo Truu 163441

MOBIILSE ISETEENINDUSPLATVORMI LOOMINE

Magistritöö

Juhendaja: Enn Õunapuu
PhD

Tallinn 2018

Autorideklaratsioon

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

Autor: Kristo Truu

30.04.2018

Annotatsioon

Käesoleva töö eesmärgiks on välja töötada mobiilipõhine iseteenindusplatvorm, mis muudaks lõppkasutaja jaoks iseteeninduse kasutamise mugavamaks ning kliendi jaoks kasutamise kättesaadavamaks, odavamaks ning tehnoloogiliselt lihtsamaks.

Iseteeninduse süsteemid on muutunud viimaste aastate jooksul inimeste igapäevaste toimingute osaks. Sageli nõuavad süsteemi pakkuvad asutused kliendikonto olemasolu. Samuti on iseteenindussüsteemide kvaliteet väga erinev. Antud töös pakutakse välja lahendus, millega proovitakse iseteeninduse platvormiga muuta pakutavad iseteenindused standartsemaks kasutades selleks lõppkasutaja mobiiltelefoni.

Antud töö esimeses pooles keskendutakse iseteeninduse süsteemide uurimisele, platvormi arendamisel vajaminevate komponentide, raamistikke ning arhitektuuri võrdlusele, analüüsimisele ning valimisele. Töö teises pooles kirjeldatakse ja testitakse loodud platvormi ning võrreldakse seda olemasoleva iseteenindussüsteemiga.

Magistritöö tulemusena valmib testitud mobiilipõhine iseteeninduse platvorm ning selle võrdlus olemasoleva iseteenindussüsteemiga.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 43 leheküljel, 6 peatükki, 32 joonist, 14 tabelit.

Abstract

Mobile based self-service platform

The goal of this master's thesis is to develop a mobile self-service platform that would make self-service more convenient for end-users and more usable, cheaper and technologically easier to use for customers.

Self-service systems have become part of people's day-to-day operations in recent years. Often, self-services offered by institutions require a client account creation. Also, the quality of self-service systems varies greatly. In this thesis, we propose a solution that attempts to make self-service offered by a self-service platform more standardized by using an end-user mobile phone.

In the first half of this work, the focus is on researching self-service systems, comparing, analyzing and selecting components, frameworks and architectures needed to develop a plaform. In the second half of the work, the plaform is described, tested and compared to the existing self-service system.

As a result of this Master's thesis, the tested mobile-based self-service platform will be created and its comparsion with the existing self-service systems will be completed.

The thesis is in Estonian and contains 43 pages of text, 6 chapters, 32 figures, 14 tables.

Lühendite ja mõistete sõnastik

SOA	*Service-Oriented Architecture,* Teenus orienteeritud arhitektuur
ESB	*Enterprise service bus,* ettevõtte suhtlussiin
PEER-TO-PEER	Arvutite vaheline detsentraliseeritud võrk
CPU	*Central processing unit*, protsessor
EVM	*Ethereum Virtual Machine*, Ethereumi virtuaalmasin
GAS	Ethereumi protokollis kaevandajatele makstav tasu
RPC	Remote procedure call, Kaugjuhitav protseduur
M2M	*Machine to Machine,* Masin-masin suhtlusvõrk
AMQP	*Advanced Message Queuing Protocol*, sõnumijärjestus protokoll
IIOP	*Internet Inter-ORB Protocol*, interneti põhine suhtlusprotokoll
RMI	*Remote Method Invocation*, Kaugmeetodi kutsuja
SOAP	*Simple Object Access Protocol,* lihnte objektipöördusprotokoll
DRAM	*Dynamic random access memory*, dünaamiline muutmälu
API	*Application programming interface*, rakendusliides
NoSQL	*No Structured Query Language,* struktuurita päringukeel
JSON	*JavaScript Object Notation,* Javascriptil põhinev andmevahetusvorming
AtomPub	Andmevahetusvorming
XML	*Extensible Markup Language*, laiendatav märgistuskeel
POJO	Plain Old Java Object, harilik Java object
AJAX	*Asynchronous Javascript and XML,* Asünkroonne Javascript ja XML
Native	Platvormipõhine funktsionaalsus
MVC	*Model-View-Controller*, Mudel-vaade-kontroller arhitektuur
HTML	*HyperText Markup Language*, hüperteksti märgistuskeel
CSS	*Cascading Style Sheets,* astmelised stiililehed
UI	*User interface,* kasutajaliides
JIT	*Just In Time,* samal hetkel

AOT	*Ahead Of Time*, ennetähtaegne
SSD	*Solid State Drive*, pooljuhtketas
HDD	*Hard Disk Drive,* kõvaketas
Hypervisor	Virtualiseerimise kiht
MVP	*Minimal valuable product*, minimaalne väärtusega toode

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Antud magistritöö eesmärk on luua mobiilipõhine iseteenindusplatvorm, mis muudaks iseteeninduse pakkumise odavamaks ja lihtsamaks ning iseteeninduse kasutamise mugavamaks. Sellest tulenevalt uuritakse ning analüüsitakse erinevaid vajaminevaid komponente, arhitektuure ja algoritme antud ülesande teostamiseks, tehakse kindlaks nõuded ning valitakse eelnevalt läbiviidud analüüsi tulemustest parimad lahendused nõuete täitmiseks.

1.1 Taust ja probleem

Iseteeninduse süsteemid on muutunud viimaste aastate jooksul inimeste igapäevaste toimingute osaks. Paljud kauplused toetavad iseteenindust kasutades selleks iseteeninduse terminale või pulte. Laialdast levikut pole leidnud mobiiltelefonil põhinevad iseteenindused. Samuti on kaupluste iseteenindused suhteliselt erinevad ning sageli nõuavad kasutamiseks poekonto olemasolu. Lisaks nõuavad iseteenindus süsteemid kaupluselt tehnilise oskuse olemasolu ning vajalikke seadmeid mille abil iseteenindust pakkuda.

1.2 Ülesande püstitus

Töö peamiseks ülesandeks on luua mobiilipõhine iseteeeninduse platvorm, millega proovitakse iseteeninduse pakkumist parendada. Antud ülesanne hõlmab endas sobiva arhitektuuri valimist, sobivate platvormide valimist, süsteemsete komponentide valimist, süsteemi testimist reaalses olukorras ning lõpphinnangut.

Eelnevalt mainitud ülesandest tuleks lähtuda põhimõttega, et kaupluse kulud oleks süsteemi kasutades võimalikult minimaalsed, kauplus peab omama võimalikult vähe spetsiaalselt riistvara, platvorm oleks kergesti skaleeruv, liidestamine peaks olema valutu ning ostlemine peaks olema võimalikult lihtne.

1.3 Metoodika

Mobiilse iseteeninduse platvormi arendamisel uuritakse erinevaid iseteeninduse osutamiseks pakutavaid meetodeid, tehakse kindlaks süsteemi nõuded ning uuritakse sobivaid raamistikke, arhitektuure ning vajaminevaid komponente teoreetiliselt kui ka osaliselt praktiliselt.

Eelneva uuringu põhjal valitakse välja uuritud alternatiividest parimad võimalikud lahendused ning rakendatakse neid platvormi arendamise käigus.

Arendamisel lähtutakse agiilse tarkvaraarenduse metoodikatest.

Prototüübi testimiseks valitakse suvaline iseteenindust pakkuv kauplus kus on võimalik kasutada iseteenindus pulte ning hinnatakse süsteemi tähtsamaid omadusi. Tähtsamad omadused selguvad lõputöö käigus.

Lõputöö väljund on eelneva analüüsi põhjal loodud mobiilse iseteeninduse platvormi prototüüp ning selle võrdlus mõne olemasoleva lahendusega.

1.4 Ülevaade tööst

Lõputöö on jagatud viieks osaks.

Lõputöö esimeses osas keskendutakse iseteenindus tehnoloogiate uurimisele.

Teises osas tehakse kindlaks platvormi nõuded.

Kolmas osa lõputööst keskendub raamistike, arhitektuuri ja komponentide leidmisele, võrdlemisele ning analüüsile.

Lõputöö neljanda osa eesmärk luua platvorm kasutades selleks eelmises peatükis analüüsitud komponente ning kirjeldada platvormi tehnilist osa.

Lõputöö viiendas osas analüüsitakse ja testitakse valminud rakendust.

2 Iseteeninduse süsteemide taust

Iseteeninduste eesmärgiks on vähendada inimeste ressurssi teenuste pakkumisel ning anda osaline vastutus kliendile teenuse toimimiseks. Teiseks suuremaks iseteeninduse rakendamise põhjuseks on kulude kokkuhoid. Lisaks on iseteeninduse rakendamise eesmärk muuta teenus atraktiivsemaks ning innovaatilisemaks kliendi jaoks.

Kõige levinumad iseteenindus meetodid on terminalipõhine iseteenindus, veebipõhine iseteenindus ning mobiilipõhine iseteenindus

Terminalipõhine iseteeninduse pakkumine toimub läbi füüsilise seadme, mis on paigaldatud fikseeritud asukohta. Laialdaselt kasutatakse terminali põhist iseteenindust automaattanklates pangasüsteemides, parklates ja kauplustes. Veebipõhine iseteenindus on laialdaselt levinud viimaste aastate jooksul. Iseteenindus toimub läbi kliendi veebibrauseri. Mobiilipõhist iseteenindust osutatakse mobiiltelefoni kaudu.

2005. aastal viidi pangasüsteemide näitel läbi uuring, kus hinnati kolme eelnevalt kirjeldatud iseteeninduse strateegiat. Uuringus osales 654 inimest ning põhiliselt hinnati uuringus kasutamise lihtsust, kasulikkust, suhtlemise vajalikkust ning riski. Uuringu tulemusest selgub funamentaalne erinevus kõigi kolme iseteeninduse vahel. Kasutajad pidasid kõige kasulikumaks terminali ning telefoni põhist iseteenindust. Terminali kasutamist pidasid aga uuringus osalenud inimesed kõige lihtsamaks, mis on selle uuringu võtmesõnaks. Veebipõhist pangandust peeti kõige suuremaks riskiks. Kuna 2005. aastal oli veebipõhine iseteenindus veel suuresti arengujärgus, siis ei osanud uuringus osalejad väga kindlat seisukohta veel selle kohta võtta [1].

3 Platvormi nõuded

Antud peatükis tuukase välja mobiilse iseteeninduse platvormi nõuded. Kuna nõudeid on palju ning lõputöö raames ei jõua kõiki nõudeid käsitleda, siis kirjeldatakse ainult kõige tähtsamaid nõudeid.

Nõuete kirjeldamiseks kasutatavad tegutsejad:

- Lõppkasutaja – Asutuse klient, kes kasutab iseteeninduse platvormi ostude tegemiseks
- Asutus – Asutus või organisatsioon, kes soovib oma teenuse pakkumiseks kasutada antud iseteeninduse platvormi

3.1 Funktsionaalsed nõuded

3.1.1 Skaneerimine

Skaneerimine peab toimuma mobiiltelefoni abil. Skanner peab suutma tuvastada kõiki ühe dimensoonilisi triipkoode välja arvatud GS1 DATABAR tüüpi triipkoodi. Kahe dimensiooniliste triipkoodide tüüpidest peab skanner suutma tuvastada ainult QR koodi. Skanner peab töötama iOS ning Android platvormidel. Skanner peab töötab minimaalselt Android 5.0 versiooni peal ning iOS 7.0 versiooni peal.

Tabel 1. SMTF-1 nõue

ID	SMTF-1
Nõue	Skaneerides QR-koodi, mis sisaldab asutuse id-d peab olema võimalik Androidi mobiiltelefoni abil siseneda ostlemise keskkonda.
Tegutsejad	Lõppkasutaja
Eeltingimus	Autoriseeritud lõppkasutaja
Järeltingimus	Kauplusesse sisenenud autoriseeritud lõppkasutaja

Põhistsenaarium	Lõppkasutaja autoriseerib ennast mobiiltelefoni abil. Pärast autoriseerimist tuleb lõppkasutajal skaneerida asutuse põhine QR kood. Eduka skaneerimise tulemusel saab kasutaja krüpteeritud kujul poe andmed ja arve ning need salvestatakse lokaalselt lõppkasutaja mobiiltelefoni. Ebaõnnestunud skaneerimise puhul antakse lõppkasutajale tagasiside probleemist ning palutakse uuesti skaneerida QR-kood.

Tabel 2.SMTF-2 nõue

ID	SMTF-2
Nõue	Skaneerides toote triipkoodi või QR koodi peab olema võimalik Androidi telefoni abil lisada toode ostukorvi.
Tegutsejad	Lõppkasutaja
Eeltingimus	Autoriseeritud lõppkasutaja, asutusse sisenenud kasutaja
Järeltingimus	Uuendatud krüpteeritud arve ning uus toode on lisatud toote listi.
Põhistsenaarium	Lõppkasutaja skaneerib toote. Toote leidmise korral lisatakse uus toode arvele ning tagastakse see kasutajale. Kasutajale kuvatakse viimase toote detailandmed. Ebaõnnestunud skaneerimisest antakse kliendile teada hüpikdialoogi abil.

Tabel 3. SMTF-3 nõue

ID	SMTF-3
Nõue	Lõppkasutajal peab olema võimalik ostu eest tasuda skaneerides asutuse QR koodi kasutades selle Android mobiiltelefoni.
Tegutsejad	Lõppkasutaja

Eeltingimus	Autoriseeritud lõppkasutaja, asutusse sisenenud kasutaja, maksevaluuta olemasolu, skaneeritud toodete olemasolu
Järeltingimus	Alustatud transaktsioon, asutustest välja logitud lõppkasutaja
Põhistsenaarium	Lõppkasutaja skaneerib asutuse id-d sisaldava QR-koodi. Sellega algatatakse transaktsioon mille tulemusel võib klient asutusest lahkuda.

3.1.2 Maksevõimalused

Tabel 4. SMTF-4 nõue

ID	SMTF-4
Nõue	Lõppkasutaja peab saama tootede eest maksta vähemalt ühes krüptovaluutas
Tegutsejad	Lõppkasutaja
Eeltingimus	Süteemi kasutaja olemasolu
Järeltingimus	Krüptovaluutas makstud arve
Põhistsenaarium	Lõppkasutaja maksab ostukorvi eest kasutades ühte populaarsematest krüptovaluutadest.

3.1.3 Kasutaja tegevused

Tabel 5.SMTF-5 nõue

ID	SMTF-5
Nõue	Lõppkasutajal peab olema võimalus registreerida ennast süsteemi kasutajaks kasutades selleks Androidi mobiiltelefoni.
Tegutsejad	Lõppkasutaja
Eeltingimus	Iseteeninduse süsteemi rakenduse olemasolu

Järeltingimus	Registreeritud kasutaja, kasutaja põhine krüptorahakoti olemasolu
Põhistsenaarium	Lõppkasutaja peab sisestama emaili, eesnime, perenime, kasutajanime, parooli ning parooli kinnituse. Nende andmete põhjal luuakse kasutajale süsteemikasutaja ning genereeritakse krüptorahakott.

Tabel 6. SMTF-6 nõue

ID	SMTF-6
Nõue	Lõppkasutaja peab saama süsteemi sisse logida registreeritud kasutajaga.
Tegutsejad	Lõppkasutaja
Eeltingimus	Registreeritud kasutaja olemasolu
Järeltingimus	Sisse loginud kasutaja
Põhistsenaarium	Lõppkasutaja sisestab kasutajanime ning parooli. Edukal sisenemisel suunatakse klient registreeritud kasutaja vaatesse.

3.1.4 Asutuse tegevused

Tabel 7. SMTF-7 nõue

ID	SMTF-7
Nõue	Asutus peab saama tellida endale aasta ajalise süsteemi kasutusõiguse.
Tegutsejad	Asutus
Eeltingimus	Iseteenindussüsteemi konto olemasolu
Järeltingimus	Asutus omab aastaajalist litsentsi

Põhistsenaarium	Asutus siseneb admin portaali, kus asutus saab lisada tellimuse. Tellimuse sisestamiseks peab asutus sisestama eeldatava lõppkasutajate arvu päevas, kes tarbib iseteeninduse teenust ning valima kas tellimust uuendatakse automaatselt tellimuse kehtivuse lõppemisel.

Tabel 8. SMTF-8 nõue

ID	SMTF-8
Nõue	Asutus peab saama tellimusele lisada poode.
Tegutsejad	Asutus
Eeltingimus	Aktiivse tellimuse olemasolu, Asutuse konto olemasolu
Järeltingimus	Lisatud pood
Põhistsenaarium	Asutus siseneb admin portaali, kus saab lisada tellimusele poode. Ühele tellimusele peab olema võimalik lisada mitu poodi. Poe lisamiseks peab asutus läbima järgnevad etapid. • Tellimuse valimine • Poe detailandmete sisestamine • Maksekontode sisestamine • Turvameetodite sisestamine • Integreeritud teenuse konfiguratsiooni andmete sisestamine • Integratsiooni automaattestide kävitamine (lõputöös ei kajastata) • Kaupluse aktiveerimine

3.2 Mittefunktsionaalsed nõuded

Antud peatükis kirjeldatakse süsteemile püstitatud mittefunktsionaalseid nõudeid vaba teksti formaadis.

3.2.1 Turvalisus

Süsteem peab tagama lõppkasutaja andmete turvalise hoiustamise ning piirama kolmanda osapoole ligipääsu nendele. Asutuste sõlmede ja mobiilse iseteenindus platvormi vaheline andmeside peab olema kaitstud ligipääsukoodidega. Süsteemile ligipääsemiseks peab leiduma vastavad ligipääsupunktid. Ülejäänud süsteemi komponendid peavad olema sisevõrgus ja tulemüüriga kaitstud. Süsteemiga liitunud asutustel peab olema võimalik ligi pääseda tulevikus tekkida võivatele kindlatele ressurssidele (nt. statistika moodul, süsteemis hoitavatele toodeteregistrile) . Iga ressursi poole pöördudes peab olema kasutaja autoriseeritud v.a sisselogimisel autoriseerimise teenuses. Igal ressursil peab olema võimalik tuvastada kasutaja õiguseid.

3.2.2 Skaleeritavus

Platvorm peab olema jaotatud mitmeks osaks, et oleks võimalik igat põhilist platvormi komponenti eraldi dubleerida või suurendada selle ressurssi. Platvormi põhilisteks komponentideks loetakse makse -ja arvemoodul, välissüsteemide suhtlusmoodul, tellimusmoodul, autoriseerimise -ja autentimise moodul, konfiguratsiooni moodul, integratsiooni testimise moodul.

3.2.3 Hallatavus

Platvormi peab olema võimalikult lihtne monitoorida ja hallata. Monitoorida peab saama üldist andmeliiklust kui ka eraldi erinevate moodulite osasid ning süsteemi masinaid. Versiooni uuendamise protsess peab olema automatiseeritud.

3.2.4 Stabiilsus

Platvormi stabiilsuse suurendamiseks peab olema võimalik mooduleid dubleerida ning paigutada koormuse tasakaalustaja nende ette. Suvalise mooduli rike ei tohi mõjutada terve süsteemi toimimist.

3.2.5 Kiirus

Toote skaneerimine peab olema võimalikult kiire, kuid mitte aeglasem kui 3 sekundit. Kauplusesse sisenemine peab olema võimalikult kiire, kuid mitte aeglasem kui 20 sekundit. Toodete eest maksmine peab olema võimalikult kiire, kuid mitte aeglasem kui 20 sekundit (krüptovaluuta puhul loetakse makseteostamiseks signeeritud transaktsiooni saatmist plokiahelasse mitte plokiahela transaktsiooni kinnitust) .

3.2.6 Keeled

Antud platvormi mobiiltelefoni rakendust peab saama kasutada vähemalt kahes keeles. Platvormi põhikeeleks peab olema inglise keel. Süsteem peab olema võimalik tuvastama kasutaja keele valiku automaatselt. Kui kasutaja keelevalikut ei ole võimalik tuvastada, siis on platvormi keeleks vaikimisi inglise keel.

4 Raamistike, arhitektuuri ja komponentide võrdlus

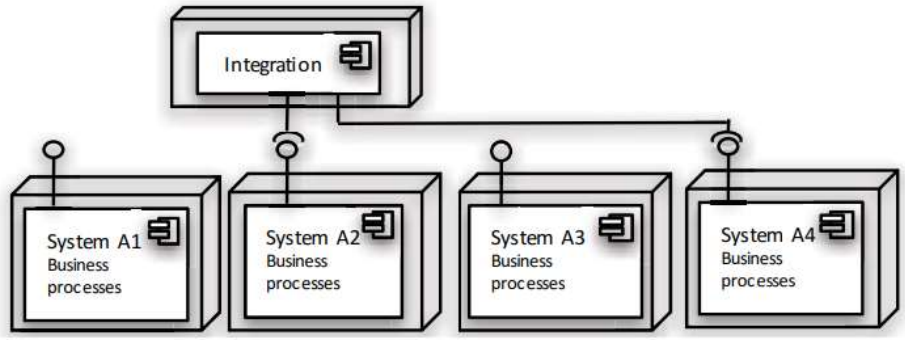
4.1 Arhitektuur

Antud alapeatükis võrreldakse kahte erinevat arhitektuuri mustrit. Võrdlusest jäetakse välja monoliitne arhitektuuri võrdlus, kuna antud ülesande jaoks pole see suure tõenäosusega kõlblik.

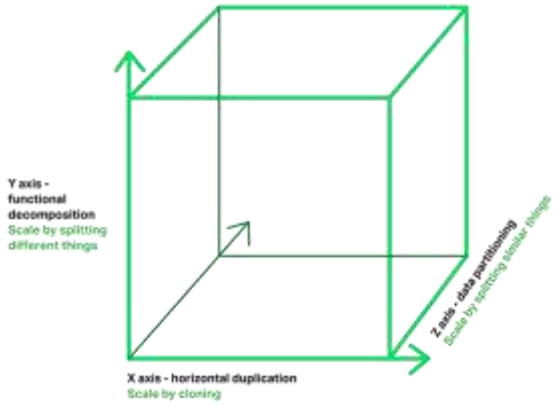
4.1.1 Mikroteenused

Kasutades mikroteenuste arhitektuuri vaadeldakse igat põhilist funktsionaalsust kui ühte väikest rakendust. Selline rakendus töötab iseseisvalt ning neid teenuseid saab täiesti iseseisvalt kasutusele võtta, uuendada ning vahetada. Rakenduste haldamine ei ole osa rakendusest vaid täiesti iseseisev rakendus. Samuti võivad olla need rakendused väga erinevad või isegi kirjutatud erinevates keeltes. Mikroteenuste negatiivseks osaks on keerukuse suurenemine. Samuti on mikroteenuste testimine keerukam. Mikroteenuse arendamine nõuab arendustiimidelt suuremat koordineermise võimet. Mikroteenuseid soovitatakse kasutada pigem projektipõhiste ülesannete lahendamiseks. Lisaks kasvatavad mikroteenused nõutavat mälumahtu ning teenuste õige jaotamise mehanism võib osutuda keerukaks. Mikroteenuste arhitekuuri põhisel teenuste jaotamisel on abiks

jaotus kuubik, mis aitab lihtsamini neid jaotada. X vektoril toimub rakenduste dubleerimine, Z teljel toimub rakenduste jaotus, kus iga komponent on dublikaat, kuid vastutab ainult kindla alamosa eest. Y teljel toiumub rakenduse jaotus erinevateks väiksemateks süsteemi komponentideks [2].



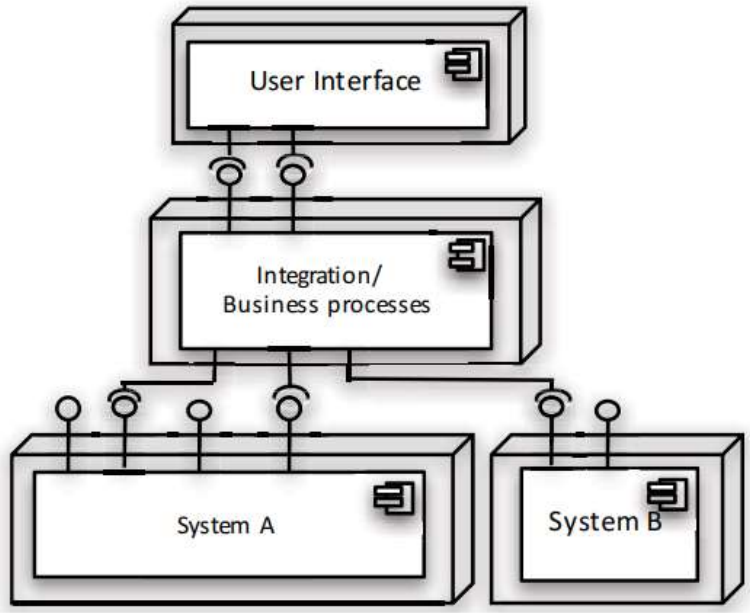
Joonis 1. Mikroteenuste arhitektuuri näide



Joonis 2 Mikroteenuste jaotuse kuubik

4.1.2 Teenus orienteeritud arhitektuur (SOA / ESB)

Teenus orieteeritud arhitektuuri eesmärk on jagada süsteem mitmeks erinevaks teenuseks. Kõik teenused omavad kindlat sorti ülesandeid mida on võimalik kohandada varjates rakendamise aluseks olevaid andmestruktuure. Teenused suhtlevad omavahel läbi integratsiooni komponendi (ESB). SOA negatiivseks osaks on see, et kindla teenuse muudatusel tuleb kogu teenus uuesti ülesse laadida. Tsentraliseeritud integratsiooni platvormid on tavaliselt keerulised ning võivad kasvada mahult suureks. Võrreldes mikroteenuste arhitektuuriga pole teenused ja äriloogika üksteisest täiesti eraldatud. Samuti on antud arhitektuuril andmebaasid tsentraalsed. SOA arhitektuuri haldamine on lihtsam, kuna see on tsentraliseeritud. Antud arhitektuur on sobilik suurtemate ettevõtete jaoks. Samuti on SOA negatiivseks osaks, et ta võib muutuda monoliitseks [3].



Joonis 3.SOA arhitektuuri näide

4.1.3 Hindamine

Algselt välistati monoliitne arhitektuur, kuna monoliitne arhitektuur ei vastanud süsteemi nõuetele või oleks selle rakendamine muutnud terve süsteemi jaoks väga keeruliseks ning andmemahtude kasvamisel kalliks.

Lähemalt uurides kahte arhitektuuri selgus, et SOA arhitektuuri on sobilik rakendada suurtes ettevõtetes ning mikroteenused pooldavad rohkem projektipõhist lähenemist.

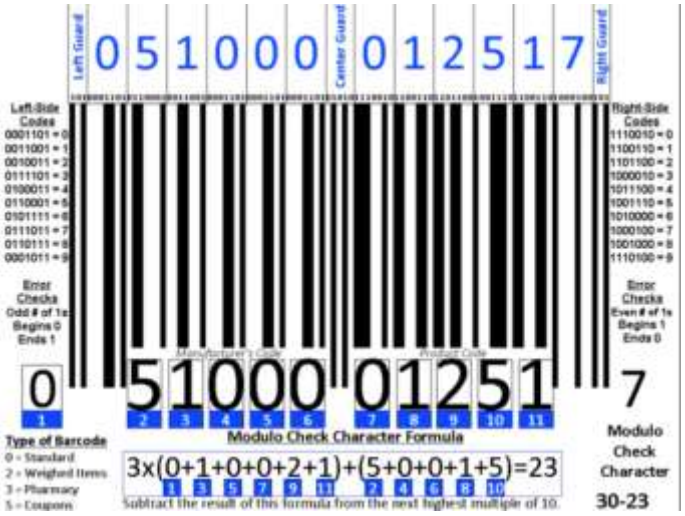
Antud süsteemi loomisel proovitakse lähtuda rohkem mikroteenuste põhimõtetest, kuna süsteemi osad on väga erinevad ning nõuavad erinevas mahus ressurssi. Samuti tuli välja arhitektuuri võrdlustest, et mikroteenuste skalleerimine (eriti horisontaalselt) on tunduvalt paindlikum kui SOA arhitektuuri puhul. Seega täidab mikroteenuste arhitektuur paremini süsteemi nõudeid.

4.2 Pildipõhine triipkoodi tuvastamine

Pildipõhine objektide tuvastus telefoni abil sai võimalikuks hetkel, kui seadmetele lisati protsessor, mille põhilisteks ülesanneteks on video koodeksite töötlemine, piltide koodeksite töötlemine ning muude samalaadsete ülesannetega tegelemine. Üheks uuritud algoritmiks triipkoodide tuvastamiseks on spiraalalgoritm ning QR koodide tuvastamiseks nurkalgoritm.

4.2.1 Triipkoodi skaneerimise taust

Ühe dimensiooniliseks triipkoodi tuvastamiseks jaotatakse triipkoodi ala 95-ks võrdseks sektsiooniks. Kuna arvuti suudab tuvastada ainult kahendkoodi, siis kodeeritakse kõik alad mis peegeldavad valgust 0-ks ning kõik alad mis ei peegelda valgust 1-ks. Pärast 95 sektsiooni skaneerimist jagatakse sektsioonid 15-ks grupiks. Nendest kaheteistkümne põhjal tehakse kindlaks triipkoodi number. Ülejäänud kolme gruppi kasutatakse valideerimiseks. Kolm valideerimise gruppi on vasak triipkoodi algus, triipkoodi keskkoht ning parem triipkoodi osa. Jaotades keskelt triipkoodi pooleks, tähistavad mõlemad triipkoodi osad erinevaid gruppe. Mõlema grupi puhul on võimalik tuvastada viga. Esimene pool triipkoodis peab algama kindlalt numbriga 0 ja lõppema numbriga 1. Teine pool triipkoodist peab kindlalt hakkama numbri 1 ning lõppema numbriga 0. Nende põhjal saab skanner aru mis pidi paikeb triipkood ja kust poolt tuleb alustada triipkoodi lugemist. Triipkoodi esimene number tähistab triipkoodi tüüpi. Järgmised 5 numbrit tähistavad tootja koodi. Alates 7-ndast numbrist loetakse viie kohaline tootekood. Viimaseks numbriks nimetatakse kontrollnumbrit.

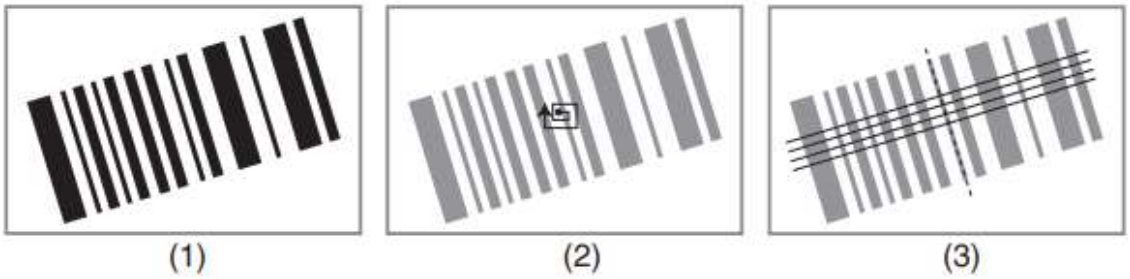


Joonis 4. Triipkoodi tuvastamine

4.2.2 Spiraal algoritm

Spiraal algoritmi ülesanne on tuvastada pildilt ühe dimensiooniline triipkood. Triipkoodi tuvastamine koosneb kolmest etapist. Esimeseks etapiks on eeltöötlemine. Selles etapis seatakse piirmäär valge ja musta intensiivsuse taseme vahel. Teiseks etapiks on rakendada spiraalse otsingu algoritmi, mis hakkab otsima pildi keskelt ja liigub otsinguga spiraalsel kujul pildi äärte suunas, et leida ülesse kõik must ala pildil. Kolmandas etapis arvutatakse

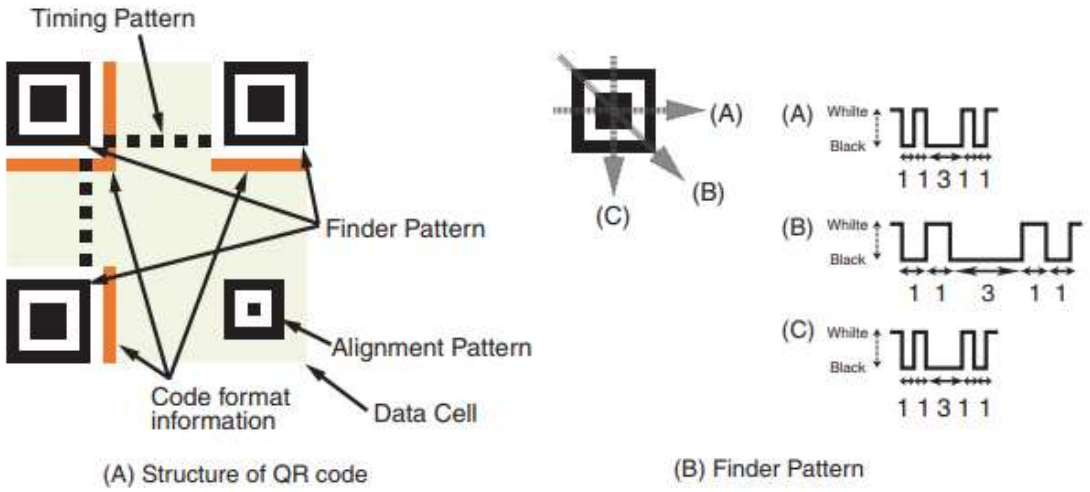
leitud mustade aladega risti olev joon ning selle põhjal on võimalik välja arvutada triipkoodi väärtus. [4]



Joonis 5 1D triipkood tuvastamine spiraal algoritmi abil

4.2.3 Nurk algoritm

Nurk algoritmi kasutatakse kahemõõtmelise koodi tuvastamiseks. Koodi tuvastamiseks kasutatakse nelja komponenti : otsingu muster, ajastuse muster, formaadi informatsioon, joondusmuster ning andmeelement. Algoritm koosneb viiest etapist. Esimeseks etapiks on eeltöötlemine. Eeltöötlemisel esimeseks etapiks on histogrammi arvutamine, et defineerida ära piirmäär musta ja valge ääreala vahel. Järgmisena kohandatakse pildi mõõtmeid ning tuvastakse ära QR koodi asukoht. Teises töötlemise etapis proovitakse leida ülesse QR koodi kolm nurka. Kolmandas etapis arvutatakse neljas nurk eelmisest etapist saadud andmete põhjal. Neljandaks etapiks on kahemõõtlemise koodi kuju normaliseermine. Viimases etapis loetakse kood kasutades alguses mainitud mustreid. [4]



Joonis 6. Nurkalgoritmiga kahemõõtmelise koodi leidmine

4.2.4 Lahenduste võrdlus ja testimine

Antud alapeatükis võrreldakse pildipõhise triipkoodi tuvastamise lahendusi ning hinnatakse põhilisi omadusi. Testimine viidi läbi kolme pakutava lahenduse peal. Testimiseks kasutati Samsung S5 mobiiltelefoni ning Samsung Galaxy A5 Tab-i.

Tabel 9. Triipkoodi tuvastamise teekide võrdlus

	Zxing	Scandit	Expo
1D triikoodi tuvastamise kiirus (keskmine aeg / sekundites)	< 1	< 1	< 1
2D triipkoodi tuvastamise kiirus (keskmine aeg /sekundites	< 1	< 1	< 1
1D triipkoodi tuvastamise kiirus pimedas ruumis	Ebaõnnestus	< 2	< 4
1D triikoodi tuvastamise kiirus varjudega (sekundites)	< 4	< 1	< 1
Edukalt skaneeritud koodid 5-st	5	5	5
Toetab androidi	Jah	Jah	Jah
Toetab ios	Ei	Jah	Jah
Toetab React native	Jah	Jah	Jah
Toetab Xamarin	Jah	Jah	Ei
Maksumus	Tasuta	Tasuline	Tasuta

Võrdluse tulemusel selgus, et Scanditi algoritm suudab kõige paremini tuvastada triipkoode pildil. Kuna Scanditi kasutamine on tasuline, siis alustati Scanditiga läbirääkimisi, mille tulemusel oli nõus Scandit andma 1 000 000 000 seadmele tasuta skaneerimise litsentsi. Seega valitakse triipkoodide ning QR koodide leidmiseks Scanditi poolt pakutavad pildipõhise koodide tuvastamise algoritmid.

4.3 Krüptovaluuta

Antud peatükis kirjeldatakse üldiselt krüptovaluuta tomimist ning võrreldakse kahte kõige populaarsemat krüptovaluutat.

4.3.1 Plokiahel

Plokiahelat võib vaadata kui detsentraliseeritud andmebaasi ning peer-to-peer võrku. Krüptograafilised algoritmid aitavad plokiahelat hoida valiidsena. Enamus plokiahela tehnoloogiaid on avatud lähtekoodiga. Plokiahela valiidsust ei kinnita ainult üks inimene vaid seda teeb kogu süsteem. [5]

Plokiahelasse kirjutamiseks on vaja saata andmed vahendajale, kellel on valiidne dublikaat olemas plokiahelast. Vahendaja kontrollib andmete korrektsust ning kirjutab need plokiahelasse. Kui plokk jõuab ahelasse, siis peavad kinnitama võrgus olevad vahendajad selle korrektsust. Kui kinnitused on saadud, siis on plokk kirjutatud ahelasse ning kõik võrgus olevad vahendajad omavad sellest dublikaati. [5]

Plokiahelad pole täiesti turvalised. Kui on olemas isik kes omab plokiahelast 51%, siis on sellel isikul võimalik reguleerida transaktsioone ning muuta transaktsiooni andmeid. Teiseks turvariskiks on plokiahela versioneerimine. Kui plokiahelast on olemas erinevad versioonid mis järgivad mingil määral erinevaid reegleid, siis ahel omavahel ei või töötada korrektselt ja võib tekitada turvaauke. Lisaks puudub plokiahelatel regulatsioon ning plokiahelad võivad minna mahult suureks. [5]

4.3.2 Bitcoin

Bitcoin on plokiahelal põhinev elektrooniline maksevahend. Selle eesmärgiks on vähendada transaktsiooni kulusid, suurendada transaktsiooni mahte, suurendada turvalisust ning vahetada välja usaldatavad isikud krüptograafiliste algoritmidega transaktsiooni kinnitamisel. Bitcoini algoritmi abil ei ole tarvis kolmandat osapoolt, kes tegeleks topelt kulutamise probleemiga. Bitcoin valideerib plokiahelat proof-of-work printsiibi alusel. Bitcoini transaktsioone on võimalik muuta, siis kui ründaja omab suuremat CPU võimsust kui usaldusväärsed võrgu sõlmed kokku. Bitcoini plokiahelas pole võimalik kirjutada tarkasid lepinguid nagu see on võimalik Ethereumi või mõne muu plohiahela protokollis. Samas pakub Bitcoin erinevaid meetoid mis võimaldaks vähesel määral jäljendada tarkasid lepinguid. Näiteks võimaldab Bitcoin lukustada fondi ajalise

piiranguga või nii kauaks kuni kindel fondi kulutaja on saatnud oma signatuuri ahelasse ning sellega vabastanud fondi. Bitcoini plokiahelaga ühendamiseks on võimalik kasutada Bitcoin Script javascript teeki. Samuti on võimalik ahelega suhelda läbi RPC [6].

4.3.3 Ethereum

Ethereumi protokoll põhineb plokiahelal. Ethereumi tuntakse kui detsentraliseeritud virtuaalmasinat (EVM) mis suudab käivitada koodi. Kood salvestatakse plokiahela peale ning seda nimetatakse lepinguks. Lepingus on võimalik kirjutada äriloogikat ning kanda üle Etherit samamoodi nagu Bitcoini protokolliga Satoshisid. Ethereumi ülekandmisel makstakse kaevandajatele tasu, mida nimetab Ethereum Gas-iks. Ülekandetasu sõltub üle kantavast andmemahust (mida rohkem, seda suurem GAS) ning kaevandajad võtavad ette esimesena transaktsioonid millel on kõige suurem GAS. Ethereumi plokiahela ühendamiseks on võimalik kasutada Web3 javascript teeki. Tarkasid lepinguid on võimalik kirjutada keeles nimega Solidity. Ethereumi arendamise jaoks on loodud ka arendaja tööriistu nagu Remix või Truffle. Truffle abil on võimalik kirjutada lepinguid, lepingutele automaatteste. Truffle on NodeJS raamistik mis on ühendatud Ethereumi RPC-ga [7].

4.3.4 Võrdlus

Analüüsist tuli välja, et antud protokollid on väga erinevad ning neid on raske omavahel kui tervikut võrrelda. Siiski on neil protokollidel ühine osa krüptovaluuta näol. Võrreldes Ethereumi Bitcoiniga pakub Ethereum paremat funktsionaalsust tarkade lepingute kirjutamiseks. Antud süsteemis võib olla tulevikus lepingute kirjutamine vajalik. Näiteks tahetakse salvestada arveid või luua süsteemi kasutamiseks oma krüptovaluuta. Seega otsustatakse, et Ethereum on sobilikum integreerida esimese krüptovaluutana süsteemi.

4.4 Sõnumivahendajad

Sõnumivahendajaid kasutatakse tavaliselt M2M suhtlusvõrgus, kus seadmed/teenused vahendavad omavahel kindlas formaadis defineeritud sõnumeid. Sõnumivahendajad kasutavad tavaliselt publish/subscribe arhitektuuri. Saatja seadmele ei ole oluline kuhu sõnum saadetakse ning saatja ei tea midagi sellest, kes sõnumit vastu võtab. See vähendab turvariski saatja ja saaja jaoks, kuid suurendab riski sõnumivahendajal.

Sõnumivahendajad omavad tavaliselt sõnumi ruuteri, -hoiustamise ning monitoorimise funktsionaalsust.

Antud töös uuritakse kahte kõige populaarsemat sõnumivahendajat ning võrreldakse sõnumivahendajate kõige tähtsamaid omadusi.

4.4.1 Apache Kafka

Kafka sõnumivahendaja ehitamist alustati vajadusele asendada detsentraliseeritud suhtlus tsentraliseeritud suhtluse vastu. Kafka kasutab publish/subscribe arhitektuuri. Antud sõnuvahendaja hoiustab andmeid kettal logi failides kui ka vahemälus nii palju kui võimalik. Kafka on ehitatud Apache Zookeeperi implementatsiooni põhiselt. Kafka klasterdab andmeid, mis võimaldab skaleerida lihtsamini mahu kasvamisel [6].

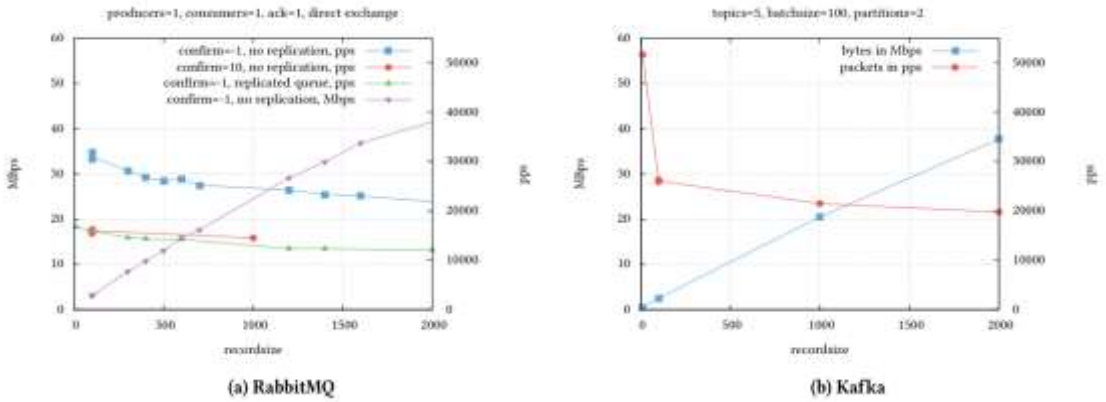
4.4.2 RabbitMQ

RabbitMQ põhineb AMQP protokollil. AMQP protoll sündis tänu asünkroonsete sõnumite saatmise vajadusele mida sünkroonsed (IIOP, RMI, SOAP jne) pakkuda ei suutnud. RabbitMQ tuntakse laialdaselt kui avatud lähtekoodiga skalleeritavat ning töökindlalt sõnumivahendajat. RabbitMQ kasutab Erleng OTP implementatsiooni [6].

4.4.3 Võrdlus ning tulemused

4.4.3.1 Aeg

RabbitMQ salvestab võimalikult palju andmeid DRAM-i. Kui maht otsa saab, siis hakkab ta salvestama kettale omamata andmete dublikaati DRAM-is. Kafka hoiab oma faile logifailides ning antud sõnumivahendaja disainitud nii, et mahu suurenemisel omab antud vahendaja eelist AMQP ees. Samuti hoiab Kafka asju DRAMIS-is [6].



(a) RabbitMQ (b) Kafka

Joonis 7. RabbitMQ ja Kafka reaktsiooniaeg

4.4.3.2 Marsruut

RabbitMQ pakub palju erinevaid marsruutimise loogika võimalusi. RabbitMQ võimaldab API kaudu luua ise sõnumivahenduse loogikat. Samuti võimaldab RabbitMQ hallata sõnumeid mida antud sõnumivahendaja töödelda ei suutnud. Kafka toetab põhilisi marsruutimise mehanisme mida laiendada pole lihtne. Seega RabbitMQ võimaldab paremat marsruutimist [6].

4.4.3.3 Tarne garantii

AMQP protollil põhinev sõnumivahendaja suudab tuvastada kadunud sõnumi ainult siis, kui sõnu saadetatakse transaktsiooni sees. Selline saatmine vähendab paraku sõnumivahendaja läbilaskevõimet. RabbitMQ täiendab AMQP protokolli, mis võimaldab saata sõnumeid ükshaaval ning seetõttu ei lähe terve sõnumivahetus kaduma. Samuti võimaldab RabbitMQ sõnumite järjekorda peegeldada. Kafkal võib tekkida andmekadu siis, kui võetakse sõnum vastu ning kettale salvestamine ebaõnnestub. Seetõttu ilma andmete replikeerimiseta võivad andmed kaduda. Kafka konfiguratsioonis on võimalik seadistada andmete replikatsioon. Sõnumi kaotamisel võib Kafka sõnumi järjekord muutuda, samas RabbitMQ seda ei muuda [6].

4.4.3.4 Järjekord

RabbitMQ võimaldab järjestust muuta kanali siseselt. Samas kasutades RabbitMQ ees koormuse tasakaalustajat ei suuda RabbitMQ kahte erinevat kanalit hoida üksteisega sünkroonis kuna need on eraldatud [6].

4.4.3.5 Hindamine

Tabel 10. RabbitMQ ja Kafka hindamine

	RabbitMQ	Kafka
Aeg	Halvem	Parem
Marsruut	Parem	Halvem
Tarnegarantii	Parem	Halvem
Järjekord	Võrdne	Võrdne

Loodavas prototüübis kasutatakse RabbitMQ, kuna võrreldes Kafkaga on tal paremad marsruutmise võimalused. Antud lõputöös on samuti tähtis sõnumite järjekord mida mingil määral suudab paremini RabbitMQ teostada. Kuna reaktsiooniaeg on mõlemal millisekundites ning väga suurt erinevust ei ole, siis loobutakse Kafka kiirusest, et saada parem marsruutimise loogika.

4.5 Andmebaasid

Iseteenindussüsteemi platvormi ehitamisel lähtutakse faktist, et süsteemi integreeritakse ühe asutuse asemel mitu asutust. See omakorda tähendab, et andmebaasid peaksid tulema toime suurte andmemahtudega ja peaksid olema piisvalt odavad ning omama piisavalt lihtsat funktsionaalsust mis tuleks toime andmemahtude suurenemisel skalleerivuse probleemidega.

Lähtudes eelmistest faktidest jäetakse kõrvale relatsioonilised andmebaasid ning uuritakse lähemalt NoSQL andmebaase, kuna nende tööpõhimõtted sobivad paremini süsteemi nõuete täitmiseks

4.5.1 MongoDB

MongoDB võib vaadelda kui NoSQL andmebaasi mis omab osaliselt relatsioonilise andmebaasi põhimõtteid. Andmebaas kasutab bjson andmetüüpi. Võrreldes relatsioonilise andmebaasiga on võimalik MongoDB-s pärida andmeid ühest tabelist (kollektsioonist) ning andmebaasimootor toetab indekseerimist. Kui andmemahud ületavad üle 50 GB, siis võib olla MongoDB 10 korda kiirem kui MySQL andmebaas [7].

4.5.2 Cassandra

Cassandra on key-value andmebaas. Antud andmebaasi mootor on optimiseeritud kirjutamise operatsiooni läbiviimiseks. Samuti toetab Cassandra horisontaalset skalleerimist ning klastrisse uue sõlme lisamisel kaasatakse uus ressurss automaatselt [8].

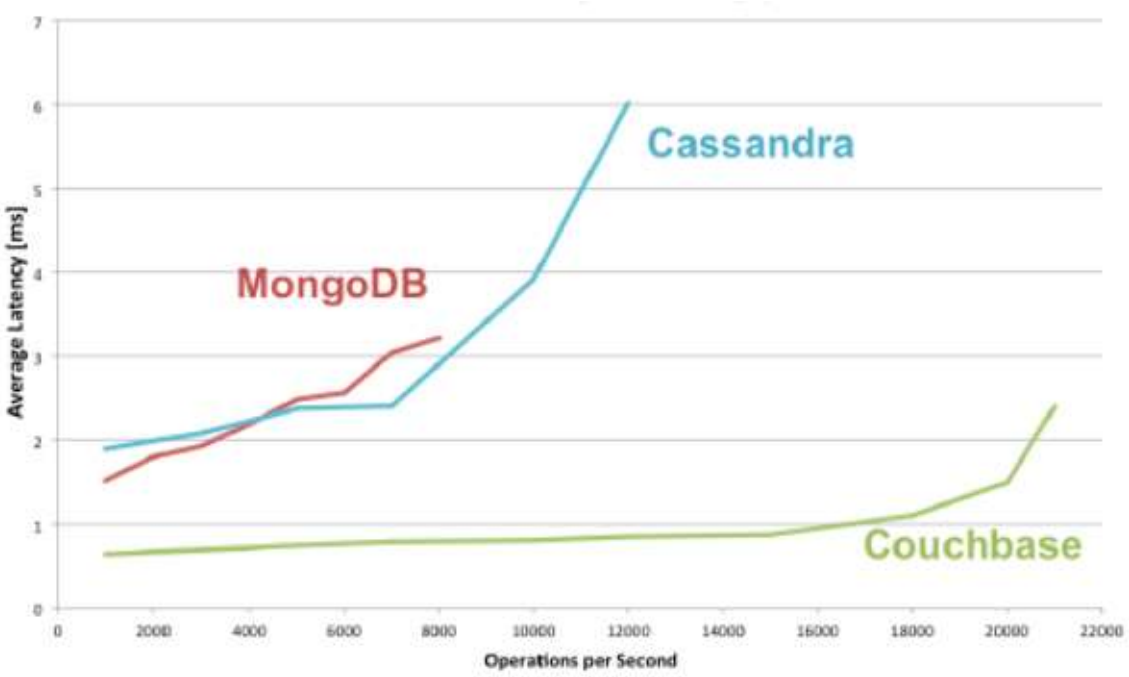
4.5.3 Couchbase

Couchbase on NoSQL andmebaas mis toetab JSON ning AtomPub andmeformaati. Klastris olevad andmebaasiserverid on identsed ning kõik serverid omavad replikeeritud

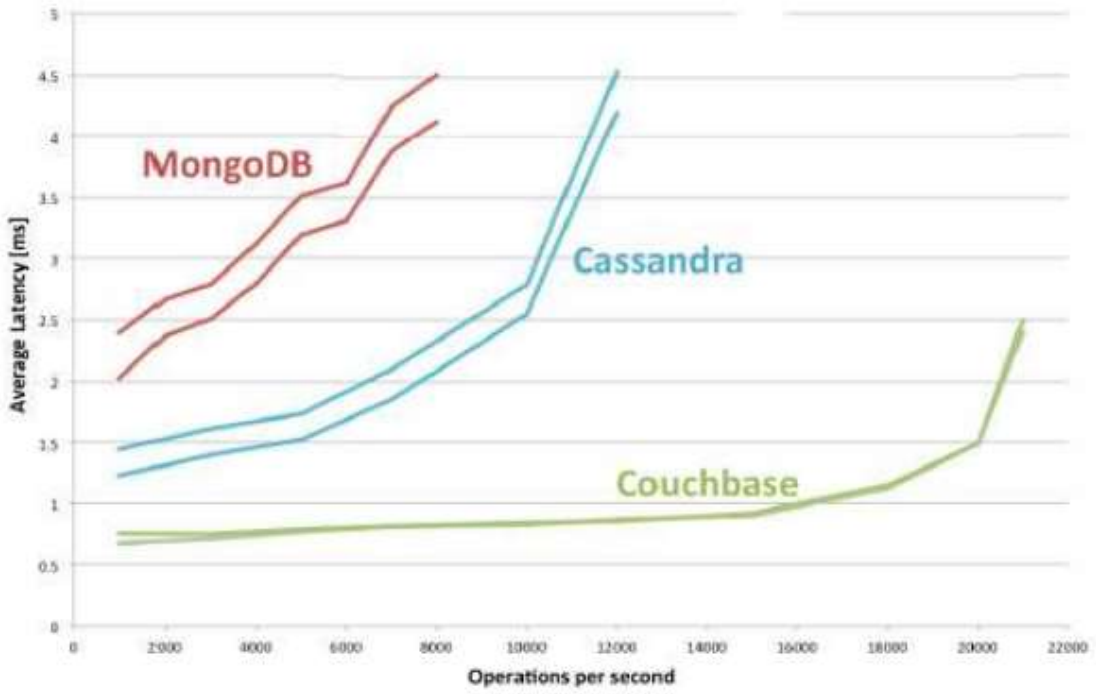
dokumente. Kui klastri sõlme ebaõnnestub kirjutada, siis suunatakse päring teise aktiivse sõlme vastu mis teeb antud andmebaasimootori veakindlaks [8].

4.5.4 Hindamine

Cristina BĂZĂR uuringust selgub, et MongoDB lugemise kiirus on suurem kui Cassandra kuid väiksem kui Couchbase-l. Cassandra ning Couchbase kirjutamise kiirus on suurem kui MongoDB-l [8].



Joonis 8. Andmebaaside lugemise kiiruse võrdlus



Joonis 9. Andmebaaside keskmine kirjutamise võrldus

Analüüsist selgus, et Couchbase NoSQL andmebaas on parim lahendus antud ülesande täitmiseks. Samas MongoDB on kõige lähedasem relatsioonilisele andmebaasile. Kõik andmebaasid omavad eelistusi relatsiooniliste andmebaaside ees. Kuna antud töö aeg on piiratud ning MongoDB on kõige sarnasem relatsioonilisele andmebaasile ning lugemise kiiruselt teisel kohal, omab suurt tarbijaskonda, siis otsustakse hetkel loobuda Couchbase kohesest rakendamisest ning valitakse hetkel MongoDB.

4.6 Raamistikud

Antud peatükis uuritakse võimalikke raamistikke. Hinnatakse raamistike olulisemaid omandusi nagu jõudlus, funktsionaalsused ning lihtsus.

4.6.1 Spring Boot

Spring Boot on Spring Framework raamistiku üks osadest, mis võimaldab alustada tarkvaraarendust võimalikult kiiresti. Raamistik pakub palju valmis tehtud komponente ning mitmeid mittefunktsionaalsete nõuete täitmiseks loodud komponente nagu sisseehitatud server, turvalisus, monitooring, tsentraliseeritud konfigratsioon. Spring Boot ei nõua XML konfiguratsiooni nagu tavaliselt nõuavad teised raamistikud töökeskkonna seadmiseks. Spring Framework kasutab kontrollereid. Kontrollerid

tekitatakse ühe korra, neid hoiustatakse mälus ning jagatakse kõikide päringute vahel. Negatiivseks küljeks antud raamistiku juures on see, et kõik Spring komponendid nõuavad Spring Core komponendi olemasolu [9].

4.6.2 Apache Struts 2

Apache Struts 2 on Java raamistik mida on lihtne laiendada ning enamasti kasutatakse seda, et luua ettevõtte tasandil veebirakendusi. Objektid mis hoolitsevad päringute ja marsruutimise eest nimetatakse tegevusteks. Võrreldes Springi kontrolleritega initsialiseerib Struts tegevusi iga päringu korral. Strutsis on konfigureeritavad MVC komponenid, POJO põhised tegevused, toetab AJAX-it ning seda on võimalik ühendada Hibernate või Spring raamistikega. Negatiivseks küljeks antud raamistiku juures on välja toodud ühildavuse probleemid ning halb dokumentatsioon. Strutsi on sobilik kasutada siis, kui vajatakse stabiilsust [10].

4.6.3 React Native

React Native on Javascripti raamistik millega on võimalik kirjutada koodi korraga Androidi kui ka IOS platvormile. React Native põhineb Facebooki poolt loodud React Javascript raamistikul. Antud platvorm renderdab javascriptis kirjutatud komponendid platvormi põhisteks native UI elementideks. React Native loob silla platvormi API ning Javascripti vahel, millega on võimalik kasutada platvormi komponente (näiteks kaamera). Arendamine React Native platvormil on lihtne, kuna antud raamistik pakub palju arendaja tööriistu ning kuna tegemist on Javascript raamistikuga, siis iga muudatuse valideerimiseks ei ole vaja koodi uuesti kompileerida. React Nativega on võimalik kirjutada komponente Javas ning Swift/Object-C keeles ning teha platvormi põhiseid muudatusi. Nagu eelnevalt mainitud kompileerib antud raamistik ennast platvormi põhiseks koodiks, mis suurendab rakenduse jõudlust ja kiirust. Kõige suuremaks miinuseks peetakse antud raamistiku juures tema noorust ning React Native ei toeta kõiki platvormi funktsionaalsusi. Samuti teeb React Native vea leidmise raskemaks arendaja jaoks [11].

4.6.4 Ionic

Ionic on hübriidraamistik mobiilitelefoni rakenduse kirjutamiseks. Rakendusi on võimalik kirjutada HTML, CSS, TypeScripti ning Javascripti keeltega. Samuti on põhineb Ionic AngularJS-l. Seega Ioncut kasutades ei ole tarvis õppida platvormi põhiseid

keeli nagu Java, Objective-C, Swift. Antud platvormiga on lihtne kirjutada UI põhist funktsionaalust. Ionic põhineb Apache Cordovial ning PhoneGap-il. Neid vajab Ionic et pääseda ligi platvormide native funktsionaalsustele. Ionicu negatiivseks küljeks on see, et antud platvorm põhineb veebipõhistel vaadetel ning sellega kannatab rakenduse jõudlus. Ionicu postiivseks pooleks on toodud välja platvormi kirjutamise lihtsus, mis ei nõua arendajalt suurt õppimisvõimet. Samuti on Ionicu kood taaskasutatav [12] [13].

4.6.5 Xamarin

Xamarin on multiplatormi raamistik mis on loodud mobiilirakenduste kirjutamiseks. Raamistiku arendas välja Microsoft. Antud platvormil on võimalik kirjutada C# keeles rakendusi. Xamarin raamistikuga pole võimalik kasutada native platvormide teeke, kuid pakutakse erinevaid .NET teeke. Xamarinist on kaks versiooni. Xamarin.Android/Ios jõudlus on native rakenduse lähedane. Multiplatvormide vahel saab kasutada ühist koodi äriloogika jaoks. Xamarin.Formsi erineb Xamarid.Android/Ios sellega, et kasutajaliidese komponente pole vaja kirjutada replikeeritult vastavalt igale platvormile vaid on võimalik kasutada mõlemas platvormis olemasolevaid Xamarini poolt pakutavaid komponente. Selle tõttu muutub rakenduse jõudlus aeglasemaks ning antud rakenduse kiirus ei ole enam native rakenduste lähedane. Xamarini rakendust on võimalik kompileerida kahe strateegia alusel JIT (Just in time) ning AOT (ahead-of-time-compilation). AOT kompileerimist peetakse paremaks JIT kompileerimisest, kui tegemist ei ole Javascripti kompileerimisega. Negatiivseks küljeks on see, et Ios platvorm ei toeta JIT kompileerimist. Samas Androidi platvormil valitakse JIT kompileerimine vaikimisi ning on võimalik seadistada Androidi rakenduse kompileerimiseks AOT mehanism [12].

4.6.6 Native

Native rakendused on mobiiliplatvormi põhised. Antud rakenduse positiivseks küljeks on see, et rakendus on võrreldes eelnevalt uuritud raamistikega kiirem. Samuti on võimalik kasutada kõiki ametlikke tööriistu mis on disainitud arendatava platvormi jaoks. Antud rakenduse miinuseks on see, et igale platvormile tuleb erinev rakendus kirjutada. Arendaja peab oskama kõiki platvormipõhiseid keeli [12].

4.6.7 Võrdlus

Spring Boot omab antud projektis eelist Apache Struts2 ees, kuna antud raamistiku on lihtne ja kiire seadistada, omab paremat mäluloogikat ning pakub palju valmis olevaid

komponente. Võrdlusest selgus, et Struts2 on pigem kasulik kasutada siis, kui tahetakse ehitada ettevõtte tasandil it infrastruktuuri ning Spring Boot on loodud projektipõhisteks lahendusteks. Antud platvorm hakkab kasutama Spring Boot raamistikku back-end süsteemi arendamiseks.

Mobiilipõhistest platvormidest uuriti React Nativeit, Ionicut, Xamarini kui ka native rakendusi. Antud platvormi puhul on kiirus oluline. Samuti on vaja kasutada palju native rakenduse funktsionaalsusi. Seega Ionic ei ole kõige sobivam antud ülesande lahendamiseks. Native rakenduse eeliseks on tema kiirus, kuid koodi replikeerimise tõttu otsustatakse aja võitmiseks ning platvormide vaheliste vigade tekkimise võimaluseks välistada native rakenduste kirjutamine. Xamarini ja React Native rakenduste eeliseks on nende kiirus, mis on ligilähedane native rakenduste omaga. Võrreldes React Native-it Xamariniga, pakub React Native paremat kontrolli üle platvormi native põhiste funktsionaalsuste. Seega valitakse platvormi arendamiseks Facebooki poolt loodud raamistik nimega React Native.

5 Mobiilne iseteeninduse platvorm

Antud peatükis kirjeldatakse loodud platormi ning selle toimimist. Samuti kirjeldatakse arendusmetoodikat ning arendusvahendeid.

5.1 Majutus

Süsteemi arenduskeskkonna majutamiseks kasutatakse Dell T430 serverit. Süsteem kasutab 32 GB RAM-i, 4 TB HDD kõvaketast ning 120 GB SSD kõvaketast. Virtuaaliseerimise keskkonna loomiseks kasutatakse VMWare ESXi 6.5 hypervisorit. Virtuaalmasinad kasutavad host süsteemina Ubuntu 17.10 operatsioonisüsteemi. Mikroteenuste hostimiseks kasutatakse Dockerit ning Docker Compose. Mikroteenuste majutamiseks kasutatakse Tomcat 8.5 veebiservereid. All toodud tabelis kirjeldatakse loodud virtuaalmasinaid.

Tabel 11. Virtuaalmasinad

Nimetus	Kõvaketta ruum (GB)	CPU (vCPU)	RAM (MB)	Eesmärk
Bitbucket-smarts	50	1	2048	Koodi hoiustamine ning versioneeriminine
Jira-smarts-fe1	15	3	2048	Agiilsete arendusvahendite majutamine nagu Jira ning Confluence
Smarts-jenkins	40	2	2048	Continuous integration tööriista majutamine
Smarts-mongo-fe1	16	1	2048	NoSQL andmebaasi majutamine
Smarts-pres-web	16	2	4096	Nginx + MySQL serveri majutamine (antud töös ei käsitleta)
Smarts-rabbit-mq	18	1	1024	RabbitMQ majutamine
Smarts-service-fe1	30	3	5120	Dockeri host masin mikroteenuste majutamiseks + proxy server
Smarts-service-fe2	120	3	4096	Dockeri host masin mikroteenuste majutamiseks
Smarts-elastic	50	2	2048	Elastic otsingumootori majutamine
Smarts-service-eth-fe-1	60	1	2048	Ethereumi plokiahela sõlmeserveri majutamine

Smarts-eth-dev	60	1	2048	Ethereumi tarkade lepingute arenduskeskkond koos Test RPC ning Truffle arendusvahendiga. (Antud lõputöös ei käsitleta)

5.2 Metoodika ning arendusvahendid

Iseteeninduse platvormi arendamiseks kasutatakse agiilset tarkvaraarenduse meetodit. Rakendust nimega Jira kasutatakse projekti juhtimiseks. Rakenduses Confluence hoiustatakse platvormi dokumentatsiooni ning Bitbucketis hoitakse rakenduse koodi. Integratsiooni tööriistana kasutatakse rakendust nimega Jenkins mis on ühendatud rakendusega Bitbucket.

Arendusvahenditest kasutatakse koodi kirjutamiseks Itellij, PhpStorm ning WebStorm rakendusi. Andmebaasidega suhtlemiseks kasutatakse NoSqlClient rakendust. Ethereumi plokiahelaga suhtlemiseks kasutatakse Web3 javascript teeki ning sõlmeserveri konsooli.

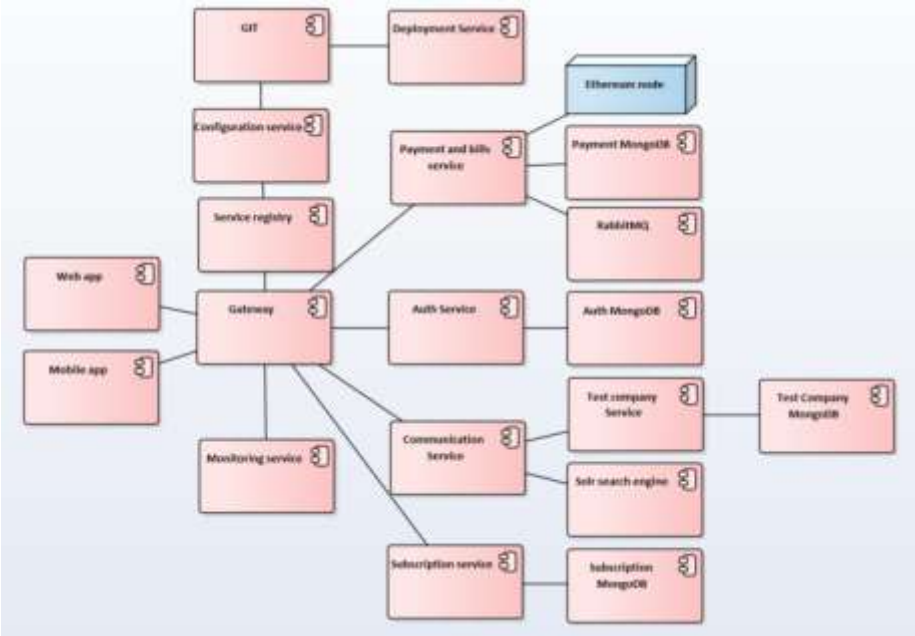
5.3 Mikroteenused

Sõltuvalt jaotuskuubikust ning süsteemi nõuetest jaotati teenused omavahel funktsionaalsete ülesannete järgi nii, et tulevikus oleks horisontaalne skalleerimine võimalikult lihtne. Süsteem jaotati 10-ks erinevaks põhiliseks teenuseks. Antud süsteemi kohta tehakse ka teine magistritöö millega võib lisanduda teenuseid. Alljärgnevalt on toodud teenuste kirjeldused ning omavahelised seosed.

Tabel 12. Mikroteenuste kirjeldused

Teenus	Kirjeldus
Configuration service	Tsentraalne konfiguratsiooni haldus kõikidele teenustele.
Service registry	Teenus, mis teab kõikide teiste teenuste asukohtasid. Samuti toimib tasakaalustajana ning võimalik kehtestada teenuste asukohale piiranguid (näiteks USA päringud suunatakse Auth

	Service 1 teenusele, aga Euroopa päringud suunatakse Auth Service 2 teenusele)
Gateway	Ligipääsupunkt kogule süsteemile. Kõik teised teenused on sisevõrgus ning nendele ligi pääsemiseks saab kasutada ainult Gateway teenust. Gateway kasutab Service registry teenust, et marsruutida päring õigele teenusele.
Monitoring service	Teenus, mis jälgib gateway liiklust ning tuvastab vigaseid päringuid.
Deployment Service	Teenus, mille ülesandeks on mikroteenuste uuendamise protsessi läbiviimine.
Auth Service	Teenus, mis tegeleb autoriseerimisega ning uute kasutajate loomisega. Teised teenused on integreeritud autoriseerimise teenusega, et valideerida ligipääsu õigusi. Auth Service kasutab OAuth 2.0 protokolli.
Subscription Service	Teenus, mis tegeleb uute asutuste lisamise, uuendamise ning kustutamisega. Samuti otsitakse subscription teenusega vastavaid liidestatud asutusi.
Communication service	Kommunikatsiooni teenuse ülesandeks on suhelda liidestatud asutuste süsteemidega (näiteks küsida toodete informatsiooni).
Payment and bills service	Teenus, mis tegeleb maksete kinnitamisega ning arvete koostamise, -avamise ning sulgemisega.
Test Company Service	Arenduse jaoks mõeldud näidis asutuse teenus.
RabbitMQ	Teenus mis vahendab andmeid M2M suhtluses.
Elastic Search engine	Päringute kiiruse suurendamiseks kasutatakse Elastic search otsingumootorit vältimaks iga kord välisliidese vastu pöördumist.

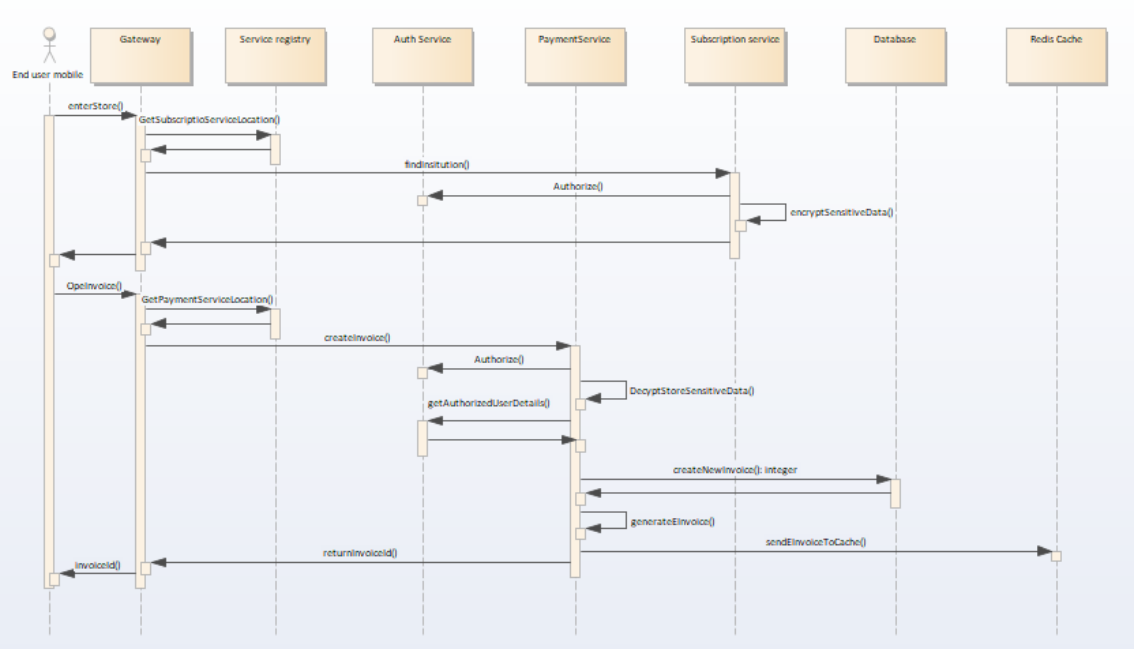


Joonis 10. Loodud mikroteenused

5.4 Põhilised kasutusjuhud

5.4.1 Asutusse sisenemine

Igale kauplusele antakse unikaalne kood QR koodi kujul. Kauplusesse sisenemiseks on vaja skaneerida kaplusele antud QR kood. Asutusse sisse logimiseks peab kasutaja enne logima süsteemi sisse. Sisenemise päring on kujutatud alloleval joonisel.

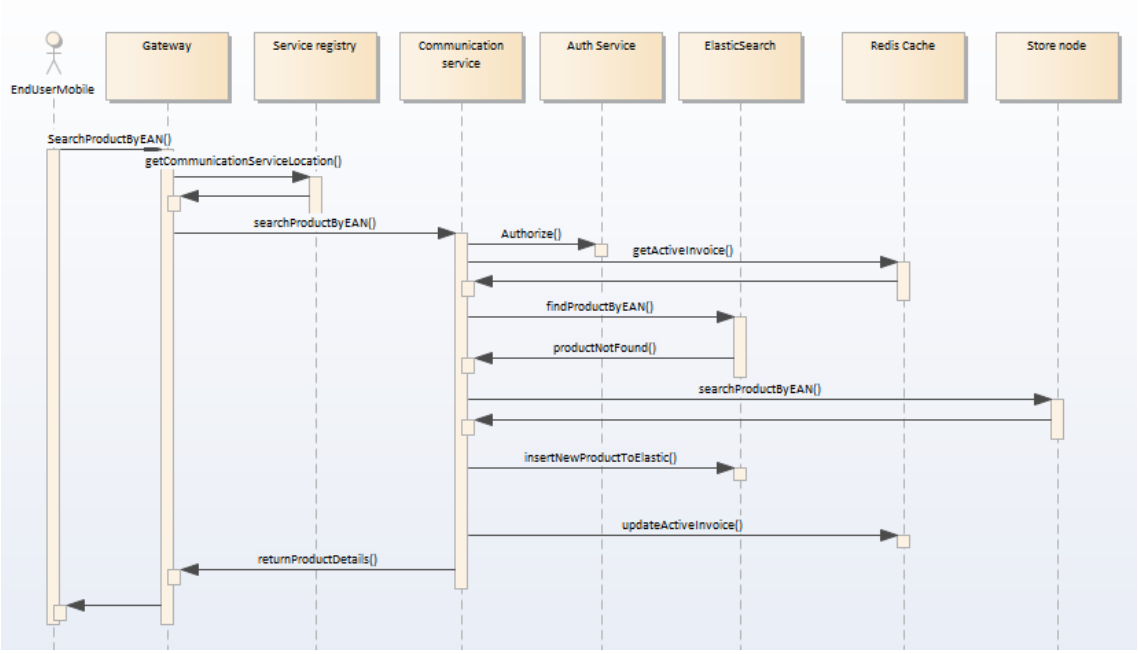


Joonis 11. Asutusse sisenemine

Joonis 12. Asutusse sisenemise vaade

5.4.2 Toote skaneerimine

Toote skaneerimisel eelduseks on süsteemi poolt autoriseeritud kasutaja asutuse sisse logimine. Toote skaneerimise tulemuseks on arve koos lisatud tootega. Skaneerimine toimub läbi mobiiltelefoni kasutades Scandit ettevõte poolt arendatud skaneerimise algoritme. Antud ettevõttega õnnestus saada kokkulepe 1 000 000 000 seadmes nende skaneerimise algoritmi kasutamiseks. Skaneerimise protsessi täpsemalt on kujutatud alloleval joonisel.



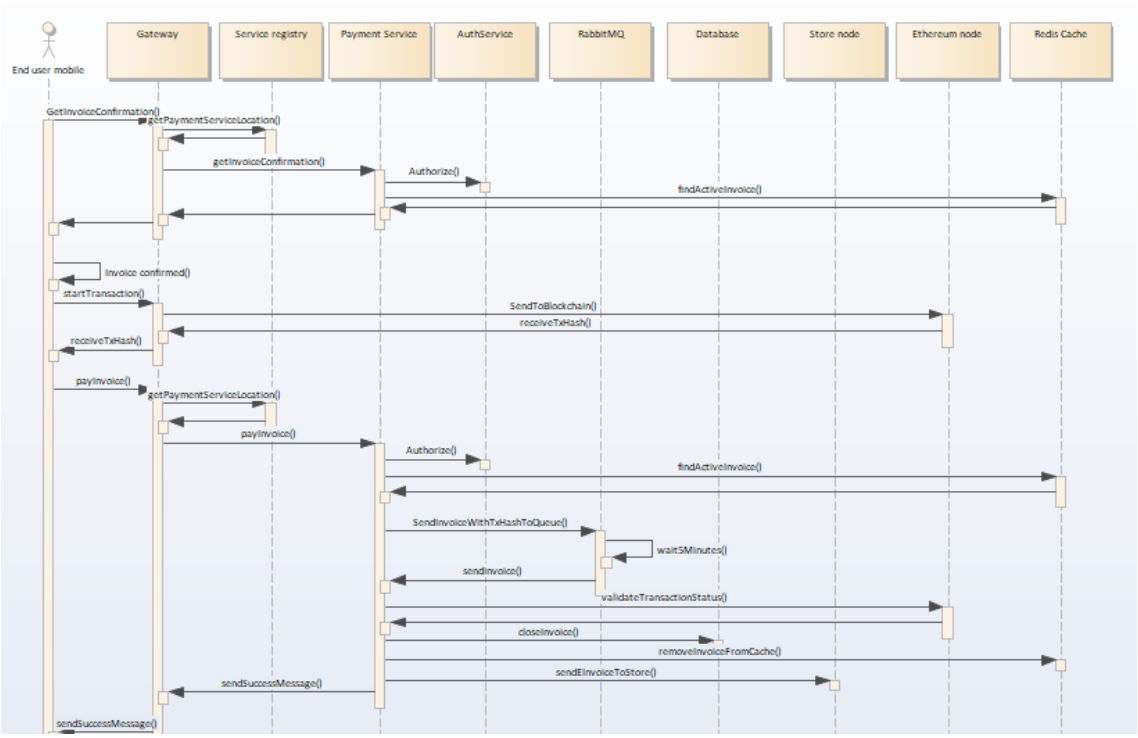
Joonis 13.Toote skaneerimine

Joonis 14.Toote skaneerimise vaade

5.4.3 Arve tasumine

Arve eest tasumiseks on võimalik kasutada Ethereum krüptoraha. Arve tasumise eelduseks on autoriseeritud kasutaja, kes on sisse loginud asutusse, omab Ethereumi rahakotti ning omab mitte vähem kui arve kogusumma väärtuses Etherit. Arve tasumise protsess on toodud alloleval joonisel. Transaktsiooni allkirjastamine toimub mobiiltelefonis, kuna turvalisuse tagamiseks ei tohi pääseda mobiiliplatvorm ligi

rahakotile. Allkirjastatud transaktsiooni aitab vahendada süsteemi kasutuses olev Ethereum-i sõlmeserver. Täpsem kirjeldus arve eest tasumisel on kujutatud alloleval joonisel.

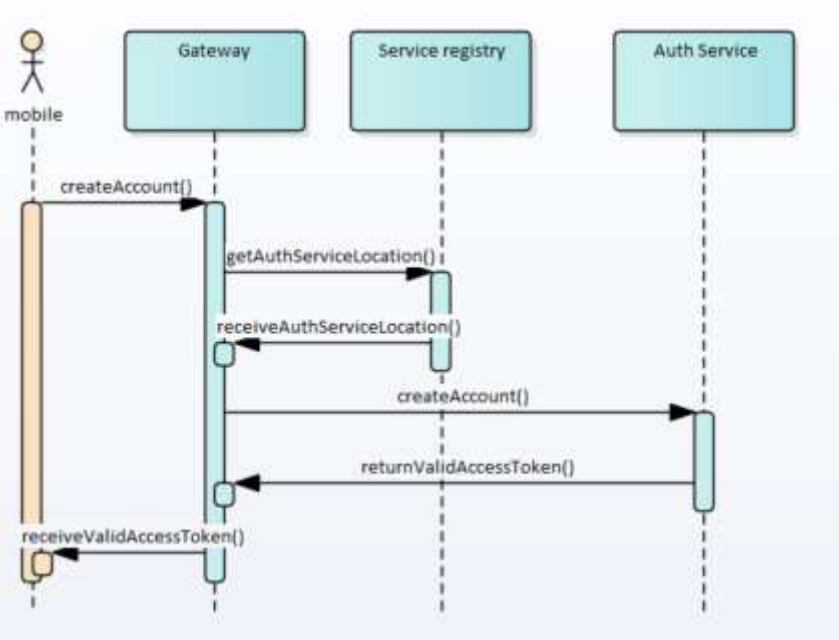


Joonis 15.Arve tasumine

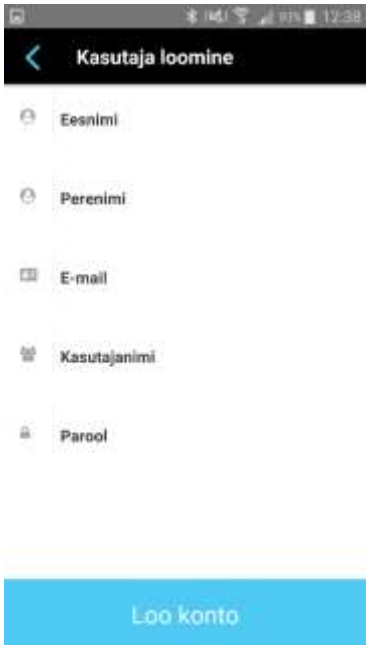
Joonis 16. Arve tasumise vaade

5.4.4 Kasutaja loomine

Kasutaja loomiseks peab uus kasutaja sisestama oma ees -ja perekonnanime, parooli, kasutajanime ning emaili. Kui kasutaja on süsteemi poolt loodud alustab mobiilirakendus Ethereumi rahakoti loomist kasutaja telefonis. Krüptograafiliste võtmete genereerimine toimub mobiiltelefonis ning rahakoti hoiustatakse samuti mobiiltelefonis. Back-end süsteem ei pääse rahakotile ligi. Kasutaja loomine süsteemi back-endis on kujutatud alloleval joonisel.



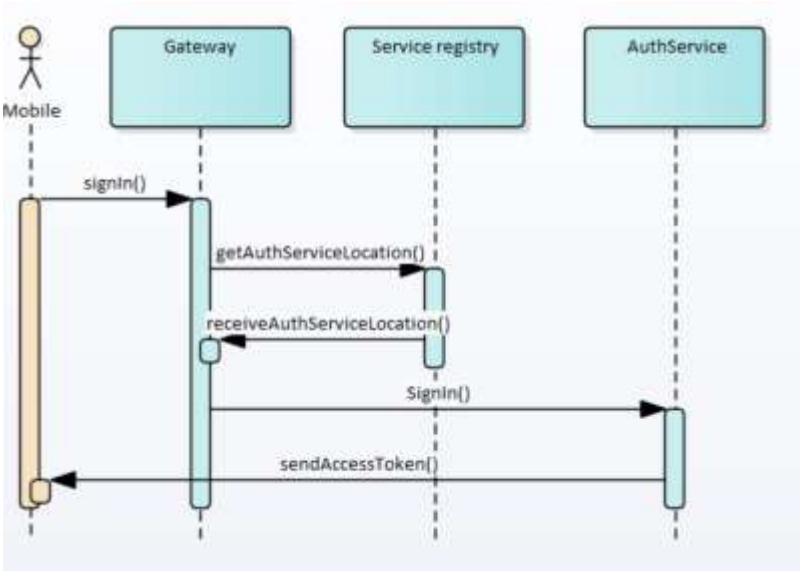
Joonis 17. Kasutaja loomine



Joonis 18. Kasutaja loomine

5.4.5 Autoriseerimine

Keskkonda sisenemiseks peab kasutaja sisestama oma kasutajanime ning parooli. Autoriseermise päring suunatakse süsteemi ligipääsupunkti, kust omakorda suunatakse päring edasi autoriseerimise serverile. Autoriseerimise server tagastab kasutajale ajas aeguva räsiväärtuse, millega antakse õigus kasutada teisi teenuseid mida antud kasutaja vajab. Edaspidi ei pea kasutaja saatma süsteemi päringute jaoks oma isikuandmeid ning iga teenus saab tuvastada isikut kasutades talle antud räsiväärtust.



Joonis 19.Autoriseerimine

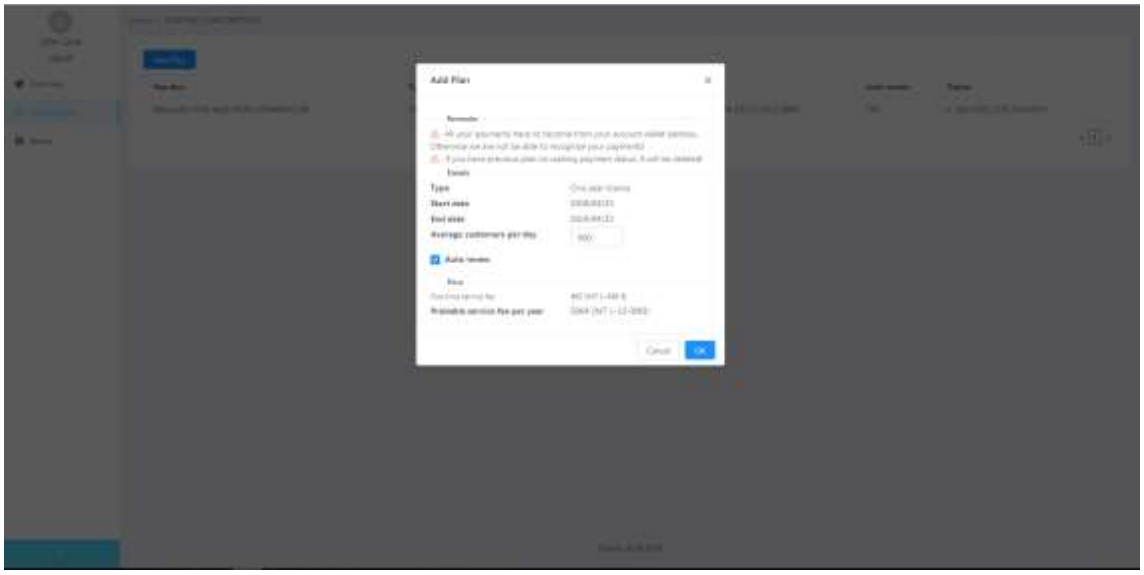
Joonis 20. Autoriseerimise vaade

5.4.6 Keele valik

Antud platvorm on kättesaadav eesti ja inglise keeles. Platvormi vaikimisi keeleks on inglise keel. Kui kasutaja keelt ei leita, siis valitakse automaatselt vaikimisi olev keel. Keele valik toimub automaatselt kasutades selleks mobiiliplatvormil valitud keelt. Keele lisamiseks süsteemi on vaja tekitada uus Javascripti fail mille nimeks peab olema ISO-639 standardi järgi keele kahekohaline lühend. Antud failis tuleb täita tõlkevõtmed ära vajalikke tõlgetega.

5.4.7 Asutuse tellimuse lisamine

Tellimuse lisamiseks peab asutuse töötaja sisse logima asutuse admin keskkonda. Tellimuse lisamiseks tuleb tellijal lisada tellimuse plaan. Tellimuse plaanis tuleb märkida eeldatav süsteemi kasutajate arv päevas. Samuti on võimalik märkida automaatne tellimuse uuendamine. Tellimuse põhjal hakatakse tulevikus määrama teenuse kasutamise hinda ning antud lõputöös tellimuse maksmise osa ei käsitleta.



Joonis 21. Tellimuse lisamine

5.4.8 Poe lisamine

Poe lisamiseks on vajalik aktiivse kasutusõiguse olemasolu. Poe lisamine toimub seitsmes etapis.

- Tellimuse valimine – Asutus valib millisele tellimusele ta sisestab poe
- Poe andmed – Asutus peab sisestama poe nime, registrikoodi, käibemaksu koodi, aadressi, lahtioleku ajad ning avalikud kontaktid

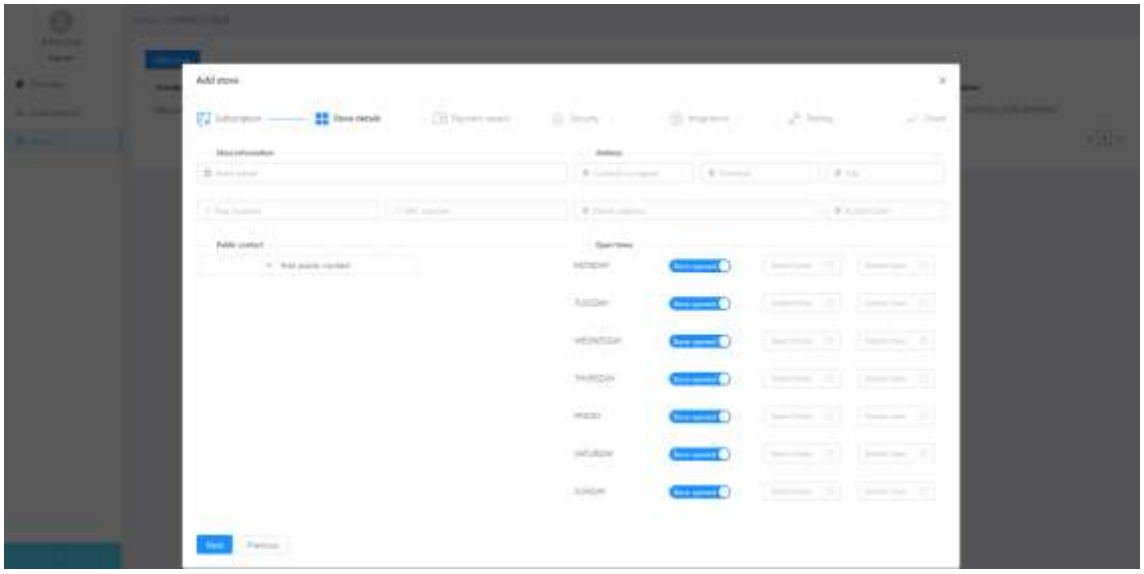
- Maksekanalid – Asutus peab sisestama Ethereumi ning Bitcoini aadressi (lõputöös kasutatakse ainult Ethereumi aadressi)

- Turvastrateegijad – Asutus saab valida milliseid turvastrateegijad rakendatakse ostjate peal. Hetkel on võimalik sisestada 3 strateegia põhist turvameetodit – ostja arvu strateegia, range strateegia ning ajapõhine strateegia. (Strateegijaid on võimalik sisestada, kuid antud lõputöös, strateegia põhist kontrolli ei jõuta realiseerida).

- Integratsioon – Kauplus peab sisestama oma sõlme asukoha koos parooliga millega on võimalik asutuse sõlmele ligi pääseda. Asutuse sõlme loomiseks tehakse teine magistritöö, mille ülesandeks on välja töötada SDK mis aitaks sõlme loomise teha kiireks ja mugavaks.

- Testimine – Poe aktiveerimine pole lubatud enne kui see on süsteemi poolt testitud. Testimiseks luuakse automaattestid, mis kontrollivad ühendatud sõlme kiirust ning valideerivad andmeobjekte ja SDK õiget kasutamist. (Antud lõputöös realiseerida ei jõuta ning seetõttu tagastatakse alati positiivne tulemus)

- Poe lisamise lõpetamine – Selles faasis on võimalik pood aktiveerida kui asutuse integratsiooni automaattestimise faas läbiti edukalt.



Joonis 22. Tellimusele poe lisamine

6 Platvormi testimise tulemused

Antud peatükis testitakse loodud rakendust ning võrreldakse seda SelveExpressi iseteeninduse süsteemiga.

6.1 Platvorm

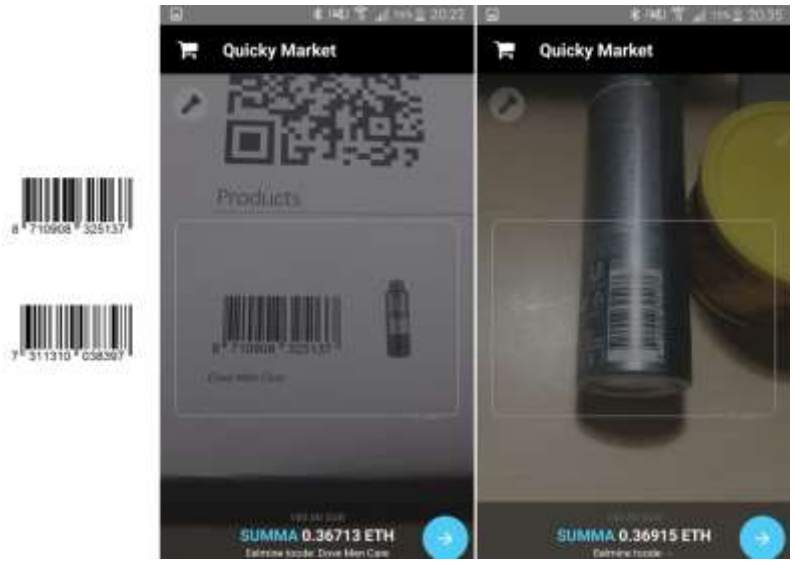
Platvormi testimiseks loodi test kauplus ning liidestati see arendatud iseteenindussüsteemiga.

6.1.1 Skaneerimine

Skaneerimise testimiseks loodi pood nimega Quicky Market ning sisestati kaks toodet – Dove Men Care deodorant ning Dip Nacho cheese dipikaste. Skaneerimise hetkel tegutses platvormil 1 kasutaja ning keskmine skaneerimise aeg jäi alla 1 sekundi. Kauplusesse sisenemiseks loodi kauplusele QR kood ning skaneeriti QR kood kasutades Samsung S5 mobiiltelefon ning Samsung Galaxy Tab A5.

Joonis 23. Kauplusesse sisenemise testimine

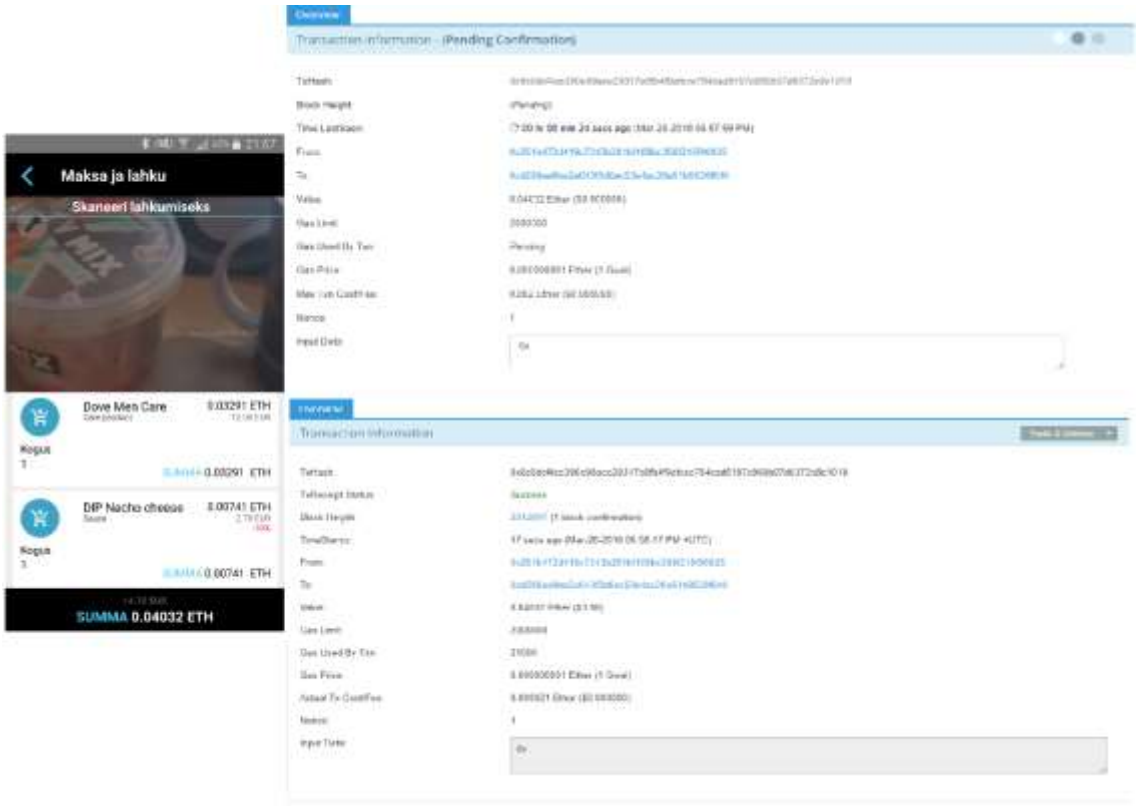
Toodete skaneerimise keskmine aeg oli 1 sekund. Selgus, et Scandit-i skaneerimise algoritm suudab tuvastada ära mitu triipkoodi korraga. Antud rakendusele on see negatiivne, kuna testimise käigus sattusid soovimatud tooted ostukorvi.



Joonis 24. Toodete skaneerimise testimine

6.1.2 Maksmine

Arve maksmise jaoks algatati transaktsioon mis saadeti Ethereumi plokiahelasse. Testimiseks kasutati algselt testvõrku nimega Ropsten, kuid selle mahu suurenemisel otsustati kasutada uuemat võrku nimega RinkeBy. Transaktsiooni algatamiseks skanneeriti kaupluse QR kood. Transaktsiooni valideerimiseks kasutati lehekülge nimega https://rinkeby.etherscan.io/. Keskmine transaktsiooni kinnitamise aeg oli Ropsteni testnetis 1.2 minutit ning RinkeBy testnetis 40 sekundit. Antud testimine toimus testnetis, kus on vähem kaevandajaid kui mainnetis ning seetõttu on testneti transaktsiooni kinnitamised aeglasemad. Iseteenindussüsteemis arve kinnitamise aeg oli keskmiselt 5 minutit ja 6 sekundit. Kuna arve saadetakse 5 minutiks RabbitMQ-sse ootele, et olla täiesti kindel transaktsiooni lõppstaatuses siis võime ooteaja lahutada kogu ajast maha. Seega arve kinnitamise protsess ühe kasutaja puhul võttis skaneerimise, arve saatmisega iseteenindussüsteemi, arve hoiustamisega RabbitMQ, arve valideerimisega ning arve sulgemisega aega 6 sekundit + 5 minutit ooteaega.



Joonis 25. Transaktsiooni kinnitamine

6.1.3 Jõudluse testimine

Antud platvormil testiti toote skaneerimise jõudlust arenduse keskkonnas. Jõudluse testimiseks kasutati rakendust nimega Jmeter. Selgus, et minutis on võimalik skaneerida üle 4000 toote, kui platvormil tegutseb samal ajal 5000 kasutajat. Antud teenused on majutatud Dockeri peale mille host omab 5 teenuse peale kokku 5 GB RAM-i. Samuti on kolme tuumaline virtuaalprotsessor jaotatud ära viie teenuse peale mida võib lugeda märkimisväärselt väheseks, kuid piisavaks arenduskeskkonna jaoks. Arvestades, et teenused pidid hakkama saama väga vähese ressursiga, et korraga saaksid skaneerida 5000 inimest minutis üle 4000 toote, siis võib jääda testi tulemusega rahule ning eeldada, et tootmiskeskkonnas on masinate jõudlused kordades suuremad. Ülejäänud tuhande inimese korral saadi tagasi teenuselt viga.

Joonis 26. Toote skaneerimise jõudluse testimine

6.1.4 Turvalisuse testimine

Tabel 13. Turvatestimine

Pöördumine teenuse poole välisvõrgust ilma ligipääsupunkti läbimata	ebaõnnestus
Pöördumine autoriseerimist vajava ressursi poole autoriseerimata kasutajaga	ebaõnnestus
Võrgupakettidest informatsiooni lugemine	ebaõnnestus
Autoriseeritud kasutajaga teise isiku andmete kättesaamine	ebaõnnestus
Asutusse sisenemine autoriseerimata kasutajaga	ebaõnnestus
Toote otsimine autoriseerimata kasutajaga	ebaõnnestus
Asutuse tundliku informatsiooni kättesaamine	ebaõnnestus
Arve summa muutmine	ebaõnnestus
Krüpto varastamine	ebaõnnestus
Krüptorahakoti hävitamine	õnnestus

Turvatestimisest selgus, et süsteem turvalisususega võib rahul olla. Krüptorahakotti oli võimalik varastada, kuid see oli koodiga kaitstud, mille tulemusel ei olnud võimalik krüptoraha varastada. Selgus, et kui varastajal on ligipääs mobiiltelefoni, siis on võimalik teha kasutajale kahju hävitades kasutaja rahakoti privaatvõtme. Seega peaks olema kasutajal võimalik sisestata ning kopeerida rahakoti privaatvõtmeid.

Võrgupakette prooviti püüda rakendusega nimega WireShark. Kuna süsteem on kaitstud HTTPS protokolliga siis kõik andmed olid krüpteeritud.

Turvatestimisel kasutati mobiilset välisvõrku.

Joonis 27. Autoriseerimata kasutaja pöördumine ressursi poole

6.1.5 Skaleeritavuse testimine

Süsteemi skalleeritavuse testimine toimus sisevõrgus. Skaleeritavust katsetati tellimusmooduli abil. Üks tellimusmooduli server asus arenduskeskkonnas ning teine server käivitati lokaalselt arendaja arvutis. Teise serveri käivitamisel kaasas teenuste register automaatselt uue ressursi ning alustas koormuse tasakaalustamist kahe serveri vahel automaatselt.

Andmebaasi skaleeritavust ei olnud võimalik testida seoses ressursi puuduse ning mahu piiranguga.

6.1.6 Kasutusmugavuse hindamine

Kasutusmugavuse hindamiseks kasutati seitset inimest, kellest 5 olid üliõpilased ning 2 pikaajalise kogemusega IT sektoris töötavat isikut, kes on varasemalt puutunud kokku SelveEksressi lahendusega. Kolm üliõpilast õpib Tallinna Tehnikaülikoolis ning kaks üliõpilast õpivad Tallinna Ülikoolis. Kasutajate vanused oli 23 – 29 eluaastat ning kõik osalejad olid meessoost isikud. Üks osalejatest puutub kokku igapäevaselt krüptovaluutaga, 2 osalejatest puutub kokku aeg-ajalt krüptovaluutaga ning 4 osalejat ei ole puutunud kokku krüptovaluutaga. Kasutajatelt küsiti toote skaneerimise, kasutaja tegemise, raha laadimise, maksmise ning üldise mugavuse kohta. Samuti paluti võrrelda kasutusmugavust SelveEkspressi süsteemidega. Lisaks paluti tuua välja ideid kasutusmugavuse tõstmiseks. Küsitlus viidi läbi suuliselt.

Kuigi tegemist oli väiksemahulise küsitlusega ning täielikult objektiivset hinnangut ei anna, siis toodi välja mõned ühised mõtted. Enamus küsitluses osalenud arvasid, et antud platvormi on mugav kasutada, kuigi krüptovaluuta saamine ning laadimine võib olla keerukas. Ühtselt arvati ka, et kaupluste liitumiseks vajatakse häid läbirääkimise oskusi, et suurendada platvormi kättesaadavust. Enamus uuringus osalejatest tõid välja selle, et platvormis võiks saada ka tavalise deebet - või krediitkaardiga maksta.

6.2 SelveEkspressi võrdlus

Antud platvormi võrreldakse Selveri poolt pakutava SelveEkspressi lahendusega. SelveEkspressi testimine viidi läbi Keila Selveris. Toodete skaneerimise kiirus oli SelveEkspressil praktiliselt kohene ning keskmine skaneerimise aeg jäi kindlasti alla 0.5 sekundi. Subjektiivse hinnangu põhjal leiti, et SelveExpressi kasutajaliides vajaks uuendamist ning parendamist. Samuti testiti expressi kasutamiseks loodavat aega. SelveExpressi kasutamiseks nõutakse Partner kaardi olemasolu. Selle loomiseks ning sisenemiseks kulus kokku 7 minutit ning 56 sekundit. Toodete eest maksmiseks tuli pöörduda iseteeninduspunkti. Maksmise testimine toimus kahe inimesega. Ühe inimesel kulus maksmiseks 1 minuti ning 4 sekundit. Teisele inimesele sooritati turvakontroll ning selle tõttu kulus toodete eest maksmiseks 2 minutit ja 47 sekundit.

Joonis 28. SelveEkspress

Antud platvormil viidi läbi täpselt samasugused testimise juhud nagu SelveEkspressiga. Keskmine toote skaneerimise aeg jäi alla 1 sekundi mille võib ümardada 1 sekundi peale. Kasutaja loomine koos Ethereumi rahakoti loomisega võttis aega 53 sekundit. Rahakotti Etheri laadimiseks kulus 1 minut ja 43 sekundit transaktsiooni aega millele lisandus 1 minut ning 25 sekundit transaktsioon saatmiseks. Loodud testpoodi sisenemine võttis aega keskmiselt 0.5 sekundit. Toote eest maksmine võttis aega keskmiselt 1.5 sekundit (antud süsteemis nimetatakse maksmist transaktsiooni alustamiseks ning plokiahelasse saatmiseks).

Ostukorvi valideerimist antud lõputöös ei kajastata ning ei testita seoses mahu ning aja piiranguga.

Tabel 14. Platvormi võrdlus SelveEkspressiga

	SelveEkspress	Platvorm
Kauplusesse sisenemiseks kuluv aeg	7 minutit 56 sekundit	4 minutit 1.5 sekundit
Toote skaneerimise aeg	< 0.5 sekundit	~ 1 sekund
Ostukorvi maksmise aeg	1 minut 4 sekundit	1.5 sekundit
Nõuded süsteemi kasutamiseks	Partnerkaardi olemasolu, Pangakaardi olemasolu Käiberaha olemasolu	Mobiiltelefoni olemasolu, Ethereumi krüptoraha olemasolu
Ostukontroll	Olemas	Puudub
Skaneerimise seade	Pult	Mobiiltelefon

Kokkuvõte

Antud töö eesmärgiks oli luua mobiilipõhine iseteeninduse platvorm mis muudaks lõppkasutaja jaoks iseteeninduse kasutamise mugavamaks ning kliendi jaoks kasutamise kättesaadavamaks, odavamaks ning tehnoloogiliselt lihtsaks.

Antud platvormi arendamiseks võrreldi, analüüsiti ning valiti vajalikke komponente, raamistikke ning arhitektuure. Analüüsi tulemustest selgus, et Scanditi skaneerimise algoritmid olid kõige sobilikumad antud ülesande lahendamiseks. Arhitektuuri analüüsist leiti, et kõige efektiivsem oleks kasutada mikroteenuste arhitektuuri. Komponentide analüüsist selgus, et M2M suhtluseks sobis kõige paremini RabbitMQ. Andmebaaside analüüsi tulemusena selgus, et antud töö raames oli kõige mõistlikum kasutada MongoDB, kuid kõige kasumlikumaks peeti Couchbase andmebaasi. Back-end raamistikuks valiti Spring Boot ning mobiilirakenduse arendamiseks peeti kõige paremaks raamistikuks React Native-it. Krüptovaluutana otsustati kasutusele võtta Ethereum. Kõik süsteemile püstitatud nõuded said täidetud.

Loodud platvormil testiti jõudlust, kiirust, skaneerimise tõhusust ning turvalisust. Testimise tulemusena selgus, et jõudlusega võib antud ressursse kasutades rahule jääda. Turvalisuse testimine oli edukas. Antud platvormi võrreldi SelveExpressi iseteenindusega. Võrdlemise käigus selgus, et SelveExpress suudab kiiremini skaneerida ning leida tooteid. Samas oli loodud platvormil kauplusesse sisenemine ning väljumine kiirem.

Antud lõputöös ei jõutud täieliku MVP valmimiseni aja piirangu tõttu. MVP valmimiseks on vajalik realiseerida ostukontrolli funktsionaalne rakendamine, automaattestid poe lisamisel kui ka perioodiline integratsiooni testimine ning süsteemi maksupoliitika realiseerimine.

Kasutatud kirjandus

[1] J. M. Curran ja M. L. Meuter, „Self-service technology adoption:comparing three technologies,“ *Journal of Services Marketing,* kd. 19, nr 2, 2005.

[2] D. Namiot ja M. Sneps-Sneppe, „On Micro-services Architecture,“ *International Journal of Open Information Technology,* kd. 2, nr 9, 2014.

[3] T. Cerny, M. J. Donahoo ja D. Trnka, „Contextual Understanding of Microservice Architecture:Current and Future Directions,“ *APPLIED COMPUTING REVIEW,* kd. 17, nr 4, 2017.

[4] E. Ohbuchi, H. Hanaizumi ja L. A. Hock, „Barcode Readers using the Camera Device in Mobile Phones,“ %1 *International Conference on Cyberworlds,* 2004.

[5] I.-C. Lin ja T.-C. Liao, „A Survey of Blockchain Security Issues and,“ *International Journal of Network Security,* kd. 19, nr 5, pp. 653 - 659, 2017.

[6] S. Nakamoto, „Bitcoin: A Peer-to-Peer Electronic Cash System,“ Bitcoin.org.

[7] N. Atzei, M. Bartoletti ja T. Cimoli, „A survey of attacks on Ethereum smart contracts,“ Universit`a degli Studi di Cagliari, Italy, 2016.

[8] P. Dobbelaere ja K. S. Esmaili, „Industry Paper: Kafka versus RabbitMQ,“ Nokia Bell Labs, Barcelona, Spain, 2017.

[9] J. Han, H. E, G. Le ja J. Du, „Survey on NoSQL Database,“ IEEE, Beijing, 2011.

[10] „The Transition from RDBMS to NoSQL. A Comparative Analysis of Three Popular Non-Relational Solutions:Cassandra, MongoDB and Couchbase,“ *Database Systems Journal,* kd. 5, nr 2, pp. 49 -59, 2014.

[11] P. Webb, D. Syer, J. Long, S. Nicoll, R. Winch, A. Wilkinson, M. Overdijk, C. Dupuis, S. Deleuze, M. Simons, V. Pavić, J. Bryant ja M. Bhave, „Spring Boot Reference Guide,“ 2012 - 2018. [Võrgumaterjal]. Available: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/. [Kasutatud 19 03 2018].

[12] M. Mehta , „DZone,“ [Võrgumaterjal]. Available: https://dzone.com/articles/struts-2-vs-springmvc-know-the-difference-choose-t. [Kasutatud 19 03 2018].

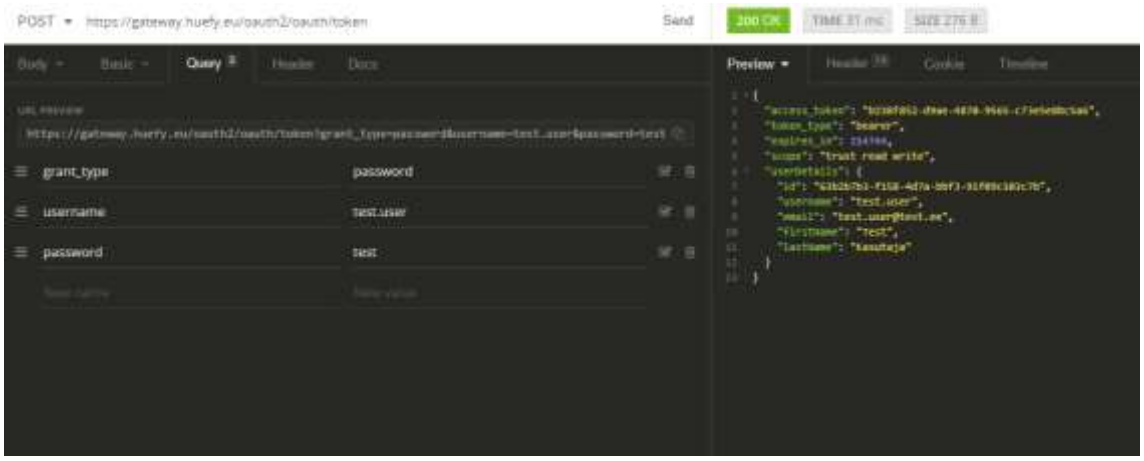
[13] B. Eisenman, Learning React Native, Gravenstein: O' Reilly, 2016.

[14] A. Aggarwal, „Medium,“ Medium, [Võrgumaterjal]. Available: https://medium.com/@ankushaggarwal/ionic-vs-react-native-3eb62f8943f8. [Kasutatud 21 03 2018].

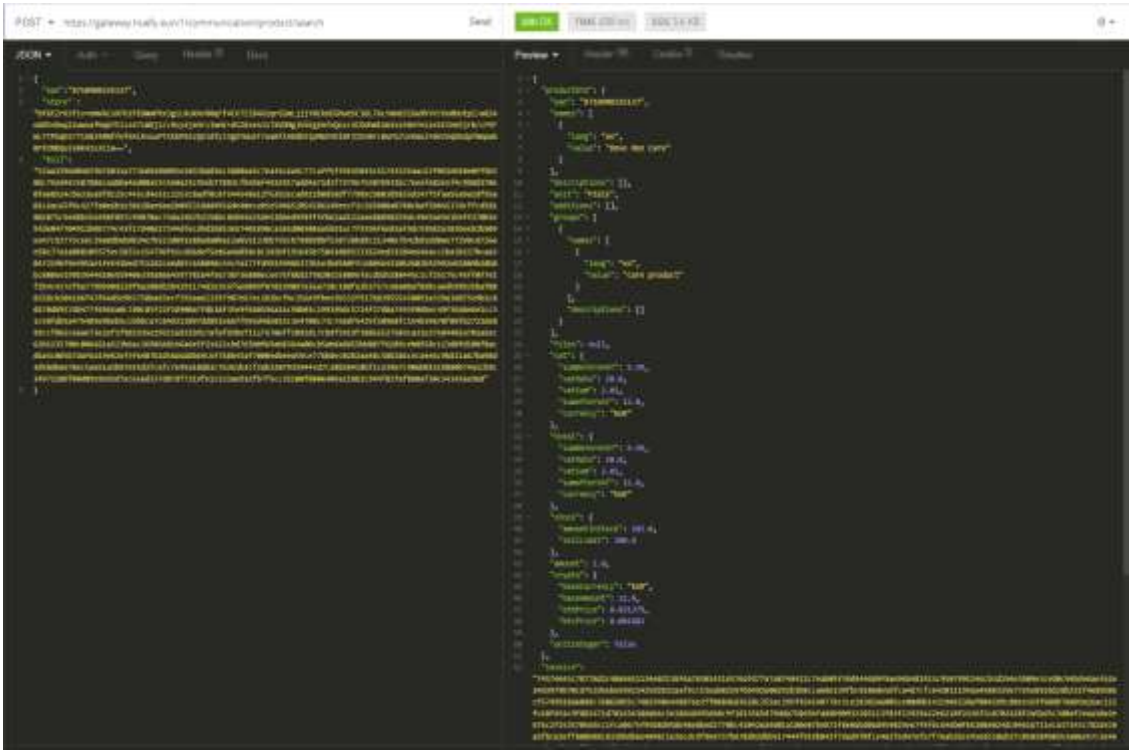
[15] „Cruxlab,“ Disqus, [Võrgumaterjal]. Available: https://cruxlab.com/blog/reactnative-vs-xamarin/. [Kasutatud 21 03 2018].

[16] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas ja S. Gil, „Evaluating the Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the Cloud,“ Systems and Computing Engineering Department, Colombia, 2015.

Lisa 1 – Päringu näited

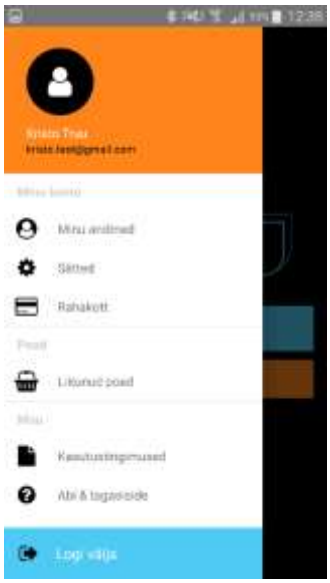


Joonis 29. autoriseermise päring



Joonis 30.Toote otsingu päring

Lisa 2 – Mobiiltelefoni vaated



Joonis 31.Autoriseeritud kasutaja hüpikmenu

Joonis 32. Autoriseeritud kasutaja vaade