TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
TalTech IT Kolledž

Marite Rammo 175895IDDR

Oracle SOA teenuse Java Spring tehnoloogiale üleviimine

Diplomitöö

Juhendaja: Jaanus Pöial
PhD
Kaasjuhendaja: Henry Mäeorg
BSc

Tallinn 2021

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: Marite Rammo

07.01.2021

Annotatsioon

Käesoleva diplomitöö eesmärk on analüüsida erinevaid kaasaegseid tehnoloogiaid, leida neist sobivaimad Oracle SOA teenuse asendamiseks ning need kasutusele võtta. Töös käsitletakse ühe rakenduse planeerimist, arendamist, testimist ja paigaldamist.

Töös tutvustatakse olemasolevat lahendust, seejärel analüüsitakse erinevaid tehnoloogiaid ja tööriistu ning valitakse neist sobivaimad. Diplomitöö praktilises osas on uue rakenduse realisatsioon.

Diplomitöö raames viidi Oracle SOA tehnoloogial põhinev võrgulepingu ja mõõtepunkti vahelise seose edastamise teenus üle Java Spring tehnoloogiale. Töö tulemusena valmis Java Spring Boot rakendus koos testide ning paigalduse tarneahelaga. Rakendus võtab sõnumi vastu, töötleb sisendit vastavalt ärireeglitele, loob uue sõnumi ning saadab selle mõõteandmete haldamise süsteemi.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 26 leheküljel, 6 peatükki, 19 joonist.

// Abstract

Migrating an Oracle SOA Service to a Java Spring Technology

The aim of this thesis is to analyze different modern technologies, find the most suitable ones to replace the Oracle SOA service and take them into use. The thesis describes the planning, development, testing and deployment of one application.

The thesis introduces the existing application, which technologies are used and highlights the advantages and disadvantages of this solution. Before creating a new application, an analysis of the potential technologies is performed, the dependencies and continuous integration and delivery tools that will be used are described.

In the practical part of the thesis is the realization of a new application. There is a solution with code samples. It describes adding dependencies to an application, the architecture of the application, testing and the creation of the delivery pipeline. In addition, the results and suggestions for future developments are presented.

The service of network service agreement and service point relationship synchronization was migrated from an Oracle SOA to a Java Spring technology. The outcome of the thesis is a Java Spring Boot application with tests and delivery pipeline. The created application uses JMS queues for messaging. The application receives the message, processes the input according to the business rules, creates a new message and sends it to the measurement data management system.

The thesis is in Estonian and contains 26 pages of text, 6 chapters, 19 figures.

Lühendite ja mõistete sõnastik

Agiilsus	*Agility,* kiire, hästi koordineeritud ja nutikas lähenemine, mis aitab reageerida ettearvamatustele tarkvara arendusprotsessis [1]
API	*Application Programming Interface,* liides, mis võimaldab luua ühenduse erinevate programmide vahel [2]
CC&B	*Oracle Utilities Customer Care and Billing,* kliendihalduse ja arvelduse süsteem [3]
Dockeri kujutis	*Docker image,* juhiste komplekt Dockeri konteineri loomiseks [4]
DTO	*Data Transfer Object,* andmeedastusobjekt [5]
Java	1995. aastal Sun Microsystems poolt loodud objektorienteeritud programmeerimiskeel ja arendusplatvorm [6]
JMS	*Java Message Service,* Java API kahe või enama rakenduse vaheliseks sõnumite saatmiseks [7]
Järjekord	*Queue,* sihtpunkt punktist punkti sõnumivahetuses [8]
Kasutajaliides	*User interface*, UI
MDM	*Meter Data Management*, mõõteandmete kogumise, töötlemise ja arveldusandmete genereerimise süsteem [9]
Mikroteenused	*Microservices,* tarkvaraarhitektuuri stiil, kus tarkvara koosneb väikestest sõltumatutest rakendustest [10]
Pidev integreerimine	*Continuous integration,* CI, programmikoodi sagedane ülespanemine ühisesse koodihoidlasse [11]
Pidev juurutamine	*Continuous deployment,* protsess, mille käigus tehakse valmis toode kliendile kättesaadavaks [11]
Pidev tarne	*Continuous delivery,* protsesside ahel, mille jooksul viiakse tarkvaraarendamise käigus tehtud koodimuudatused tarkvara tarnimiseni [11]
Plugin, pistikprogramm	Tarkvaramoodul, mis lisab suuremale süsteemile (nt seadmele või veebilehitsejale) täiendavaid funktsioone [12]
Saatja	*Producer,* sõnumite saatja [8]
Skaleeritavus	*Scalability*, süsteemi, mudeli, funktsiooni omadus tulla toime suurenenud töökoormusega [13]
SOA	*Service-Oriented Architecture*, teenusepõhine arhitektuur [10]

Tarbija	*Consumer,* sõnumite vastuvõtja [8]
Task	Ülesanne, mida koodi kokkuehitamise käigus teostatakse [14]
Teenuspunkt	*Service point,* koht, kus toimub mõõtmine
WSDL	*Web Services Description Language,* XML-formaadis teenuse kirjeldus [15]
Võrguleping	*Network service agreement,* leping, millega sätestatakse võrguteenuse kasutamine, üksteisega arveldamine jm
XML	*Extensible Markup Language,* märgistuskeel [15]
XSD	*XML Schema Definition*, kirjeldab, kuidas XML dokument on liigendatud [15]

Sisukord

1 Sissejuhatus 10
2 Olemasolev rakendus 12
2.1 Tutvustus 12
2.2 Tehnoloogia 12
2.3 Probleemid 14
2.4 Oracle SOA head omadused 14
3 Analüüs 15
3.1 Teenuse arhitektuur 15
3.2 Programmeerimiskeele valik 15
3.3 Raamistiku valik 16
3.4 Suhtlus 16
3.5 Tehnoloogiad ja tööriistad 18
3.5.1 Lombok 18
3.5.2 Artifactory ja Nexus 18
3.5.3 WebLogic JMS 19
3.5.4 JAXB 19
3.5.5 Log4j2 19
3.6 Pideva integreerimise ja tarne tööriistad 19
3.6.1 Jenkins vs. Bamboo 20
3.6.2 Docker 20
3.6.3 Kubernetes 20
3.6.4 SonarQube 21
3.6.5 Bitbucket 21
3.7 Rakenduse ärireeglid 21
4 Uue rakenduse realisatsioon 22
4.1 Sõltuvuste lisamine 22
4.2 Arhitektuur 23
4.2.1 JMS-i seadistamine 23
4.2.2 Sõnumite vastuvõtmine 24

4.2.3 Sõnumite saatmine 25
4.2.4 DTO 26
4.2.5 Sündmused 27
4.2.6 *Service* klass 28
4.3 Testimine 30
4.4 Ehitus- ja paigaldusprotsess 31
5 Tulemus ja järeldused 34
6 Kokkuvõte 35
Kasutatud kirjandus 36
Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks 39
Lisa 2 – Rakenduse ärireeglid 40

Jooniste loetelu

Joonis 1. JDeveloper arenduskeskkonnas olev SOA Composite Editor. 13

Joonis 2. Fail SOA muudatuste paigalduseks... 13

Joonis 3. Pluginate lisamine *build.gradle* failis.. 22

Joonis 4. SonarQube seadistamise task. 22

Joonis 5. Rakenduse arhitektuur lihtsustatud kujul. 23

Joonis 6. *application.yaml* JMS järjekordade andmete jaoks....................................... 24

Joonis 7. Sõnumite kuulamine JMS järjekorrast. 25

Joonis 8. *SDPSyncMessage* objekti teisendamine... 25

Joonis 9. *Producer* klass.. 26

Joonis 10. *ServiceAgreementDTO* objekt... 26

Joonis 11. *RequestMessageToServiceAgreementDTOConverter* klass........................... 27

Joonis 12. Sündmuste avaldamine.. 27

Joonis 13. Sündmuse klass. 28

Joonis 14. Sündmuse kuulaja.. 28

Joonis 15. Meetod mõõtmistulemuste sidumiseks teenuspunkti, kanali ja parameetriga.
.... 29

Joonis 16. Meetod virtuaalse kanali lisamiseks... 29

Joonis 17. Näide ühiktestimisest. 30

Joonis 18. Ehitus- ja paigaldusprotsess. 32

Joonis 19. Analüüsitulemused SonarQube kasutajaliideses. 33

1 Sissejuhatus

Tarkvaraarendamise protsessis muutub järjest olulisemaks, et tarkvara arendamine ja muutmine oleks lihtne ja tulemus kliendile kiiresti kättesaadav. Suur osa ettevõttesisestest arendustest kulub integratsioonide arendamisele. Paljud suured ettevõtted kasutavad tänapäeval erinevate süsteemide vaheliseks suhtluseks tootestatud SOA lahendusi. Üheks suuremaks neist on Oracle SOA. Uuemate vabavaraliste tehnoloogiate ja vahendite tekkimisel on aga neile tekkinud alternatiive.

Oracle SOA teenuste arendamisel ja kasutamisel esineb mitmeid probleeme. Põhilisteks probleemideks on spetsialistide nappus, mille tõttu kannatab teenuste arendamise kiirus ja kvaliteet ning kergelt ülesseadistatavate automaattestide võimekuse puudumine, mis takistab pidevat tarnet (ingl *continuous delivery*). Lisaks sellele ka logimise aeglus, paigalduste keerukus ning arendamiseks kasutatava programmi ebamugavus.

Diplomitöö eesmärk on analüüsida erinevaid kaasaegseid tehnoloogiaid, leida neist sobivaimad Oracle SOA teenuse asendamiseks ning need kasutusele võtta. Töös käsitletakse ühe rakenduse planeerimist, arendamist, testimist ja paigaldamist. Autori panus töös on rakenduse planeerimine, praktilises osas arendamine, testimine ning seetõttu on põhirõhk eelkõige nendel.

Metoodikaks on sobivate tehnoloogiate väljavalimine mitme kandidaadi seast. Valimiseks tutvutakse ekspertide poolt koostatud kirjandusega ning analüüsitakse erinevaid võimalusi, et leida sobivaim lahendus probleemide lahendamiseks ning süsteemi kaasaegsemaks muutmiseks.

Diplomitöö teises peatükis tutvustatakse olemasolevat rakendust, milliseid tehnoloogiaid see kasutab ja tuuakse välja selle lahenduse probleemid ning head küljed.

Kolmandas peatükis analüüsitakse ning leitakse uue rakenduse arendamiseks sobivaid tehnoloogiad. Kirjeldatakse kasutusele tulevaid sõltuvusi, milliseid pideva integreerimise ja tarne tööriistasid kasutatakse. Lisaks tuuakse välja rakenduse ärireeglid, mida arendamise tulemusel rakendus peab täitma.

Neljandas peatükis on diplomitöö praktiline osa ehk uue rakenduse realisatsioon. Seal tuuakse välja lahenduskäik koos koodinäidistega. Kirjeldatakse sõltuvuste lisamist rakendusele, rakenduse arhitektuurilist ülesehitust, testimist ning paigalduse tarneahela loomist.

Viiendas peatükis tuuakse välja tulemused ning järeldused, pakutakse välja edasiarendamise võimalusi.

2 Olemasolev rakendus

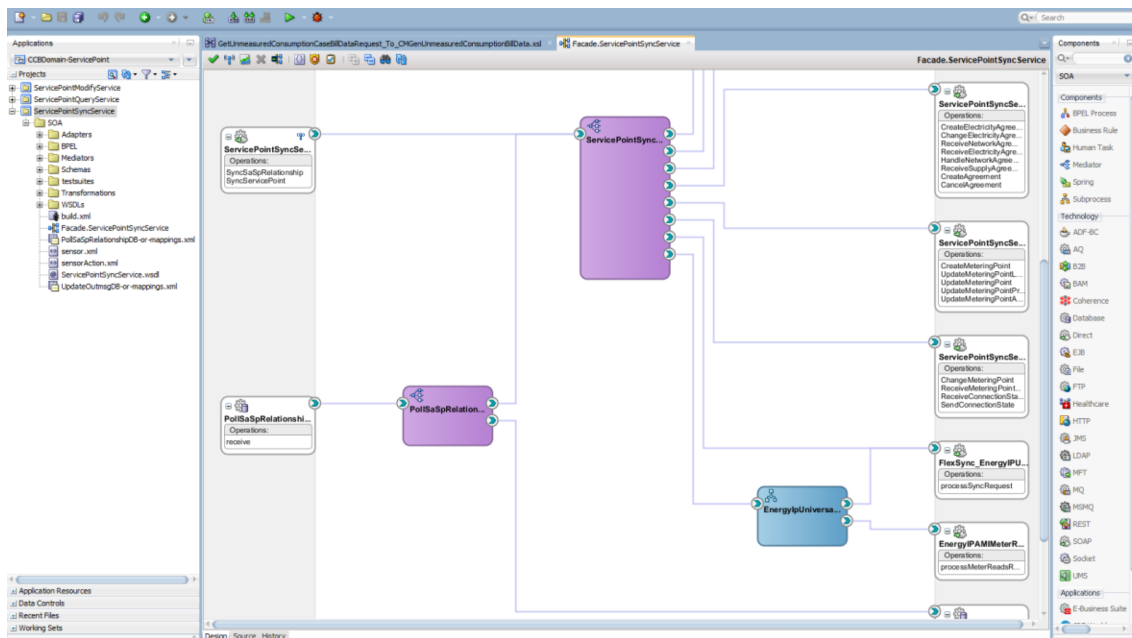
Peatükis tutvustatakse olemasolevat rakendust, kirjeldatakse kasutatavaid tehnoloogiad ja tuuakse välja antud lahenduse probleemid ning head küljed.

2.1 Tutvustus

Tegemist on võrgulepingu ja teenuspunkti vahelise seose edastamise teenusega. Kui rakenduses CC&B (*Oracle Utilities Customer Care and Billing*) uuendatakse kliendi konto ja teenuspunktiga seotud informatsiooni, siis muutunud andmed peavad jõudma mõõteandmete haldamise süsteemi MDM (*Meter Data Management*). Teenus peab tagama, et arvelduse aluseks olevad mõõteandmed jõuaksid reaalajas mõõteandmete süsteemi ja see on oluline selleks, et klientidele esitatud võrguteenuste arvetel oleksid korrektsed andmed.

2.2 Tehnoloogia

Olemasolev teenus on arendatud kasutades Oracle SOA Suite raamistikku. Oracle SOA Suite on terviklik tarkvarakomplekt, mis võimaldab integratsioone luua, juurutada ja hallata kasutades teenusepõhist arhitektuuri (ingl *Service-Oriented Architecture*, SOA). Rakenduste loomiseks kasutatakse Oracle JDeveloper arenduskeskkonda. Antud keskkonnal on SOA Composite Editor, mis võimaldab kokku panna SOA projekti komponente ja tehnoloogiaid (vt Joonis 1). SOA rakenduste jälgimiseks ja haldamiseks on veebipõhine kasutajaliidese tööriist Oracle Enterprise Manager Fusion Middleware Control. [16]



Joonis 1. JDeveloper arenduskeskkonnas olev SOA Composite Editor.

Olemasolevas lahenduses kasutatakse evituseks (ingl *deployment*) Bamboo keskkonda ja monitooringuks Zabbix tarkvara. Suhtluseks teiste rakendustega kasutatakse SOAP veebiteenuseid.

Paigalduste tegemine on osaliselt automatiseeritud. Kõigepealt tuleb soovitud koodimuudatused lisada Bitbucketis projekti spetsiaalsesse *release* harusse (ingl *branch*) ning käsitsi lisada eraldi spetsiaalsesse faili vastavate SOA domeenide, *commit'*ide ning teenuste nimed, mida soovitakse paigaldada (vt Joonis 2). Järgnevalt toimub automaatne valitud muudatuste paigaldamine kasutades Bamboo keskkonda.

```
1  <releaseDeploymentConf>
2      <prepareConf>
3          <targetEnvironment>test</targetEnvironment>
4          <releaseBranch>release/Rel-20.3</releaseBranch>
5          <prepareDomains>
6              <domain>CommonDomain.3f85d6bfdda</domain>
7               <domain>CCBDomain.be71b23bab8</domain>
8          </prepareDomains>
9      </prepareConf>
10     <deploymentConf>
11         <deploymentUnits>
12             <Composite>
13                 <path>/CCBDomain/ServicePoint/Facade/Facade.ServicePointSyncService</path>
14             </Composite>
15         </deploymentUnits>
16     </deploymentConf>
17 </releaseDeploymentConf>
```

Joonis 2. Fail SOA muudatuste paigalduseks.

2.3 Probleemid

Peamiseks Oracle SOA platvormil rakenduste arendamise probleemiks on arenduskiirus. Kuna Oracle SOA on Eestis küllaltki vähekasutatav platvorm, siis selle arendajaid ja spetsialiste on raske leida. Arendajad peavad õppima uut raamistikku, mida mujal tõenäoliselt vaja ei ole. Uue raamistiku õppimine on aga ajamahukas ja arendamisele kuluv aeg seega palju suurem kui see oleks mõne populaarsema raamistiku puhul. Seetõttu oleks mõistlik kolida Oracle SOA teenused mõnele rohkem kasutatavale tehnoloogiale.

Ettevõttes kasutusel olevad Oracle SOA teenused ei ole testidega kaetud. Kui arendaja koodis muudatusi teeb või uut loogikat juurde arendab, siis ei saa kunagi täiesti kindel olla, et sellega midagi eelnevalt arendatut katki ei lähe. Lisaks sellele pidurdab testide puudumine automaatpaigaldust, mis takistab otseselt pideva tarne põhimõtete rakendamist. Testide kirjutamine Oracle SOA puhul on võimalik, kuid nende kirjutamine ja automatiseerimine on keeruline ning tõenäoliselt seetõttu neid siiani arendatud ei ole.

Teenuste testimine toimub käsitsi, selleks kasutatakse SoapUI programmi. Iga teenuse juures on SoapUI projekt, millega saab teenuse töötamise läbi katsetada. Manuaalne testimine on aga rutiinne, aeganõudev ja on suur võimalus, et mõni viga jääb märkamata.

Arenduseks kasutatakse programmi Oracle JDeveloper, mis on aga ebamugav, kuna lakkab tihti töötamast ja kuna programmikood ei salvestu automaatselt, siis võivad ka tehtud muudatused kaduma minna.

Lisaks ei ole Oracle SOA mõeldud keerulise loogika jaoks, kuid teenuste transformatsioonikihti on arendatud mitmeid keerulisi arvutusi [17]. Oracle SOA teenuste paigaldus on küllaltki keeruline ja aeganõudev ning olemasolevas lahenduses on logimine aeglane.

2.4 Oracle SOA head omadused

Oracle SOA Suite kasutamisel on ka mitmed head omadused, näiteks automaatsed *retry*'d ning sisseehitatud veahaldus. Kasutajad saavad kasutajaliidese kaudu jälgida protsesside töötamist ja vajadusel neid uuesti käivitada.

3 Analüüs

3.1 Teenuse arhitektuur

Olemasolev rakendus on loodud kasutades teenusorienteeritud arhitektuuri. Uus rakendus arendatakse aga kasutades mikroteenuste arhitektuuri. Mõlemad erinevad traditsioonilisest monoliitsest arhitektuurist selle poolest, et igal teenusel on omaette vastutus, mõlemad on skaleeritavad ja agiilsed lähenemised. Nii SOA kui mikroteenused koosnevad nõrgalt sidestatud ja korduvkasutatavatest komponentidest. [10]

SOA-ga võrreldes kasutatakse mikroteenuseid eraldiseisvalt toimivate teenuste loomiseks, nii et need oleksid veel rohkem agiilsed, skaleeritavad ning vastupidavad. Kuna rakendus on jaotatud väiksemateks osadeks, siis seda on lihtsam mõista, arendada ja testida. Mikroteenustel on mitmeid eeliseid, näiteks nende paigaldamine ja hooldamine on lihtsamad, sest nad võimaldavad pidevat ja automaatset paigaldamist. Iga teenus mikroteenuste arhitektuuris lahendab ühte äriprobleemi ning on eraldiseisvalt kasutatav ja arendatav. Üks tiim vastutab ühe või mitme teenuse eest. Mikroteenuseid kasutades saab koodi lihtsamini uuendada ning kasutada erinevaid tehnoloogiaid erinevate teenuste arendamisel. Kuna komponendid on üksteisest sõltumatud, siis ühe teenuse ülekoormuse puhul saab jõudlust juurde anda ainult sellele teenusele, mitte ei pea seda kõikidele lisama. [10], [18]

3.2 Programmeerimiskeele valik

Ettevõttepoolne nõue on kasutada Java programmeerimiskeelt. Java on 1995. aastal Sun Microsystems poolt loodud objektorienteeritud programmeerimiskeel ja arendusplatvorm. 2009. aastal soetas Suni ettevõte Oracle Corporation, mis on nüüd Java keele peamine arendaja. Java on tasuta ja avatud lähtekoodiga ning töötab paljudel erinevatel platvormidel. [6]

Java arendamiseks kasutab autor Intellij IDEA arenduskeskkonda.

3.3 Raamistiku valik

Mugavamaks arendamiseks on soov kasutusele võtta mõni Java raamistik. Mikroteenuste arendamiseks enim kasutatud Java raamistik on Spring Boot, kuid on ka mõningaid viimastel aastatel pead tõstnud alternatiive, näiteks Quarkus ja Micronaut. Quarkus on raamistik, mis võimaldab kiiremat alglaadimise aega ja väiksemat mälukasutust. Micronaut on raamistik, mis on mõeldud pilvepõhiste mikroteenuste jaoks. Kumbki neist ei paku aga sama ulatuslikke võimalusi nagu Spring Boot raamistik. [19]

Spring Boot on Spring raamistiku täiendus [20]. See on raamistik, mida arendajatel on väga mugav kasutada, sest projekti ülesseadmine on ääretult lihtne. Lisaks sellele on raamistikul põhjalik dokumentatsioon, rohkelt erinevaid õpetusi ja näidiskoodi. [19] Spring Boot puhul on võimalik kasutada paljusid erinevaid sõltuvusi, neid saab lisada juba projekti genereerides või hiljem arendamise käigus juurde. Kui võrrelda nende raamistike kogukonnapõhiseid näitajaid, siis näiteks Stack Overflow veebilehel, kus on kõige suurem tarkvaraarendajate kogukond, on Spring Boot märksõnaga küsimusi 94508, kui aga Micronaudi märksõnu on ainult 819 ja Quarkusel 1068 [21]. See on oluline näitaja, sest arendamise käigus abi vajades on sel juhul suurem võimalus, et kellelgi on veel samasugune probleem esinenud.

Seega analüüsi tulemusena valitakse uue rakenduse raamistikuks Java Spring Boot. Eeliseks on ka see, et kui olemasoleva Oracle SOA Suite kasutamise eest tuleb tasuda litsentsitasu, siis Java Spring tehnoloogiat saab kasutada täielikult litsentsitasudeta [19].

3.4 Suhtlus

Enamik mikroteenuste vahelistest suhtlusest toimub kas HTTP päring-vastus (ingl *request-response*) tüüpi API-de (*Application Programming Interface)* või asünkroonse sõnumivahetuse kaudu [22].

Loodav rakendus suhtleb kahe süsteemiga, nendeks on CC&B ning MDM. Rakendusele saadab sõnumeid CC&B, mis on monoliitne pärandsüsteem (ingl *legacy*) ning ei toeta uuemaid tehnoloogiaid. Seega tuleb arvestada, millised võimalused omavaheliseks suhtluseks sel juhul on. CC&B dokumentatsiooni põhjal selgub, et kasutada saab SOAP veebiteenuseid või JMS-i. Rakendus saadab sõnumeid edasi MDM-i, mille puhul on andmevahetuseks samuti võimalik kasutada JMS-i või SOAP veebiteenuseid.

Veebiteenus on standardite või protokollide kogum andmete vahetamiseks kahe seadme või rakenduse vahel [23]. Lihtsamalt tähendab see programmide omavahelist suhtlemist ja andmevahetust üle hariliku veebi [24]. Veebiteenused kasutavad päringu tegemiseks tavaliselt HTTP protokolli. Veebiteenuseid on kahte tüüpi, nendeks on SOAP ja RESTful veebiteenused. [23]

Veebiteenuste puhul on tavaliselt tegemist sünkroonse suhtlusega. Kui tehakse HTTP päring, siis jääb rakendus veebiteenuselt vastust ootama ehk selleks ajaks blokeeritakse edasitöötamine. See tekitab kahe rakenduse vahel tugevat sidusust. SOAP veebiteenuste puhul on võimalik luua ka asünkroonne suhtluskanal. Sel juhul rakendus ootama jääma ei pea ning siis on rakenduste vahel nõrk sidusus, kuid selle halvaks omaduseks on lisakeerukus lisapäringute näol. [25] SOAP veebiteenuste puhul kasutatakse WSDL (*Web Services Description Language*) faili, kus on kirjas kogu info veebiteenuse kohta, näiteks mis tegevusi saab teha ja millised on andmetüübid. Kui WSDL peaks aga muutuma, siis tuleb teha koodimuudatused kõikides teenustes, mis seda kasutavad. [26] Olemasoleva Oracle SOA teenuse puhul kasutataksegi teiste rakendustega suhtluseks SOAP veebiteenuseid.

JMS (*Java Message Service*) on Java API kahe või enama rakenduse vaheliseks sõnumite saatmiseks. See on sõnumivahetuse standard, mis võimaldab sõnumeid luua, saata, vastu võtta ja lugeda. JMS-i kasutamiseks on vaja JMS-teenuse vahendajat (ingl *broker*), nendeks on näiteks Apache ActiveMQ, RabbitMQ, JBoss, Oracle WebLogic Server. [7]

Rakendus peab lihtsalt saatma sõnumi JMS-i serveri sihtpunktile (ingl *endpoint*). Sihtpunktid saavad olla kahte tüüpi: järjekord (ingl *queue*) või teema (ingl *topic*). Sõnumite saatjat nimetatakse saatjaks (ingl *producer*) ja sõnumite vastuvõtjat tarbijaks (ingl *consumer*). Järjekorras olevaid sõnumeid tarbib ainult üks tarbija ja neid on võimalik tarbida erinevas järjestuses. Järjekordi kasutatakse punktist punkti sõnumivahetuses. Teema sõnumeid saab tarbida rohkem kui üks tarbija ehk sel juhul on võimalik samaaegselt suhelda mitme rakendusega ning see on publitseeri-telli (ingl *publish-subscribe*) sõnumivahetus. [8]

JMS kasutades ei sõltu rakendus otseselt teistest rakendustest, sest sõnumitootja ja -tarbija suhtlevad omavahel läbi vahendaja. Kui võrrelda JMS kasutamist veebiteenustega, siis JMS puhul rakendused ei pea teadma, kuidas omavahel suhelda, sest teenused ei suhtle

üksteisega otse [18]. Tegemist on asünkroonse sõnumivahetusega [25]. Sõnumit saab saata ka siis, kui teine osapool ei ole kättesaadav. Mikroteenuste põhises arhitektuuris, kus palju erinevaid rakendusi suhtlevad omavahel, on oluline, et ühe teenuse rikke või ülekoormuse korral, ei oleks takistatud teiste rakenduste töö. Kui peaks juhtuma, et sõnumeid vastuvõttev teenus lõpetab töötamise, siis teenusevahendaja saab sõnumid ikka kätte ning säilitab need hilisemaks kasutamiseks [27]. JMS kasutamise puuduseks on see, et vigu on keerulisem püüda [18].

Analüüsi tulemusena otsustatakse, et eelistatum on kasutada JMS-põhist sõnumivahetust, sest sel juhul ei suhtle rakendused omavahel otse ning tänu sellele ei sõltu loodav rakendus teistest rakendustest. Seega rakenduses kasutatakse sõnumivahetuseks JMS järjekordi ning JMS-teenuse vahendajana Oracle WebLogic Serverit. Antud vahendaja kasutamise otsus tehti ettevõtte poolt, kuna antud lahendus oli juba varasemalt kasutusel. Alternatiivina ja autori eelistusena oleks kasutada mõnda tasuta teenusepakkujat nagu ActiveMQ või RabbitMQ.

Kuna valitud Java raamistikuks on Spring Boot, siis on oluline välja tuua, et Spring raamistikul on olemas JMS API kasutamist lihtsustav JMS integreerimisraamistik [28].

3.5 Tehnoloogiad ja tööriistad

3.5.1 Lombok

Lombok ehk Project Lombok on Java teek, mida kasutatakse klassides nii-öelda genereeritava koodi ja meetodite vähendamiseks, näiteks *getter*, *setter*, *toString* meetodid. Seda tehakse lisades Java klassidesse annotatsioone, näiteks @*Getter*, @*Setter*, @*ToString*. Annotatsioonide nimed ise kirjeldavad hästi, milleks neid kasutada saab. Näiteks klassile @*Getter* annotatsiooni lisamisel genereeritakse automaatselt kõikidele väljadele *getter* meetodid. [29], [30]

3.5.2 Artifactory ja Nexus

Projektile lisatakse JFrog Artifactory plugin ja seaded. Artifactory on teekide ja paigalduspakkide hoidla [31]. Antud rakenduse puhul hoitakse seal projektis vajaminevaid teeke. Tänu sellele ei pea neid Maveni hoidlast iga kord alla laadima, vaid

saab seda teha sealt. Dockeri kujutiste (ingl *Docker image*) hoiustamise jaoks kasutatakse pakkide hoidlat Sonatype Nexus [32].

3.5.3 WebLogic JMS

Sõnumite saatmiseks ja vastuvõtmiseks kasutatakse WebLogic JMS-i. WebLogic JMS on sõnumsüsteem, mis on integreeritud WebLogic Serveri platvormi. See toetab täielikult JMS-i spetsifikatsiooni ja pakub mitmeid laiendusi. [33]

3.5.4 JAXB

Sõnumite struktuur on esitatud XSD formaadis, seega on vaja XSD-skeemi failist genereerida Java klassid, mille välju hiljem kasutada. Kasutades *gradle-jaxb-plugin* tööriista XJB saab genereerida XSD-st Java klasse [34].

3.5.5 Log4j2

Logimiseks kasutatakse Spring Booti toetatavat logimisteeki Log4j2. Seda on mugav rakendada kasutades Lomboki annotatsiooni *@Slf4j*.

3.6 Pideva integreerimise ja tarne tööriistad

Pidev integreerimine (ingl *continuous integration,* CI) tähendab uue programmikoodi sagedast ülespanemist ühisesse koodihoidlasse ning kohest muudatuste valideerimist [11]. CI kasutades saab arendaja koodi üles panna, kui kasvõi väike funktsionaalsus on realiseeritud või muudetud. Toimub automaatne koodi kokkuehitamine (ingl *build*), ühiktestide käivitamine ning tänu sellele saab arendaja kohese tagasiside.

Pidev tarne (ingl *continuous delivery*) on protsesside ahel, mille jooksul viiakse tarkvaraarendamise käigus tehtud koodimuudatused tarkvara tarnimiseni. Tüüpiliselt tehakse seda ilma inimese sekkumiseta ehk automaatselt. [11]

Protsessi, mille käigus tehakse valmis toode kliendile kättesaadavaks, kutsutakse pidevaks juurutamiseks (ingl *continuous deployment*). [11]

Pideva integreerimise ja tarne tarkvara on vajalik, et automatiseerida tarkvaraprojektide ehitust ja paigaldust. Tänu sellele saab keskenduda kvaliteetse tarkvara loomisele ning selle kliendile edastamisele. Kui protsesse automatiseerida, siis saab tarkvara arendamist, testimist ning paigaldamist teha kiiremini ja efektiivsemalt. [11]

3.6.1 Jenkins vs. Bamboo

Ettevõttes on kasutusel kaks pideva integratsiooni tarkvara, nendeks on Jenkins ja Bamboo.

Jenkins on turul olev kõige populaarsem automaatika server. See aitab luua pideva integratsiooni ja tarne ahelaid (ingl *pipeline*). Jenkins on tasuta ja avatud lähtekoodiga tarkvara. [35]

Bamboo seevastu on tasuline tarkvara. Bamboo eeliseks on kasutajasõbralikkus ning klienditoe olemasolu. Bambool on juba sisseehitatud lisad ning võimalus installeerida erinevaid pistikprogramme, kuid nende kasutamine on üldiselt lisatasu eest. Jenkinsil on ainult pluginate kasutamise võimalus, nende valik on suur ja need on avalikult saadaval. [36]

Kuna ettevõtte soov on kasutada rohkem vabavaralist tarkvara, siis kasutatakse antud projektis Jenkinsit.

3.6.2 Docker

Dockeri konteiner on tarkvaraüksus, kuhu pakendatakse kood ja sellega seotud sõltuvused ning mida saab käivitada erinevates keskkondades samamoodi, ilma et peaks keskkondadesse eraldi sõltuvusi installeerima. Dockeri kujutist käivitades luuakse konteiner, kus rakendus töötab. Dockeri kujutis on iseseisvalt käivitatav tarkvarapakett, mis sisaldab kõike, mida on rakenduse jooksutamiseks vaja: rakenduse koodi, käitusaega (ingl *runtime*), süsteemi tööriistu, teeke ja seadeid. [4] Dockeri abil saab jooksutada ühe masina pealt palju rakendusi. Lisaks vähendab see kulusid ja kiirendab kogu protsessi, sest konteinerid võtavad vähe ruumi ning käivituvad kiiresti. [37]

Rakendus pakendatakse Dockeri konteinerisse. Rakenduse koodihoidlasse lisatakse *Dockerfile*, mis on Dockeri kujutise loomiseks mõeldud konfiguratsioonifail.

3.6.3 Kubernetes

Kubernetes on konteinerite haldussüsteem, mis on mõeldud rakenduste juurutamise, skaleerimise ja haldamise automatiseerimiseks [38].

Kuberneteses saab rakendusi lihtsasti skaleerida kasutades kasutajaliidest või seadistada automaatne skaleeruvus vastavalt rakenduse kasutusele. Nii saab määrata, et rakendus

suurema koormuse ajal kasutaks töötamiseks rohkem võimsust. Kui konteiner peaks lõpetama töötamise, siis Kubernetes taaskäivitab selle. [38] Ühte rakendust saab jagada mitme konteineri peale, nii et kui rakenduse ühe konteineriga peaks midagi juhtuma, siis suunatakse liiklus ümber teistele töötavatele konteineritele [37]. Tänu sellele on tagatud rakenduse kättesaadavus ja stabiilsus.

Kasutajanimede, paroolide ja URLide salastatud hoidmise jaoks on kasutusel Kubernetese saladused (ingl *secrets*). Andmed on salvestatud Kubernetese keskkonda ning rakenduse konfiguratsioonifailis on viidatud, millist saladust kasutada.

3.6.4 SonarQube

SonarQube on platvorm, mille abil saab kontrollida koodi kvaliteeti. See on automaatne koodi läbivaatuse (ingl *code review*) tööriist programmivigade, turvaaukude ning lõhnava koodi (ingl *code smell*) tuvastamiseks koodis. Selle abil saab iga haru (ingl *branch*) koodi eraldi analüüsida ning võrrelda põhiharu (ingl *master branch*) koodiga. [39]

Analüüsitulemusi saab vaadata SonarQube kasutajaliidesest ning Developer versiooni puhul ka Bitbucketis *pull request'*i juurest [39]. Lisaks sellele saab SonarQubega koos kasutada programmeerimiskeskkonna laiendust SonarLint, mis aitab leida ja parandada vigu juba koodi kirjutamise ajal [40].

3.6.5 Bitbucket

Rakenduse koodi hoitakse Bitbucketis. Lisaks programmikoodile on projekti juures ka Jenkinsi ja Kubernetese seadistused ja konfiguratsioonifailid.

3.7 Rakenduse ärireeglid

Ettevõtte ja analüütiku poolt on paika pandud teenuse ärireeglid, mida rakendust arendades peab täitma. Töö autor on ärireeglid kokkuvõtvalt kirja pannud ning tõlkinud. Kõigepealt tuuakse kasutavate mõistete seletused, mis pärinevad ettevõtte mõistete süsteemist. Rakenduse ärireeglid koos mõistetega on toodud Lisas 2.

4 Uue rakenduse realisatsioon

Töö käigus luuakse rakendus, mis on kirjutatud Java programmeerimiskeeles ning milles kasutatakse Spring Boot raamistikku ja Gradle projekti ehitamise ja sõltuvuste haldamise tööriista.

Projekti põhi luuakse kasutades Spring Initializer tööriista, mis genereerib esmase rakenduse [41]. Projektis on kasutusel Java 11, Spring Boot 2.2 ja Gradle 6.

4.1 Sõltuvuste lisamine

Faili *build.gradle* lisatakse analüüsi tulemusena väljavalitud tehnoloogiate kasutamiseks vajalikud pluginad, teegid (ingl *dependencies*) ning *taskid*. Kood projekti pluginate lisamisest on toodud Joonis 3.

```
plugins {
    id 'org.springframework.boot' version '2.2.6.RELEASE'
    id 'io.spring.dependency-management' version '1.0.9.RELEASE'
    id "com.jfrog.artifactory" version "4.8.1"
    id 'java'
    id "com.github.jacobono.jaxb" version "1.3.5"
    id "org.sonarqube" version "2.7.1"
}
```

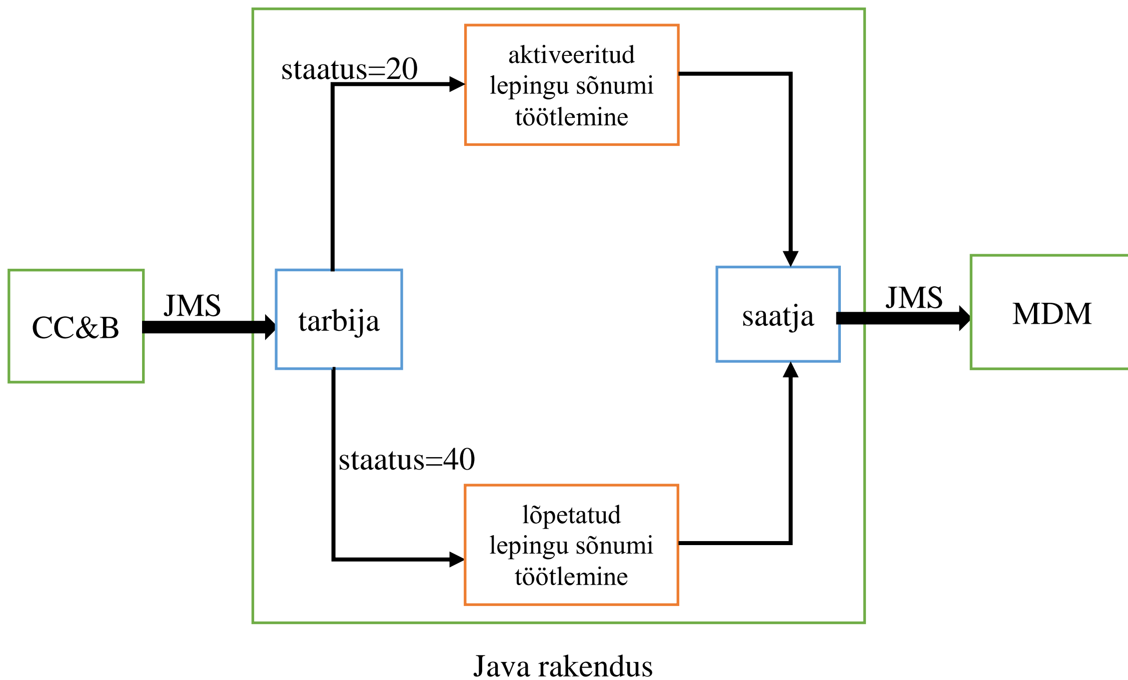
Joonis 3. Pluginate lisamine *build.gradle* failis.

Näiteks koodi läbivaatuse tööriista SonarQube kasutamiseks lisatakse projekti plugin (vt Joonis 3) ning *task* (vt Joonis 4).

```
sonarqube {
    properties {
        property "sonar.projectName", "service-point-sync-service"
        property "sonar.projectKey", "[redacted].servicepointsyncservice"
        property "sonar.host.url", "[redacted]"
        property "sonar.exclusions", [
                "src/generated",
                "src/main/java/[redacted]/servicepointsyncservice/configuration/**",
                "src/main/java/[redacted]/servicepointsyncservice/monitoring/**"
        ]
    }
}
```

Joonis 4. SonarQube seadistamise task.

4.2 Arhitektuur



Joonis 5. Rakenduse arhitektuur lihtsustatud kujul.

Rakenduse eesmärk on vastu võtta CC&B poolt saadetud sõnum, sõnumis olev objekt äriloogika reeglite järgi töödelda uueks objektiks ning saata uus objekt sõnumina edasi MDM poole.

Kui CC&B rakendus saadab sõnumi JMS järjekorda, siis loodav rakendus püüab selle kinni. Selleks on rakenduses tarbija, mis seda järjekorda kuulab. CC&B saadab välja sõnumi *RequestMessage* objekti kujul. Kui CC&B-s toimub lepingu aktiveerimine ehk staatus on 20, siis peab rakendus töötlema objekti aktiveeritud lepingu töötlemise ärireeglite alusel ning kui toimub lepingu lõpetamine ehk staatus on 40, siis lõpetatud lepingu töötlemise tingimuste alusel. Kui sõnum on töödeldud, saadab rakenduses olev saatja selle JMS järjekorda, mida kuulab MDM. MDM võtab sõnumi vastu *SDPSyncMessage* objekti kujul. Joonis 5 on toodud lihtsustatud kujul loodav Java rakendus ja tema suhtlus teiste süsteemidega.

4.2.1 JMS-i seadistamine

JMS järjekorrad loovad administraatorid. Lisaks tavalistele järjekordadele seadistatakse ka vigade järjekorrad (ingl *error queue*), kuhu lähevad sõnumid, mille saatmine on ebaõnnestunud.

Rakendusesiseselt tuleb seadistamiseks lisada konfiguratsioonifaili *application.yaml* JMS järjekordade aadressid ning ligipääsuks vajalikud viidad (vt Joonis 6). URL, kasutajanimi ja parool hoitakse turvaliselt Kuberneteses. Vaikimisi saavad kõik kasutajad JMS ressurssidele WebLogic Serveris ligi, seetõttu peab see kindlasti paroolidega kaitstud olema [42]. JMS-i kasutamiseks lisatakse projekti vajalikud sõltuvused ja JMS-i seadistamiseks luuakse Java klassid.

```yaml
app:
  weblogic:
    properties:
      url: ${JMS_URL}
      security:
        username: ${JMS_USER}
        password: ${JMS_PASSWORD}
    channels:
      serviceAgreementSync:
        enabled: true
        queue: ccb.serviceagreementsync.stream
        concurrency: 1
      mdm:
        servicePointSync:
          queue: mdm.servicepointsync.stream
```

Joonis 6. *application.yaml* JMS järjekordade andmete jaoks.

4.2.2 Sõnumite vastuvõtmine

JMS-põhine sõnumivahetus on oma olemuselt asünkroonne, mis tähendab, et kui rakendusele saadetakse sõnum, siis saatja ei pea ootama, et sõnumi vastuvõtja sõnumi kätte saaks. Kõige lihtsam viis sõnumit asünkroonselt vastu võtta on kasutada Spring raamistiku poolt pakutavat annotatsiooni *@JmsListener* [28]. See annotatsioon lubab järjekorras olevaid sõnumeid kuulata. Järjekorra nimi on määratud elemendiga *destination*. Kui järjekorda tuleb sõnum, siis sõnumitarbija tarbib sõnumi sealt järjekorrast ja *receive* meetod kutsutakse välja.

Annotatsiooni *@JmsListener* elemendiga *containerFactory* määratakse sõnumikuulajate konteinerite (ingl *message listener container*) loomiseks *serviceAgreementSyncJmsListenerContainerFactory* [43].

@JmsListener annotatsiooni kasutamine koos elementidega *destination* ja *containerFactory* on toodud Joonis 7. *@JmsListener* annotatsiooni kasutamiseks tuleb lisada annotatsioon *@EnableJms* konfiguratsiooniklassi [28].

```java
@JmsListener(destination = "${app.weblogic.channels.serviceAgreementSync.queue}",
        containerFactory = "serviceAgreementSyncJmsListenerContainerFactory")
public void receive(@Payload RequestMessage requestMessage) {...}
```

Joonis 7. Sõnumite kuulamine JMS järjekorrast.

4.2.3 Sõnumite saatmine

Springi klass *JmsTemplate* sisaldab erinevaid meetodeid sõnumite saatmiseks. Osa saatmismeetodeid määrab sihtkoha *Destination* objekti abil ning teised kasutades *String*'i. Antud juhul kasutatakse meetodit *convertAndSend()*, mille puhul järjekorra asukoht on määratud *String* kujul. See meetod kasutab sõnumi sisu jaoks Java objekti ning Java objekti teisendamise teeb ära *MessageConverter* liides (ingl *interface*). See liides defineerib konverteerimise Java objekti ja JMS sõnumi vahel. Vaikimisi toimub teisendamine *String* ja *TextMessage*, *byte[]* ja *BytesMessage*, *Map* ja *MapMessage* vahel. [28] Kuna saadetav objekt ei ole neist ükski, siis konfiguratsiooniklassis defineeritakse objekti teisendamine kasutades Spring raamistiku klasse *Jaxb2Marshaller* ja *MarshallingMessageConverter* (vt Joonis 8).

```java
@Bean
public MarshallingMessageConverter sdpSyncMessageConverter(Jaxb2Marshaller sdpSyncJaxb2Marshaller) {
    MarshallingMessageConverter converter = new MarshallingMessageConverter();
    converter.setTargetType(MessageType.TEXT);
    converter.setMarshaller(sdpSyncJaxb2Marshaller);
    converter.setUnmarshaller(sdpSyncJaxb2Marshaller);
    return converter;
}

@Bean
public Jaxb2Marshaller sdpSyncJaxb2Marshaller() {
    Jaxb2Marshaller jaxb2Marshaller = createDefaultJaxb2Marshaller();
    jaxb2Marshaller.setClassesToBeBound(SDPSyncMessage.class);
    return jaxb2Marshaller;
}
```

Joonis 8. *SDPSyncMessage* objekti teisendamine.

Sõnumite saatmine JMS järjekorda toimub *Producer* klassis, mis on toodud Joonis 9.

```java
@Slf4j
@Component
public class ServicePointSyncProducer {
    private final JmsTemplate sdpSyncJmsTemplate;
    private final String destinationName;

    public ServicePointSyncProducer(JmsTemplate sdpSyncJmsTemplate,
                                    @Value("${app.weblogic.channels.mdm.servicePointSync.queue}") String destinationName) {
        this.sdpSyncJmsTemplate = sdpSyncJmsTemplate;
        this.destinationName = destinationName;
    }

    public void sendMessage(SDPSyncMessage sdpSyncMessage) {
        log.info("Sending service point sync message to " + destinationName + ": message id {}",
                sdpSyncMessage.getHeader().getMessageID());
        sdpSyncJmsTemplate.convertAndSend(destinationName, sdpSyncMessage);
    }
}
```

Joonis 9. *Producer* klass.

4.2.4 DTO

Sissetulevad andmed salvestatakse ja hoitakse DTO (*Data Transfer Object*) objektina. DTO objekte kasutatakse, et tagada andmete turvalisus ja õigsus ning et kasutataks ainult vajaminevaid andmeid. Selleks tuleb luua kõigepealt *ServiceAgreementDTO* objekt (vt Joonis 10).

```java
@Getter
@Builder
public class ServiceAgreementDTO {
    private final String serviceAgreementId;
    private final String saIdLevelTwo;
    private final String eic;
    private final CCBStatus servicePointStatus;
    private final String serviceAgreementStartDate;
    private final String serviceAgreementEndDate;
    private final String servicePointType;
    private final String contFuseAmp;
    private final String contPhasesVoltage;
    private final String producer;
    private final String microProducer;
    private final String consumptionGraphType;
    private final String lossReactiveEnergy;
    private final String lossActiveEnergy;
    private final String contSharedFuse;
    private final String contMaxLoad;
    private final String conMultItem;
    private final String settlementMethod;
}
```

Joonis 10. *ServiceAgreementDTO* objekt.

RequestMessage objekti vajaminevad väljad konverteeritakse *ServiceAgreementDTO* objektiks kasutades Lomboki Builderi funktsionaalsust (vt Joonis 11).

```java
@Component
public class RequestMessageToServiceAgreementDTOConverter implements Converter<RequestMessage, ServiceAgreementDTO> {

    @Override
    public ServiceAgreementDTO convert(RequestMessage source) {
        ServiceAgreementDTO.ServiceAgreementDTOBuilder serviceAgreementBuilder = ServiceAgreementDTO.builder();
        SaspInfoDetails saspInfoDetails = source.getBody().getSaspInfoDetails().get(0);
        SpInfoDetails spInfoDetails = saspInfoDetails.getSpInfoDetails();
        SaInfoDetails saInfoDetails = saspInfoDetails.getSaInfoDetails();

        serviceAgreementBuilder
                .saIdLevelTwo(saspInfoDetails.getSaIdLevelTwo())
                .serviceAgreementId(saInfoDetails.getSaId())
                .eic(spInfoDetails.getEic())
                .servicePointStatus(CCBStatus.fromValue(saInfoDetails.getStatus()))
                .serviceAgreementStartDate(saInfoDetails.getStartDate())
                .serviceAgreementEndDate(saInfoDetails.getEndDate())
                .servicePointType(spInfoDetails.getSpType())
                .contFuseAmp(spInfoDetails.getContFuseAmp())
                .contPhasesVoltage(spInfoDetails.getContPhasesVoltage())
                .producer(spInfoDetails.getProducer())
                .microProducer(spInfoDetails.getMicroproducer())
                .consumptionGraphType(spInfoDetails.getConsumptionGraphType())
                .lossActiveEnergy(spInfoDetails.getLossActiveEnergy())
                .lossReactiveEnergy(spInfoDetails.getLossReactiveEnergy())
                .contSharedFuse(spInfoDetails.getContSharedFuse())
                .contMaxLoad(spInfoDetails.getContMaxLoad())
                .conMultItem(spInfoDetails.getConMultItem())
                .settlementMethod(spInfoDetails.getSettlementMethod());

        return serviceAgreementBuilder.build();
    }
}
```

Joonis 11. *RequestMessageToServiceAgreementDTOConverter* klass.

4.2.5 Sündmused

Rakenduses kasutatakse sündmusi (ingl *event*), et klasside omavahelist sidusust vähendada. Sündmuste kasutamine muudab rakenduse arendamise paindlikumaks kui meetodi otse väljakutsumine. Sel juhul saab sündmuste kuulajaid lihtsalt lisada ning kustutada, siis kui selleks vajadus tekib. Samal sündmusel saab olla ka mitu kuulajat. [44]

Kui CC&B saadetud sõnumis on staatuse väli väärtusega 20 ehk aktiivne, siis avaldatakse *ServiceAgreementServicePointActivated* sündmus ning kui 40 ehk peatunud, siis *ServiceAgreementServicePointStopped* sündmus. Sündmuse avaldamiseks kasutatakse Springi liidest *ApplicationEventPublisher* ja selle meetodit *publishEvent*. Sündmuste avaldamise kood on toodud Joonis 12.

```java
if (CCBStatus.ACTIVE  == serviceAgreementDTO.getServicePointStatus()) {
    applicationEventPublisher.publishEvent(new ServiceAgreementServicePointActivated( source: this, serviceAgreementDTO));
}
if (CCBStatus.STOPPED == serviceAgreementDTO.getServicePointStatus()) {
    applicationEventPublisher.publishEvent(new ServiceAgreementServicePointStopped( source: this, serviceAgreementDTO));
}
```

Joonis 12. Sündmuste avaldamine.

ServiceAgreementServicePointActivated on sündmuse klass, mis on mõeldud sündmuse andmete hoidmiseks (vt Joonis 13).

```java
@Getter
public class ServiceAgreementServicePointActivated extends ApplicationEvent {
    private final transient ServiceAgreementDTO serviceAgreementDTO;

    public ServiceAgreementServicePointActivated(Object source, ServiceAgreementDTO serviceAgreementDTO) {
        super(source);
        this.serviceAgreementDTO = serviceAgreementDTO;
    }
}
```

Joonis 13. Sündmuse klass.

Sündmusel saab olla mitu kuulajat, millel on erinevad ülesanded. Sündmuse kuulamiseks on kaks võimalust, kas kasutada annotatsiooni *@EventListener* või implementeerida *ApplicationListener* liidest. [44] Antud juhul kasutatakse esimest varianti. Sündmuse kuulamise kood on toodud Joonis 14.

Kui avaldatakse *ServiceAgreementServicePointActivated* sündmus, siis antud sündmuse kuulaja püüab sündmuse kinni ning *onEvent* meetod kutsutakse välja. Meetodi *onEvent* sees toimub *Service* klassi kaudu äriloogika realiseerimine ning seejärel *Producer* klassi *sendMessage* meetodi välja kutsumine.

```java
@Component
@RequiredArgsConstructor
public class ServiceAgreementServicePointActivatedHandler {
    private final ActivatedServiceAgreementServicePointRelationshipSyncService activatedServiceAgreementServicePointRelationshipSyncService;
    private final ServicePointSyncProducer servicePointSyncProducer;

    @EventListener
    public void onEvent(ServiceAgreementServicePointActivated event) throws DatatypeConfigurationException {
        SDPSyncMessage saSpRelationshipSyncMessage = activatedServiceAgreementServicePointRelationshipSyncService
                .createServiceAgreementServicePointRelationshipSyncMessage(event.getServiceAgreementDTO());
        servicePointSyncProducer.sendMessage(saSpRelationshipSyncMessage);
    }
}
```

Joonis 14. Sündmuse kuulaja.

4.2.6 *Service* klass

Enne sõnumi MDM poole saatmist töödeldakse sisend *Service* klassis. *Service* klassis asub äriloogika realisatsioon. *Service* klassi sisendiks on *ServiceAgreementDTO* objekt, mille väärtuste põhjal luuakse *SDPSyncMessage* objekt.

Kuna lepingute aktiveerimise ja lõpetamise loogika on erinev, siis luuakse kaks erinevat *Service* klassi. Mõlema klassi ülemklassiks saab abstraktne klass, milles on mõlema alamklassi ühised meetodid.

Eraldi *Service* klass luuakse *High Limit* mõõdiku arvutamise jaoks, sest tegemist on keeruliste spetsiifiliste arvutustega, mida on targem koodi loetavuse mõttes peamisest *Service* klassist eraldi hoida. Arvutatava *High Limit*'i väärtuseks on maksimaalne võimsus, mida peakaitse lubab pikaajaliselt tarbida.

Service klassides kasutatakse abiklassi, kus on erinevad kuupäevade, kellaaegade ja ajatsoonidega seotud teisendused. Joonis 15 on toodud näide meetodist, kus mõõtmistulemused seotakse teenuspunkti, kanali ja parameetriga. Selles meetodis kasutatakse nii *High Limit* mõõdiku arvutamist kui ka abiklassi kuupäeva arvutamiseks.

```
private MeasReferenceValue createMeasReferenceValue(ServiceAgreementDTO serviceAgreementDTO, String channelNumber, String channelType) throws Datat
    MeasReferenceValue measReferenceValue = new MeasReferenceValue();
    measReferenceValue.setServicePointId(createServicePointId(serviceAgreementDTO.getEic()));
    measReferenceValue.setChannelNum(channelNumber);
    measReferenceValue.setHighLimit(highLimitCalculationService.calculateHighLimitValue(serviceAgreementDTO).doubleValue());
    measReferenceValue.setStartDate(localDateTimeUtils.convertDateTimeToXmlGregorianCalendar(DefaultDate.MEAS_REFERENCE_VALUE_DEFAULT_START_DATE));
    measReferenceValue.setChannelType(channelType);

    return measReferenceValue;
}
```

Joonis 15. Meetod mõõtmistulemuste sidumiseks teenuspunkti, kanali ja parameetriga.

Võrreldes Oracle SOA-ga pakub Java keel lihtsaid võimalusi korduvate koodiridade haldamiseks. Kui Java arendajale on elementaarne, et mitmeid kordi kasutatav koodiplokk kirjutatakse eraldi meetodisse ning kasutatakse selle korduvat väljakutsumist, siis Oracle SOA puhul lihtne võimekus sellise olukorra jaoks puudub. Joonis 16 on näide kanali lisamise meetodist, mida kasutatakse 9 korda erinevat tüüpi kanalite lisamiseks. Kõikides SOA teenustes on kanalid aga ükshaaval välja kirjutatud ja seetõttu esineb väga palju koodikorduseid.

```
private VirtualChannel createVirtualChannel(ServiceAgreementDTO serviceAgreement, String channelType,
                                            String contributorChannelType, String formula) throws DatatypeConfig
    VirtualChannel virtualChannel = createVirtualChannel(channelType, eic: "");
    virtualChannel.setServicePointId(createServicePointId(serviceAgreement.getEic()));
    virtualChannel.getVcFormula().add(createFormula(formula, VIRTUAL_CHANNEL_START_DATE));
    virtualChannel.getVcContributor().add(createContributor
            (serviceAgreement.getEic(), contributorChannelType, CHILD_SYMBOL_MEAS, VIRTUAL_CHANNEL_START_DATE));

    return virtualChannel;
}
```

Joonis 16. Meetod virtuaalse kanali lisamiseks.

4.3 Testimine

Ühiktestimine on üksikprogrammide või -komponentide testimine selleks, et veenduda, et neis pole analüüsi- või programmeerimisvigu [45]. Eesmärk on kontrollida, et koodi iga üksus toimib ootuspäraselt. Üksuseks võib olla üksikfunktsioon, meetod, protseduur, moodul või objekt. Ühiktestimine on oluline, sest tänu sellele avastatakse vigu juba arendustsükli alguses ja nende kohene parandamine säästab hilisemaid kulusid. [46]

Ühiktestimiseks kasutatakse JUnit Jupiter ja Mockito tööriista. Mockito võimalustest kasutatakse näiteks *spy* liidest, mis võimaldab kutsuda välja kõiki objekti tavapäraseid meetodeid, kuid sealjuures saab neid kontrollida ja juhtida ning *doReturn* liidest, mille abil saab täpsustada, kuidas meetod peaks käituma [47]. Joonis 17 on toodud näide *Service* klassi ühe funktsionaalsuse testimisest.

```
20  class ActivatedServiceAgreementServicePointRelationshipSyncServiceTest {
21      private static final LocalDateTimeUtils localDateTimeUtils = spy(new LocalDateTimeUtils());
22      private static final HighLimitCalculationService highLimitCalculationService =
23              spy(new HighLimitCalculationService( defaultAmpere: "25",  highLimitTolerance: "1.1", localDateTimeUtils));
24      private static final ActivatedServiceAgreementServicePointRelationshipSyncService activatedServiceAgreementServicePointRelationshipSyncService =
25              new ActivatedServiceAgreementServicePointRelationshipSyncService(localDateTimeUtils, highLimitCalculationService);
26      private static ServiceAgreementDTO serviceAgreementDTO;
27
28      @BeforeEach
29      void init() throws DatatypeConfigurationException {
30          serviceAgreementDTO = ServiceAgreementDTO
31                  .builder()
32                  .serviceAgreementId("0068024350")
33                  .serviceAgreementStartDate("2019-04-01")
34                  .serviceAgreementEndDate("2019-04-09")
35                  .servicePointStatus(CCBStatus.ACTIVE)
36                  .eic("38ZEE-00422648-O")
37                  .build();
38
39          doReturn(DatatypeFactory.newInstance().newXMLGregorianCalendar( lexicalRepresentation: "2019-08-31T21:00:00.000Z"))
40                  .when(localDateTimeUtils).getCurrentDateTime();
41      }
42
43      @Test
44      void createSaSpRelationshipSyncMessage_setContPhasesVoltage_createServicePointParametersPHASESandVOLTAGE() throws DatatypeConfigurationException {
45          ReflectionTestUtils.setField(serviceAgreementDTO,  name: "contPhasesVoltage",  value: "3X380V");
46          Parameter parameterPhases = activatedServiceAgreementServicePointRelationshipSyncService
47                  .createServiceAgreementServicePointRelationshipSyncMessage(serviceAgreementDTO).getPayload().getServicePoint().get(0).getParameter().get(0);
48          Parameter parameterVoltage = activatedServiceAgreementServicePointRelationshipSyncService
49                  .createServiceAgreementServicePointRelationshipSyncMessage(serviceAgreementDTO).getPayload().getServicePoint().get(0).getParameter().get(1);
50          assertAll(
51                  () -> assertNotNull(parameterPhases,  message: "ServicePoint parameter PHASES object should not be null"),
52                  () -> assertEquals( expected: "PHASES", parameterPhases.getName(),  message: "Parameter PHASES name should be PHASES"),
53                  () -> assertEquals( expected: "2019-04-01T00:00:00.000+03:00", parameterPhases.getStartDate().toString(),
54                           message: "Parameter PHASES startDate should be 2019-04-01T00:00:00.000+03:00"),
55                  () -> assertEquals( expected: "3", parameterPhases.getValue(),  message: "Parameter PHASES value should be 3"),
56                  () -> assertNotNull(parameterVoltage,  message: "ServicePoint parameter VOLTAGE object should not be null"),
57                  () -> assertEquals( expected: "VOLTAGE", parameterVoltage.getName(),  message: "Parameter VOLTAGE name should be VOLTAGE"),
58                  () -> assertEquals( expected: "2019-04-01T00:00:00.000+03:00", parameterVoltage.getStartDate().toString(),
59                           message: "Parameter VOLTAGE startDate should be 2019-04-01T00:00:00.000+03:00"),
60                  () -> assertEquals( expected: "0.38", parameterVoltage.getValue(),  message: "Parameter VOLTAGE value should be 0.38")
61          );
62      }
```

Joonis 17. Näide ühiktestimisest.

Kogu voo (ingl *flow*) läbi testimiseks tuleb CC&B poolt sõnum teele saata ehk leping kas aktiveerida või peatada. CC&B salvestab väljuvad sõnumid andmebaasi, kust saab võtta näidissõnumi testimiseks. Kui CC&B poolt saadetava sõnumi näide on olemas, siis kõige lihtsam viis testimiseks on saata sõnum manuaalselt JMS järjekorda. Lokaalseks

rakenduse testimiseks on võimalik rakendus käivitada Dockeri abil ning luua lokaalne JMS Server. Kui JMS järjekorrad on juba seadistatud, siis võib kasutada näiteks JMSToolBox tööriista. Lisaks tuleb kontrollida, et korrektsed andmed jõuaksid MDM süsteemi. Käsitsi kogu voo testimisel peab kõik juhtumid eraldi läbi testima.

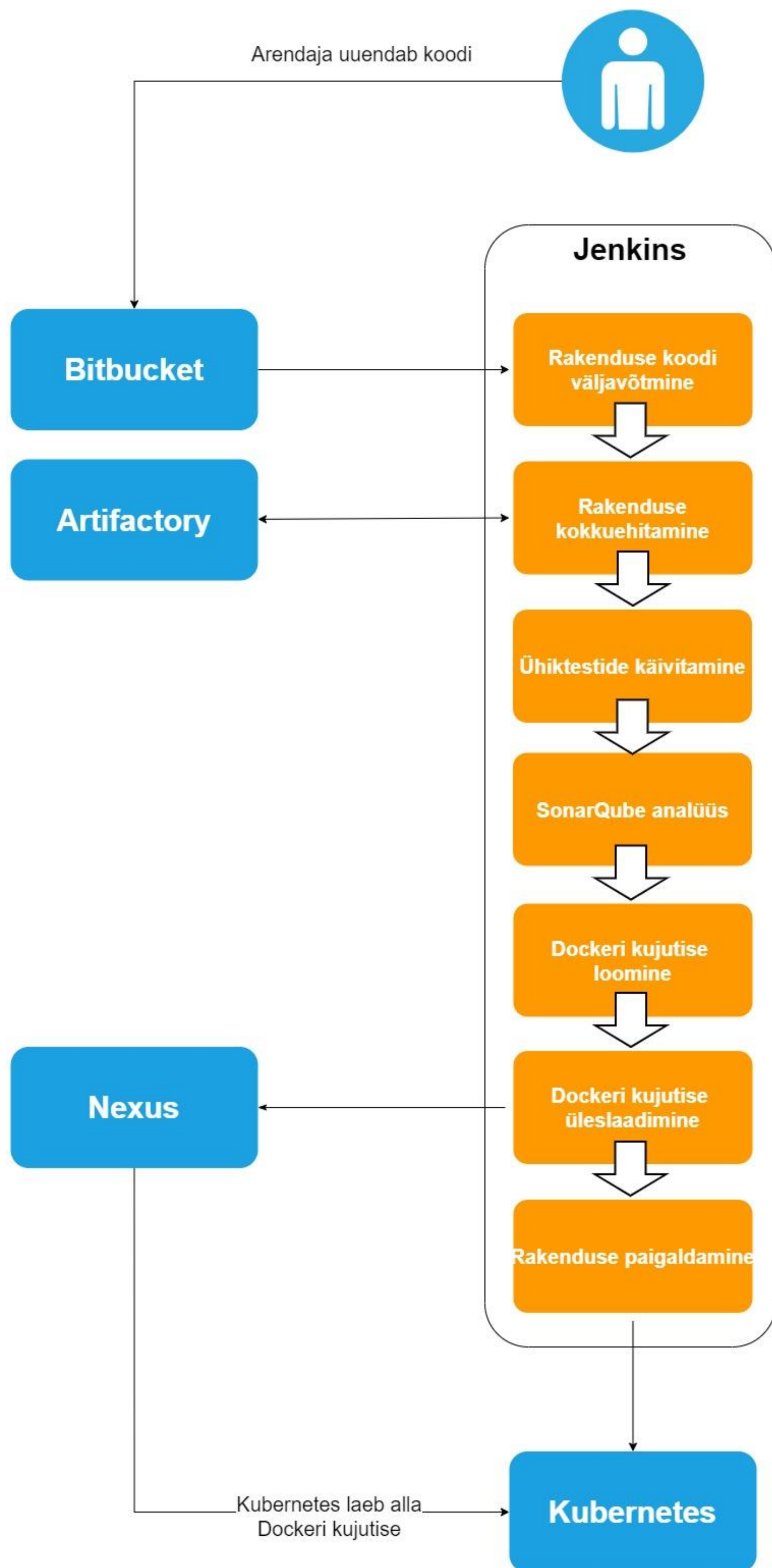
4.4 Ehitus- ja paigaldusprotsess

CI tööriistana kasutatakse Jenksinsit. Jenkinsi kasutajaliidese kaudu lisatakse uus *multibranch pipeline*, kus seotakse tarneahel (ingl *pipeline*) Bitbucketi projektiga. Jenkinsi tarneahela saab luua kasutajaliidese kaudu või skriptiga. Antud juhul kasutatakse skripti ehk kirjutatakse valmis *Jenkinsfile*, mis lisatakse rakenduse juurde. Tarneahel on kui sündmuste või ülesannete rida. [48]

Järgnevalt kirjeldatakse ehitus- ja paigaldusprotsessi, mis on graafiliselt näha Joonis 18. Kui arendaja paneb koodi Bitbucketisse üles, siis Jenkins reageerib sellele ning võtab koodi (ingl *check out*), ehitab selle Gradlet kasutades kokku ning jooksutab ühiktestid. Koodi kokkuehitamisel kasutab Jenkins teeke Artifactoryst. Kui ühiktestidest mõni ebaõnnestub, siis kuvatakse viga ja paigalduspotsess peatatakse.

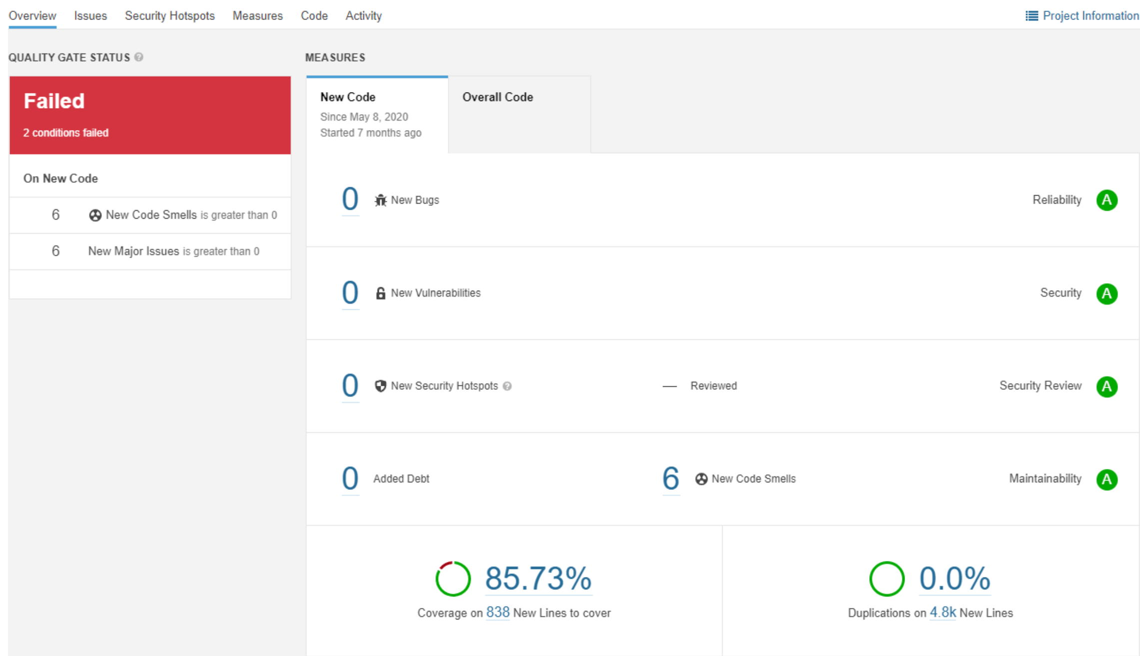
Uut koodi analüüsitakse SonarQubega ehk tehakse kvaliteedikontroll. Seejärel luuakse Dockeri kujutis ja salvestatakse see Nexusesse. Viimasena toimub rakenduse paigaldamine Kubernetese keskkonda. Vaikimisi kasutatakse paigaldamiseks loodud Dockeri kujutist, aga vajaduse tekkimisel on võimalik ka varasemat versiooni kasutada.

Tarkvara arendusprotsessis on kasutusel kolm keskkonda, nendeks on *dev*, *test* ja *live*. *Dev* keskkond on mõeldud eelkõige arendajatele ja esmaseks testimiseks, *test* keskkond uute arenduste demonstreerimiseks ja äripoolseks testimiseks ning *live* lõppkasutajatele kasutamiseks. Jenkinsi kasutajaliidesest saab mugavalt paigaldada rakendusi kõikidesse keskkondadesse.



Joonis 18. Ehitus- ja paigaldusprotsess.

SonarQube detailseid analüüsitulemusi saab vaadata SonarQube kasutajaliidesest (vt Joonis 19).



Joonis 19. Analüüsitulemused SonarQube kasutajaliideses.

5 Tulemus ja järeldused

Töö tulemusena on valminud Java Spring Boot rakendus koos testide ning paigalduse tarneahelaga, mis võetakse kasutusele 2021. aasta alguses. Töö kirjutamise lõpu seisuga on rakendus läbinud testimise faasi ja valmis kasutamiseks.

Loodud rakendus on võrreldes varasema lahendusega iseseisvalt paigaldatav ning hallatav. Rakendus on täielikult ühe tiimi käes, kes vastutab rakenduse töötamise eest. Tänu sellele, et suhtluseks kasutatakse JMS järjekordi, ei sõltu rakendus otseselt teistest rakendustest. Arendamiseks kasutatav programmeerimiskeel Java ning raamistik Spring Boot on laialt kasutatav ning spetsialiste on lihtne leida. Monitooringu platvormi kasutamine realiseeriti diplomitöö väliselt ning töös seda detailselt ei kajastata.

Antud teenuse viimine Oracle SOA platvormilt Java Spring tehnoloogiale on algus üleüldisele Oracle SOA väljavahetamisele organisatsioonis. Autori arvates tuleks edaspidi eelkõige jälgida, et uusi funktsionaalsusi ei arendata mitte Oracle SOA platvormil, vaid võetaks aina rohkem kasutusele kaasaegsed tehnoloogiad. Tänu uutele tehnoloogiatele on seda aina lihtsam teha, sest Jenkinsi, Kubernetese ja Dockeri kasutamine lihtsustavad oluliselt paigaldusi ja nende automatiseerimist, raamistikud pakuvad erinevaid lahendusi veahalduseks, logimine ja monitooring on lihtsasti seadistatavad ka arendajate, mitte ainult administraatorite jaoks. Varasemaga võrreldes on rakenduses muudatuste tegemine lihtne ja kiire, sest uus rakendus on ühiktestidega kaetud ning paigaldusprotsess on automaatne.

Oracle SOA täielik väljavahetamine tooks ettevõttele ka suurt finantsilist kasu, sest Oraclele makstavad litsentsitasud on kõrged [49]. Seetõttu oleks mõistlik ka antud rakenduses ning teistes JMS-i kasutatavates rakendustes välja vahetada Oracle WebLogic JMS mõne tasuta alternatiivi vastu.

6 Kokkuvõte

Diplomitöö raames viidi Oracle SOA tehnoloogial põhinev võrgulepingu ja teenuspunkti vahelise seose edastamise teenus üle Java Spring tehnoloogiale.

Kõigepealt kirjeldati olemasolevat Oracle SOA teenust, toodi välja selle probleemid ning head omadused. Analüüsi käigus leiti sobivad tehnoloogiad Oracle SOA teenuse asendamiseks, valiti arendamise stiil, programmeerimiskeel, raamistik, rakenduste vaheliseks suhtluseks sobivad vahendid ning pideva integreerimise ja tarne tööriistad. Lisaks toodi välja ärireeglid, mida rakendus peab täitma.

Diplomitöö tulemusena loodi Java Spring Boot raamistikuga mikroteenus, mis sõnumivahetuseks teiste rakendustega kasutab JMS järjekordi. Kui rakenduses CC&B uuendatakse kliendi ja teenuspunktiga seotud informatsiooni, siis saadetakse sõnum JMS järjekorda, mida loodud rakendus kuulab. Rakendus võtab sõnumi vastu, töötleb sisendit vastavalt ärireeglitele, loob uue sõnumi ning saadab selle edasi. Selle tulemusena jõuavad muutunud andmed mõõteandmete haldamise süsteemi MDM. Teenus on vajalik selleks, et klientidele saadetud arvetel oleksid korrektsed andmed. Rakenduse ootuspärase käitumise kontrolliks kasutati ühiktestimist ning lisaks ka kogu voo käsitsi läbitestimist.

Rakenduse paigaldamiseks loodi tarneahel tänu millele saab rakendust automaatselt paigaldada ning muudatusi kiiresti kliendile kättesaadavaks teha.

Kasutatud kirjandus

[1] MTÜ Eesti Siseaudiitorite Ühing, „5 soovitust agiilse lähenemise rakendamiseks," MTÜ Eesti Siseaudiitorite Ühing, 9 6 2020. [Võrgumaterjal]. Available: https://www.siseaudit.ee/artiklid/5-soovitust-agiilse-lahenemise-rakendamiseks. [Kasutatud 2 12 2020].

[2] „Rakendusliides," EUCIP, [Võrgumaterjal]. Available: https://eopearhiiv.edu.ee/e-kursused/eucip/haldus/213_rakendusliides.html. [Kasutatud 2 12 2020].

[3] Oracle, „Oracle Utilities Customer Care," Oracle, 2014. [Võrgumaterjal]. Available: http://www.oracle.com/us/industries/utilities/utilities-customer-care-billing-ds-2281135.pdf. [Kasutatud 19 11 2020].

[4] Docker Inc, „What is a Container?," Docker Inc, [Võrgumaterjal]. Available: https://www.docker.com/resources/what-container. [Kasutatud 3 12 2020].

[5] M. Fowler, „Data Transfer Object," [Võrgumaterjal]. Available: https://martinfowler.com/eaaCatalog/dataTransferObject.html. [Kasutatud 1 12 2020].

[6] „Java Introduction," [Võrgumaterjal]. Available: https://www.w3schools.com/java/java_intro.asp. [Kasutatud 18 11 2020].

[7] „Java Code Geeks - JMS Tutorials," Exelixis Media P.C., [Võrgumaterjal]. Available: https://www.javacodegeeks.com/jms-tutorials. [Kasutatud 18 11 2020].

[8] E. S. K. S. Michael Hofmann, Microservices Best Practices for Java, IBM Redbooks, 2016.

[9] Siemens, „EnergyIP Meter Data Management (MDM)," Siemens, [Võrgumaterjal]. Available: https://new.siemens.com/global/en/products/energy/energy-automation-and-smart-grid/energyip-meter-data-management.html. [Kasutatud 14 11 2020].

[10] I. Cloud, „SOA vs. Microservices: What's the Difference?," 2 9 2020. [Võrgumaterjal]. Available: https://www.ibm.com/cloud/blog/soa-vs-microservices. [Kasutatud 22 11 2020].

[11] B. Laster, Continuous Integration vs. Continuous Delivery vs. Continuous Deployment, 2nd Edition, O'Reilly Media, Inc, 2020.

[12] Termeki, „E-õppe termineid/ E-learning terms," [Võrgumaterjal]. Available: https://term.eki.ee/termbase/view/9546447/et/et/?initial=P#/concept/view/1321143797/. [Kasutatud 29 11 2020].

[13] A. Hayes, „Scalability," [Võrgumaterjal]. Available: https://www.investopedia.com/terms/s/scalability.asp. [Kasutatud 24 11 2020].

[14] Gradle Inc, „Build Script Basics," Gradle Inc, [Võrgumaterjal]. Available: https://docs.gradle.org/current/userguide/tutorial_using_tasks.html. [Kasutatud 4 12 2020].

[15] „what is the difference between XSD and WSDL," Stack Overflow, [Võrgumaterjal]. Available: https://stackoverflow.com/questions/1952015/what-is-the-difference-between-xsd-and-wsdl. [Kasutatud 3 12 2020].

[16] O. Corporation, Oracle Fusion Middleware Understanding Oracle SOA Suite, 12c (12.1.3), 2015.

[17] E. Hoch, „The Dos and Don'ts of Oracle SOA," 7 10 2014. [Võrgumaterjal]. Available: https://www.zirous.com/2014/10/07/the-dos-and-donts-of-oracle-soa/. [Kasutatud 12 11 2020].

[18] K. Galbraith, „3 methods for microservice communication," 27 8 2019. [Võrgumaterjal]. Available: https://blog.logrocket.com/methods-for-microservice-communication/. [Kasutatud 27 11 2020].

[19] K. Chandrakant, „Why Choose Spring as Your Java Framework?," 13 12 2019. [Võrgumaterjal]. Available: https://www.baeldung.com/spring-why-to-choose. [Kasutatud 18 11 2020].

[20] baeldung, „A Comparison Between Spring and Spring Boot," [Võrgumaterjal]. Available: https://www.baeldung.com/spring-vs-spring-boot. [Kasutatud 18 11 2020].

[21] „Stack Overflow tags," Stack Exchange Inc, [Võrgumaterjal]. Available: https://stackoverflow.com/tags. [Kasutatud 18 11 2020].

[22] S. Prasad, „Event-Driven Microservices with Spring Boot and ActiveMQ," 8 8 2019. [Võrgumaterjal]. Available: https://itnext.io/event-driven-microservices-with-spring-boot-and-activemq-5ef709928482. [Kasutatud 22 11 2020].

[23] „What is Web Service," javatpoint, [Võrgumaterjal]. Available: https://www.javatpoint.com/what-is-web-service. [Kasutatud 26 11 2020].

[24] T.Tammet, „Veebiteenuste võlu ja valu," 22 9 2009. [Võrgumaterjal]. Available: http://lambda.ee/wiki/Veebiteenuste_v%C3%B5lu_ja_valu. [Kasutatud 26 11 2020].

[25] M. Matłoka, „How to communicate your Microservices?," 17 10 2019. [Võrgumaterjal]. Available: https://blog.softwaremill.com/how-to-communicate-your-microservices-6542cb4f98c7. [Kasutatud 26 11 2020].

[26] „SOAP Vs. REST: Difference between Web API Services," Quru99, [Võrgumaterjal]. Available: https://www.guru99.com/comparison-between-web-services.html. [Kasutatud 27 11 2020].

[27] M. Joliveau, „Microservices communications. Why you should switch to message queues.," 23 2 2018. [Võrgumaterjal]. Available: https://dev.to/matteojoliveau/microservices-communications-why-you-should-switch-to-message-queues--48ia. [Kasutatud 26 11 2020].

[28] Pivotal, Inc, „Integration - JMS (Java Message Service)," Pivotal, Inc, [Võrgumaterjal]. Available: https://docs.spring.io/spring-framework/docs/current/reference/html/integration.html#jms. [Kasutatud 19 11 2020].

[29] D. Marx, „Common Java Object Functionality with Project Lombok," 19 09 2010. [Võrgumaterjal]. Available: https://www.infoworld.com/article/2073566/common-java-object-functionality-with-project-lombok.html. [Kasutatud 17 11 2020].

[30] T. P. L. Authors, „Project Lombok," [Võrgumaterjal]. Available: https://projectlombok.org/. [Kasutatud 17 11 2020].

[31] JFrog, „JFrog Artifactory," JFrog Ltd, [Võrgumaterjal]. Available: https://jfrog.com/artifactory/. [Kasutatud 15 11 2020].

[32] S. Inc, „nexus repository pro," Sonatype Inc, [Võrgumaterjal]. Available: https://www.sonatype.com/nexus/repository-pro. [Kasutatud 15 11 2020].

[33] Oracle, „Understanding WebLogic JMS,“ [Võrgumaterjal]. Available: https://docs.oracle.com/en/middleware/fusion-middleware/weblogic-server/12.2.1.4/jmspg/overview.html. [Kasutatud 22 11 2020].

[34] D. Mijares, „jacobono / gradle-jaxb-plugin,“ [Võrgumaterjal]. Available: https://github.com/jacobono/gradle-jaxb-plugin. [Kasutatud 24 11 2020].

[35] R. Leszko, Continuous Delivery with Docker and Jenkins, Birmingham, UK: Packt Publishing, 2017.

[36] D. FIDELIS, „Bamboo vs Jenkins: Which CI/CD tool to use?,“ Atlassian Administration, 14 2 2019. [Võrgumaterjal]. Available: https://valiantys.com/en/blog/atlassian-administration/jenkins-vs-bamboo/. [Kasutatud 29 11 2020].

[37] K. Viiburg, „Docker – mis asi see on ja miks kõik seda kasutavad?,“ 6 7 2018. [Võrgumaterjal]. Available: https://blog.twn.ee/et/docker-mis-asi-see-on-ja-miks-koik-seda-kasutatavad. [Kasutatud 30 11 2020].

[38] Kubernetes, „Production-Grade Container Orchestration,“ Kubernetes, [Võrgumaterjal]. Available: https://kubernetes.io/. [Kasutatud 30 11 2020].

[39] SonarQube, „SonarQube Documentation,“ SonarSource S.A, [Võrgumaterjal]. Available: https://docs.sonarqube.org/latest/. [Kasutatud 24 11 2020].

[40] SonarSource S.A, „SonarLint - Fix issues before they exist,“ SonarSource S.A, [Võrgumaterjal]. Available: https://www.sonarlint.org/. [Kasutatud 24 11 2020].

[41] „Spring Initializr,“ VMware, Inc., [Võrgumaterjal]. Available: https://start.spring.io/. [Kasutatud 11 15 2020].

[42] Oracle, „Developing JMS Applications for Oracle WebLogic Server - Understanding WebLogic JMS Security,“ Oracle, [Võrgumaterjal]. Available: https://docs.oracle.com/en/middleware/fusion-middleware/weblogic-server/12.2.1.4/jmspg/security.html. [Kasutatud 16 11 2020].

[43] Spring Framework, „Annotation Type JmsListener,“ [Võrgumaterjal]. Available: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/jms/annotation/JmsListener.html. [Kasutatud 19 11 2020].

[44] reflectoring.io, „Spring Boot Application Events Explained,“ [Võrgumaterjal]. Available: https://reflectoring.io/spring-boot-application-events-explained/. [Kasutatud 20 11 2020].

[45] Cybernetica AS, „ANDMEKAITSE JA INFOTURBE LEKSIKON,“ Cybernetica AS, [Võrgumaterjal]. Available: https://akit.cyber.ee/term/13110-unit-testing. [Kasutatud 21 11 2020].

[46] Guru99.com, „Unit Testing Tutorial: What is, Types, Tools, EXAMPLE,“ [Võrgumaterjal]. Available: https://www.guru99.com/unit-testing-guide.html. [Kasutatud 21 11 2020].

[47] Mockito, „Tasty mocking framework for unit tests in Java,“ Mockito, [Võrgumaterjal]. Available: https://site.mockito.org/. [Kasutatud 21 11 2020].

[48] Jenkins, „Getting started with Pipeline,“ [Võrgumaterjal]. Available: https://www.jenkins.io/doc/book/pipeline/getting-started/. [Kasutatud 28 11 2020].

[49] Oracle, „Oracle Technology Global Price List,“ Oracle, 8 10 2020. [Võrgumaterjal]. Available: https://www.oracle.com/assets/technology-price-list-070617.pdf. [Kasutatud 2 12 2020].

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Marite Rammo

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose Oracle SOA teenuse Java Spring tehnoloogiale üleviimine, mille juhendaja on Jaanus Pöial ja kaasjuhendaja Henry Mäeorg
    1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
    1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

07.01.2021

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Rakenduse ärireeglid

Konto (ingl *account*) on andmete kogum, mis iseloomustab rahalisi tehinguid antud konto piires. Kontod kätkevad endas infot arvete koostamise ja korrigeerimise, nende väljastamise aja ning viisi, tasumise ning tasumata jätmise kohta.

Teenuspunkt (ingl *service point*) on termin, millega tähistatakse kohta, kus teenust osutatakse. Füüsilises maailmas sellist punkti ei eksisteeri, füüsilises maailmas eksisteerivad mõõtepunkt ja liitumispunkt. Teenuspunktis toimub reeglina mõõtmine, see tähendab, et seal on mõõtepunkt ja arvesti.

Mõõtepunkt (ingl *metering point*), ühes liitumispunktis võib olla palju mõõtepunkte, kuid ei pruugi olla ühtegi.

Liitumispunkt (ingl *connection point*) on koht, kus lõppeb jaotusvõrk ja algab kliendile kuuluv võrk.

Ärireeglid on järgmised:

Konto ja teenuspunkti informatsiooni tuleb uuendada MDM-is ainult siis, kui luuakse uus konto koos teenuspunktiga (võrgulepingu staatus on 20 ehk aktiivne) või konto on muudetud mitteaktiivseks ja seos teenuspunktiga on suletud (staatus on 40 ehk peatatud). Teiste staatuste puhul ei muudeta midagi.

Kui toimub konto loomine, siis tuleb:

1. Luua konto
2. Luua teenuspunkt ja teenuspunktile lisada vastavalt spetsifikatsioonile parameetrid (näiteks mõõtepunkti tüüp, tarbimisgraafiku tüüp, liitumispunkti maksimaalne lepinguline lubatud koormus, aktiivenergia kadu, reaktiivenergia kadu, tootja)

3. Arvutada *High Limit* mõõdiku väärtus (maksimaalne võimsus, mida peakaitse lubab pikaajaliselt tarbida)

4. Luua konto ja teenuspunkti vaheline seos

5. Lisada arvesti

6. Luua teenuspunkti ja arvesti vaheline seos

7. Lisada teenuspunktile virtuaalsed kanalid

8. Luua mõõtmistulemuste seos teenuspunkti, kanali ja parameetriga

9. Luua teenuspunkti ja lisaandmete vaheline seos

10. Eraldi realiseerida mõõtepunktita kontode lisamise loogika

Kui toimub konto sulgemine, siis tuleb:

1. Sulgeda konto

2. Sulgeda konto seos teenuspunktiga

3. Sulgeda teenuspunkt