TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Jaan Kostõgov 175226IDDR

Rakendus mõõteandmete haldussüsteemi andmete migreerimiseks

Diplomitöö

Juhendaja: Jaanus Pöial
PhD

Kaasjuhendaja: Andre Soop
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: Jaan Kostõgov

15.05.2021

Annotatsioon

Antud diplomitöö eesmärk on analüüsida erinevaid tehnoloogiaid ja leida sobivaimad tehnoloogiad rakenduse arendamiseks, millega saaks töödelda ja migreerida suurt andmehulka.

Lõputöös tutvustatakse võrguettevõtte projekti, millega soovitakse välja vahetada vana mõõteandmete haldussüsteemi ja sellega seoses tuuakse välja vajadus teha valmis migratsiooni rakendus, millega saab migreerida andmeid ühest süsteemist teise. Diplomitöö raames analüüsitakse erinevaid tehnoloogiaid ja valitakse neist sobivamaid.

Lõputöö tulemusena tehti valmis Spring Batch konsooli rakendus, mis pärib andmebaasist andmeid, transformeerib need ja siis salvestab need XML failidena serverisse maha. Rakendusel on olemas automaattestid ja pidev tarkvaratarne ahel. Selle rakendusega tehti edukalt ära andmete migratsiooni. Lõputöö raames jõuti järelduseni, et Spring Batch on hea raamistik andmetöötluse rakenduste arenduseks, mis sobib hästi andmete migratsiooniks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 21 leheküljel, 5 peatükki, 18 joonist.

Abstract

Application for Meter Data Management System Data Migration

This thesis aims to analyze different technologies and find the most suitable technologies for application development, which can process and migrate large amounts of data.

The thesis introduces the network company's project, which aims to replace the old measurement data management system with the new one. This project highlights the need to create a migration application that can migrate data from one system to another. In the diploma thesis, different technologies are analyzed, and the most suitable ones are selected.

As a result of the thesis a Spring Batch console application was created, which retrieves data from the database, transforms it, and then saves it as XML files to the server. The application has automated tests and a continuous software delivery pipeline. This application completed successfully the data migration. In the thesis, the author concluded that Spring Batch is a good framework for data processing application development, which is well suited for data migration or complex script replacement.

The thesis is in estonian and contains 21 pages of text, 5 chapters, 18 figures.

Lühendite ja mõistete sõnastik

API	*Application Program Interface.* Rakendusliides.
FTP	*File Transfer Protocol.* Failiedastusprotokoll.
GIT	Versioonihaldustarkvara
JAR	*Java Archive.* Java arhiivifail.
Jenkins	Automaatika server.
JMS	*Java Message Service.* Java sõnumiteenus.
JVM	*Java Virtual Machine*. Java virtuaalmasin.
MDM	*Meter data management.* Mõõteandmete haldussüsteem.
OJDBC	*Oracle Java Database Connectivity.* Java API mis määrab ära kuidas kasutaja saab Oracle andmebaasile ligi.
SFTP	*Secure File Transfer Protocol.* Turvaline failiedastusprotokoll.
SOAP	*Simple Object Access Protocol.* Lihtne objektipöördusprotokoll, mida kasutatakse struktureeritud andmete edastamiseks.
SQL	*Structured Query Language.* Struktuurpäringukeel, mis on loodud andmebaasi haldamiseks ja päringute tegemiseks.
WSDL	*Web Service Description Language.* Veebiteenuste kirjelduskeel.
XML	*Extensible Markup Language.* Laiendav märgistuskeel.

Sisukord

Jooniste loetelu

1 Sissejuhatus

Võrguettevõttes võetakse kasutusele uus mõõteandmete haldussüsteem. Selle kasutamiseks on vaja tänasest süsteemist migreerida mõõtepunkti andmed uude süsteemi. Mõõtepunkti andmed on arvestid, mõõtepunkti karakteristikud ja muud MDM-i tööks vajalikud algandmed. Andmete migratsiooniks on vaja luua lahendus, mis loeks ja transformeeriks vana süsteemi andmeid ja migreeriks need uude süsteemi.

Lõputöö eesmärgiks on analüüsida ja kirjeldada tehnoloogiaid, millega saaks ehitada valmis andmete töötlemiseks rakenduse. Lõputöö tulemusena peaks saama valmis migratsiooni rakendus, mis võimaldaks migreerida suuri andmekogumeid ühest süsteemist teise.

Lõputöö metoodikaks on kirjandusallikate kasutamine ja erinevate tehnoloogiate analüüsimine. Rakenduse arenduse metoodikaks on agiilne arendus. Rakenduse arendus toimus 2 nädalaste sprintidega, kus iga sprindi lõpp sai valmis rakenduse tükk, mida sai kohe kasutada.

Lõputöö teises peatükis tutvustatakse teoreetilist tausta. Teoreetiline taust koosneb võrguettevõtte vana mõõteandmete haldussüsteemi väljavahetamise projekti kirjeldusesest. Tuuakse välja selle projekti eesmärgid ja sellega seoses vajadust luua rakendus andmete migratsiooniks. Lisaks selles peatükis tuuakse välja migratsiooni rakenduse lähtetingimusi, millega tuleb arvestada rakenduse loomisel.

Kolmandas peatükis algul analüüsitakse mõõtepunkti andmete migreerimiseks võimalikud andmete edastamise viise ja valitakse neist parim. Pärast seda analüüsitakse raamistike ja tööriistu, mis sobivad uue rakenduse arendamiseks.

Neljandas peatükis on diplomitöö praktilise osana loodud migratsiooni rakenduse kirjeldus. Seal tuuakse välja rakenduse arhitektuuri ja kõiki komponente koos koodinäidistega.

Viiendas peatükis on diplomitöö kokkuvõtte. Kokkuvõttes tuuakse välja tulemused ja järeldused. Lisaks pakutakse välja antud rakenduse edasiarendamise võimalusi.

2 Teoreetiline taust

Selles peatükis tutvustatakse ettevõtte uut projekti, mõõteandmete süsteemi ja tehtava rakenduse lähtetingimusi.

2.1 Projekti taust

Võrguettevõttes vahetatakse välja vana mõõteandmete haldussüsteem. Süsteem on olnud kasutusel 10 aastat ja tootjapoolne tugi lõppes 2020 aastal. Süsteem on oma vanuse tõttu liiga aeglane ja ebastabiilne, mis teeb võrguettevõtte arveldust keeruliseks.

Mõõteandmete haldussüsteem on tarkvara, mis hoiab ja haldab kaugloetavate arvestite andmeid. Neid andmeid kasutatakse arveldamiseks.

Projekti eesmärgid on :

- Mõõteandmete ja arvelduprotsesside kulu vähendamine.
- Kogusepõhise arvelduse kasutusele võtmine.
- Vana mõõteandmete haldussüsteemi välja vahetamine

Selle projekti üks osadest on andmete migratsioon. Uue MDMi kasutamiseks on vaja tänasest süsteemist migreerida mõõtepunktide andmeid uude süsteemi. Migratsiooni jaoks on vaja luua rakendus, mis loeks andmebaasist kokku korjatud andmeid ja transformeeriks neid MDMile arusaadavaks kujuks.

2.2 Rakenduse lähtetingimused

Ettevõttel ei ole ligipääsu MDM süsteemi andmebaasile. Seega andmebaasist andmebaasi migratsiooni ei ole võimalik teha.

Kõik mõõtepuntki andmed asuvad ORACLE andmebaasis. Mõõtepunkti andmed on eelnevalt kokku korjatud mitmest andmebaasist ja salvestatud maha andmebaasi tekitatud

vahetabelitesse. Selline eeltöö oli tehtud selle jaoks, et andmeid saaks enne valideerida. Samuti see teeb andmete migratsiooni lihtsamaks, kuna andmete lugemine hakkab olema ainult ühest andmebaasist. Eelnevalt on analüütiku poolt paika pandud SQL laused, millega saab vajalikud andmed tehtud vahetabelitest kätte.

Uus MDM kasutab andmete vastuvõtmiseks ja saatmiseks SOAP adaptereid. Andmete migreerimiseks peaksid andmed olema transformeeritud SOAP XML sõnumiteks.

Arendatavale migratsiooni rakenduse funktsionaalsed ja mitte funktsionaalsed nõuded olid paika pandud analüütiku poolt. Rakenduse nõuded on toodud välja Lisas 2.

3 Lahenduse analüüs

Antud peatükis analüüsitakse erinevaid andmete edastamise meetodeid ja valitakse neist sobivaim. Rakenduse ehitamiseks analüüsitakse erinevaid tööriistu ja raamistike, millega saab teha valmis andmete migratsiooniks sobiva rakenduse.

3.1 Mõõteandmete edastamise meetodid

Selles peatükis analüüsitakse erinevaid andmeedastamise viise, mida uus MDM lubab ja valitakse parim viis andmete migreerimiseks uude süsteemi.

Antud süsteemis on võimalik kasutada SOAP adaptereid. SOAP on veebiteenuste sõnumite protokoll, mida süsteemid kasutavad struktureeritud andmete edastamiseks. SOAP kasutab andmete edastamiseks olemasolevaid transport protokolle nagu näiteks HTTP või SMTP. Sõnumite edastamiseks kasutatakse XML-i. Sõnumi juurelemendiks on nn ümbrik, mis koosneb päisest ja kehast. Päis on valikuline element, mis hoiab enda sees suhtluseks vajalikud rakenduse põhiseid andmeid nagu näiteks kasutaja ja parool. Keha on kohustuslik element, mille sees hoitakse päringu või vastuse sõnumi sisu. Veebiteenuse kirjeldamiseks kasutatakse WSDL-i. WSDL näitab ära sõnumi struktuuri ja milliseid operatsioone on antud veebiteenusel võimailik kasutada. WSDLi abil on võimalik genereerida koodi, millega saab rakendus panna sõnumeid kokku ja saata neid veebiteenuse pihta. [1]

Tavaliselt SOAP veebiteenused kasutavad sõnumi edastamiseks HTTP protokolli [2]. Uues süsteemis saab kasutada ainult HTTP sünkroonset sõnumiedastust. Sünkroonse sõnumiedastamise eelis on see, et rakendus saab iga päringuga teada, kui sõnum on teise rakenduse poolt vastu võetud ja edukalt ära töödeldud. Sünkroonse sõnumi puuduseks on rakenduste sidusus. Sõnumi saatmisel rakendus jääb ootama päringu vastust, mis blokeerib rakenduse tööd. [3] See on halb, kuna migratsiooni rakenduse andmete töötlemise kiirus ja töökindlus hakkaksid sõltuma teisest rakendusest.

SOAP saab kasutada sõnumite edastamiseks ka JMSi [2]. JMS on API, mis võimaldab saata ja võtta vastu sõnumeid asünkroonselt. JMSis kasutatakse vaherakendust, mis tegeleb sõnumite edastamisega. Tänu sellele sõnumi saatja ei pea teadma midagi sõnumi vastuvõtjast. Nii ei ole rakendused omavahel sõltuvuses. Rakendus saab saata sõnumeid

ka siis, kui vastuvõtja rakendus ei ole püsti. Vastuvõtja võib hiljem neid sõnumeid oma tempoga ära töödelda. JMS-l on olemas 2 sõnumside mudelit: *point-to-point* ja *publish-and-subscribe*. [4]

Point-to-point mudelis saatja saadab sõnumit *queue*'sse ja vastuvõtja võtab seda *queue*'st vastu. Selles JMS sõnumside mudelis saab vastuvõtja lugeda sõnumit ainult 1 kord [4]. See ei sobi andmete migratsiooniks, kuna töötlemise vea korral peaks katkist sõnumit jätma alles, et hiljem oleks võimalik seda analüüsida ja proovida seda uuesti saata. JMS-s on võimalik kasutada katkiste sõnumite *queues*-i [4], aga see nõuaks uue MDMi konfigureerimist.

Publish-and-subscribe mudelis võib olla mitu vastuvõtjat. Selle sõnumside mudeli eelis on see, et sõnumi vastuvõtmisel sõnum ei kao ära ja hiljem vajadusel saab seda sõnumit uuesti saata. [4] See oleks parem valik kui *point-to-point* sõnumside mudel, kuid sellist tüüpi adaptrit pole uues MDM-s konfigureeritud ja selle konfigureerimine oleks tasuline.

MDM-l on konfigureeritud adapter, mis loeb MDM-i server kaustast XML sõnumi faile. Faili töötlemise ebaõnnestumise korral süsteem salvestab faili eraldi veakausta, mis annab võimaluse pärast vigaseid sõnumeid uurida ja uuesti saata. Failide edastamiseks kasutatakse SFTP-d [5]. SFTP on protokoll turvaliseks failide edastamiseks üle veebi. SFTP kasutab failide edastamiseks FTP protokolli ja see sisaldab SSH turvaelemente [5].

Migratsiooni andmete edastamiseks valitakse failide edastamist läbi SFTP . Selle jaoks peab rakendus salvestama faile ettevõtte serveri kausta ja sealt hiljem viiakse need andmed MDMi serveri kausta, kasutades SFTP-t. Selline andmete edastamisviis on hea, kuna siis saab XML faile enne valmis teha ja andmete sisestamist uude süsteemi saab teha kunagi hiljem. See lubab enne failide migreerimist transformeeritud andmeid üle vaadata ja teha kindlaks, et andmete lugemisel pole tekkinud probleeme.

3.2 Programmeerimiskeel

Java on 1995 aastal loodud objektorienteeritud programmeerimiskeel, mille peamine eesvedaja on praegu Oracle. Java jookseb JVMi peal, mis teeb seda platvormi sõltumatuks. [6] Praegusel hetkel Java on üks enim kasutatavaid programmeerimiskeeli [7].

Kotlin on 2009. aastal loodud objektorienteeritud ja funktsionaalne programmeerimiskeel, mis jookseb JVMi peal ja mida saab kompileerida ka javascriptiks. Kotlini standart teek kasutab Java teeke, mille tõttu Kotlinil on olemas kõik java funktsionaalsused. Lisaks Kotlinil on olemas funktsionaalsusi mida Javal ei ole, mis teevad koodi kirjutamist mugavamaks. Kotlinil on väga hea integratsioon Intellij IDEA-ga. [8] Kotlini puudus on see, et see ei ole väga laialdaselt kasutatud programmeerimiskeel, mille tõttu on raske leida arendajaid, kes oskaks kasutada antud programmeerimiskeelt [7].

Programmeerimise keele valikus peab arvestama arendusmeeskonna oskustega. Antud keeltest osatakse kõige paremini Javat. Lisaks Java on populaarsem kui Kotlin, mis on tähtis näitaja, kuna tulevikus, kui peaks rakendust täiendama, saab selle jaoks leida lihtsamini arendajaid [7].

3.3 Raamistik

Spring Batch on avatud lähtekoodiga kergekaaluline pakktöötluse raamistik. Antud raamistikul on palju kasulikke funktsionaalsusi, mis lihtsustavad andmete töötlemist. Sellised funktsionaalsused on näiteks logimine, masstöö statistika, töö taaskäivitamine ja ressursside haldus. Lisaks raamistik võimaldab töödelda suurt andmehulka kasutades protsesside optimeerimise ja partitsioneerimise funktsionaalsusi. Spring Batch on ülesehitatud Spring raamistiku peal. [9]

Spring on avatud lähtekoodiga 2003 aastal loodud tarkvara arendamise raamistik. Spring on java põhine raamistik, kuid see toetab ka teisi JVM keeli nagu näiteks Groovy ja Kotlin. Spring rakenduse ehitamiseks kasutakse Java objekte, mida nimetatakse Springi Bean'ideks. Objektide sidumiseks kasutatakse XML konfiguratsiooni faile. [10]

Springi konfigureerimine on küllaltki keeruline, seetõttu on soovituslik kasutada Spring Boot raamistiku. Spring Bootil on lihtsustatud sõltuvuste haldamine. Ühe sõltuvuse lisamisega lisatakse kõik sellega seotud teegid. Lisaks Spring Bootis kasutakse autokonfigureerimist, mis automaatselt konfigureerib rakendust vastavalt lisatud sõltuvustele. See teeb arendust mugavamaks ja kiiremaks, kuna arendaja ei pea käsitsi midagi konfigureerima. [11] Springi Boot on enim kasutatav Java tehnoloogia [12]. See

on hea, sest vajadusel tulevikus saab leida lihtsasti rakenduse jätkuarenduseks Java arendajaid ja tekkinud probleemide korral leiab kergemini internetist abi

Analüüsi põhjal valitakse rakenduse raamistikuks Springi booti koos Spring Batchiga. Spring Batchi kasutamiseks tuleb lisada rakendusele Spring Batch teeki ja konfiguratsiooni failis lülitada sisse Batch protsessi [13].

3.4 Rakenduse ehitustööriistad

Rakenduse ehituseks saab kasutada ehitustööriistu, mis automatiseerivad rakenduse kokkupanemist ja sellega seotud tegevusi nagu näiteks testide käivitamist. [14]

Java rakenduste populaarne ehitustööriist on Maven. Maveni projekti kirjeldus asub XML failis. Mavenis kasutatakse enamasti eeldefineeritud käske, mis teevad ehitusprotsessi kokkupanemist väga lihtsaks, arendajal ei ole vaja defineerida igat ehitusetappi. Selletõttu Maven pole aga väga paindlik. Kohandatud funktsioonide loomine on sellel ehitustööriistal väga keeruline. [15]

Maveni kõrval teine populaarne ehitustöörist on Gradle. Gradle on avatud lähtekoodiga ja see töötab JVM-i peal. Gradle skriptis saab kasutada Groovy’t või Kotlin’it ja skriptis saab tekitada lihtsasti kohandatud funktsioone. Seetõttu Gradle on väga paindlik. Lisaks Gradle’i jõudlus on parem kui Maveni oma. [11]

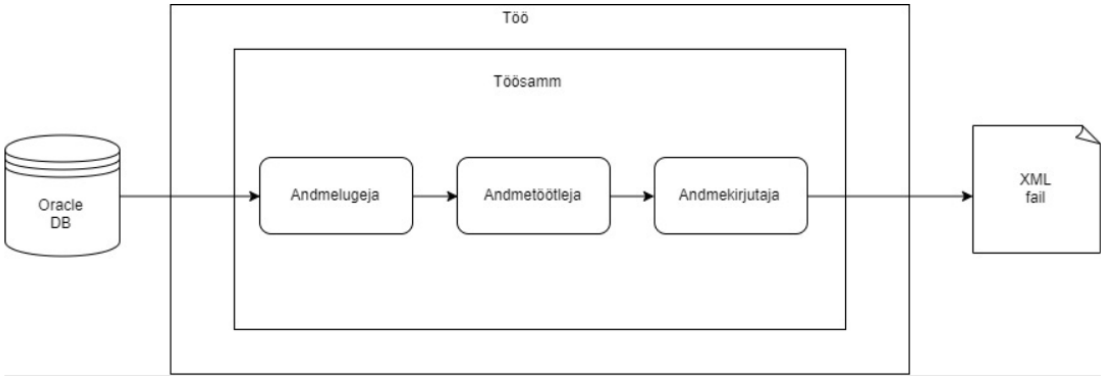
Rakenduse ehitustööriistaks valitakse Gradle’t, sest tehtaval rakendusel peavad olema erinevad ehitusega seotud kohandatud funktsioonid nagu näiteks koodi analüüs ja WSDL’ist Java klasside genereerimine.

4 Lahendus

Lõputöö tulemusena sai loodud konsooli rakendus, mida saab kasutaja käivitada serveris, et luua migratsiooniks vajalikud XML failid, mida pärast saadetakse läbi SFTP-d MDM serverisse, kus neid uus MDM võtab vastu.

4.1 Rakenduse arhitektuur

Kõik vajalikud andmete migratsioonid on tehtud eraldi töödena. Kokku on 23 tööd. Töö koosneb ühest või mitmest sammust. Töösammus on kirjeldatud andmetöötlemise protsess. Lugeja loeb andmebaasist tükk andmeid ja need liiguvad edasi ühekaupa andmetöötlejasse, kus tehakse nendega äriloogikat ja pannakse kokku SOAP sõnumi objekte. Pärast töödeldud andmete hulka kirjutatakse XML failina maha(Joonis1).



Joonis 1. Rakenduse arhitektuur.

4.2 WSDL2Java

SOAP sõnumite kirjutamiseks kasutatakse Java objekte, mis on genereeritud WSDL-st. WSDL-i genereerimiseks kasutatakse WSDL2Java Gradle pistikprogrammi. Selle jaoks tehakse valmis *wsdl2java* Gradle'i ülesanne. Selles ülesandes määratakse ära WSDL failide asukohta ja kuhu Java failid genereeritakse(Joonis 2).

```
wsdl2java {
    println 'Generate classes from WSDL:'
    wsdlsToGenerate =
            [
                    ["$projectDir/src/main/resources/wsdl/UniversalSyncService.wsdl"],
                    ["$projectDir/src/main/resources/wsdl/UniversalAMIMeterReadsResponseService.wsdl"]
            ]
    generatedWsdlDir = file("src/generated/main/java")
    locale = Locale.getDefault()
}
```

Joonis 2. wsdl2java Gradle ülesanne.

4.3 Töö

Töö on Spring batchi juurkonfiguratsioon, mis kirjeldab käivitatavat pakktöötluse protsessi [16]. Iga migratsiooni töö on konfigureeritud eraldi failis. Konfigureerimiseks on kasutatud Java-põhist Spring Boot konfiguratsiooni. Selle jaoks on klassile lisatud *@Configuration* annotatsioon. Spring Batchi funktsionaalsuse ja baaskonfiguratsiooni kasutamiseks on lisatud *@EnableBatchProcessing* annotatsioon. Konfiguratsiooni parameetrite asukoha määramiseks kasutatakse *@PropertySource* annotatsiooni. [13] Antud tööd käivitatakse ainult siis, kui *run.device.job* parameeter on *true* (Joonis 3).

```
@Configuration
@EnableBatchProcessing
@ConditionalOnProperty(name="run.deviceSync.job", havingValue = "true")
@PropertySource("classpath:application.properties")
@PropertySource("classpath:jobQuery.properties")
public class DeviceExportJobConfig {
```

Joonis 3. Töö konfiguratsiooni klass.

Tööd pannakse kokku Spring Batch *JobBuilder*'iga. Alguses kasutades *JobBuilderFactory't* luuakse uus *JobBuilder* ja määratakse ära töö nimeks *deviceJob*. *JobBuilder*'ga saab lisada üht või mitu töösammu. Esimest sammu lisatakse *start* meetodiga ja järgmiseid samme lisatakse *next* meetodiga. Samme käivitatakse samas järjekorras nagu neid lisati. Kuna antud tööl on ainult 1 samm, siis saab seda lisada lihtsalt *flow* meetodiga. Lisaks tööle lisatakse juurde ka *jobExecutionListener*'i, mis lisab tööprotsessi elutsüklile tagasiside funktsioone(Joonis 4). [15]

```
@Bean
public Job deviceJob(JobBuilderFactory jobs, @Qualifier("deviceStep") Step step, JobExecutionListener jobExecutionListener){
    return jobs.get("deviceJob") JobBuilder
            .incrementer(new RunIdIncrementer())
            .preventRestart()
            .listener(jobExecutionListener)
            .flow(step) JobFlowBuilder
            .end() FlowJobBuilder
            .build();
}
```

Joonis 4. Töö ehitamine.

JobExecutionListener's lisatakse töö alustamise ja töö lõpetamise info logimist. Selle jaoks implementeeritakse *JobExecutionListener*'i liidest ja kirjutatakse üle *afterJob* ja *beforeJob* meetodeid.

BeforeJob meetodis logitakse maha info, et töö läks käima. Stringi lihtsamaks kokkupanemiseks kasutatakse *StringBuilder*'i(Joonis 5).

```
@Override
public void beforeJob(JobExecution jobExecution) {
    StringBuilder protocol = new StringBuilder();
    protocol.append("\n" + LINE + "\n")
            .append("Starting ").append(jobExecution.getJobInstance().getJobName()).append(" \n")
            .append(LINE + "\n");

   if (LOGGER.isInfoEnabled()) LOGGER.info(protocol.toString());
}
```

Joonis 5. Töö alustamise logimine.

AfterJob meetodis pannakse kokku töö lõpetamisega seotud infot. Töö lõpetamisel logitakse maha alustamise, lõpetamise kellaajad ja seda, kas tööd lõpetati edukalt või tekkis mingi viga. Lisaks sõnumis on toodud välja lõpetatud töö parameetrid ja kirjutatud ning vahele jäetud failide arv(Joonis 6).

```
@Override
public void afterJob(JobExecution jobExecution) {
    StringBuilder protocol = new StringBuilder();
    protocol.append("\n" + LINE + "\n")
            .append("Protocol for ").append(jobExecution.getJobInstance().getJobName()).append(" \n")
            .append("  Started     : ").append(jobExecution.getStartTime()).append("\n")
            .append("  Finished    : ").append(jobExecution.getEndTime()).append("\n")
            .append("  Exit-Code   : ").append(jobExecution.getExitStatus().getExitCode()).append("\n")
            .append("  Exit-Descr. : ").append(jobExecution.getExitStatus().getExitDescription()).append("\n")
            .append("  Status      : ").append(jobExecution.getStatus()).append("\n")
            .append(LINE + "\n")
            .append("Job-Parameter: \n");
    JobParameters jp = jobExecution.getJobParameters();
    for (Entry<String, JobParameter> entry : jp.getParameters().entrySet()) {
        protocol.append("  ").append(entry.getKey()).append("=").append(entry.getValue()).append("\n");
    }
    protocol.append(LINE + "\n");
    for (StepExecution stepExecution : jobExecution.getStepExecutions()) {
        protocol.append("\n" + LINE + "\n")
                .append("Step ").append(stepExecution.getStepName()).append(" \n")
                .append("WriteCount: ").append(stepExecution.getWriteCount()).append("\n")
                .append("SkipCount: ").append(stepExecution.getSkipCount()).append("\n")
                .append(LINE + "\n");
    }

    if (LOGGER.isInfoEnabled()) LOGGER.info(protocol.toString());
}
```

Joonis 6. Töö lõpetamise logimine.

4.4 Töösamm

Samm on objekt, mis on sõltumatu töö osa, mille sees on kõik tööprotsessiks vajalikud komponendid. Sammu pannakse kokku *stepBuilder*'iga. Alguses kasutades

StepBuilderFactory-t luuakse uus *stepBuilder* ja määratakse ära sammu nimeks *deviceStep*. Sammul saab olla üks või mitu lugejat, protsessorit või kirjutajat, kuid on soovituslik kasutaja komponente ainult ühekaupa, et sammu loogika oleks võimalikult lihtne. Seadme migreerimise töösammul on *StepBuilder*'iga lisatud 1 lugeja, 1 protsessor ja 1 andmekirjutaja. Sammule on lisaks lisatud ka *chunkListener*, millega logitakse maha töödeldud andmete arvu. *Build* meetodiga tehakse valmis kokkupandud töösamm. (Joonis 7). [15]

```
@Bean
public Step deviceStep(StepBuilderFactory stepBuilderFactory,
                       ItemReader<DeviceDTO> reader,
                       ItemProcessor<DeviceDTO, Device> processor,
                       ItemWriter<Device> writer,
                       ChunkListener chunkListener) {
    return stepBuilderFactory.get("deviceStep") StepBuilder
            .<DeviceDTO, Device>chunk(chunkSize) SimpleStepBuilder<DeviceDTO, Device>
            .reader(reader)
            .processor(processor)
            .writer(writer)
            .listener(chunkListener) AbstractTaskletStepBuilder<SimpleStepBuilder<DeviceDTO, Device>>
            .build();
}
```

Joonis 7. Töösammu ehitamine.

4.5 Andmete lugemine

Spring Batchis kasutakse andmete lugemiseks *ItemReader*'i. Antud raamistikul on palju eeldefineeritud andmelugejaid, kuid vajadusel saab kasutada ka omatehtud lugejat. Selle jaoks tuleb implementeerida *ItemReader<T>* liidest. Selles rakenduses kasutatakse eeldefineeritud andmebaasilugejat. Andmebaasilugejaks on *JdbcCursorItemReader*, mis kasutab andmebaasi allikat ja sealt tulevate andmete objektideks kokkupanemiseks *Rowmapper*'i. [15]. Konfiguratsiooni failis pannakse kokku *JdbcCursorItemReader*, millele lisatakse andmebaasi andmeallikas, SQL päring ja *RowMapper*(Joonis 8).

```
@Bean
@StepScope
public JdbcCursorItemReader<DeviceDTO> reader(@Qualifier("purkDataSource") DataSource dataSource) {
    JdbcCursorItemReader<DeviceDTO> reader = new JdbcCursorItemReader<>();
    reader.setSql(query);
    reader.setDataSource(dataSource);
    reader.setRowMapper(new DeviceRowMapper());

    return reader;
}
```

Joonis 8. Andmelugeja ehitamine.

Andmebaasi ühenduseks luuakse konfiguratsiooni klassis andmeallikat. Andmeallikaks kasutatakse Hikarit. Hikari on 2012 aastal loodud väga kergekaaluline ja kiire andmebaasi ühenduste jagamise raamistik [16] . Selles konfiguratsiooni failis lisatakse andmeallikale vajalikud parameetrid, mis on konfigureeritavad *application.properties* failis. Kuna andmeid hakatakse lugema Oracle andmebaasist, siis selle jaoks lisatakse andmeallikale *driverClassName* parameetri, millega määratakse ära, et on vaja kasutada OJDBC draiverit. Andmebaasi ühenduseks lisatakse ka vajalikud parameetrid: url, andmebaasi kasutajanimi ja parool(Joonis 9).

```
@Configuration
public class DatabaseConfiguration {
    @Bean(name = "purkDataSource")
    @Primary
    public DataSource purkDataSource (@Value("${database.purk.jdbc-url}") String url,
                                      @Value("${database.purk.driver-class-name}") String driver,
                                      @Value("${database.purk.username}") String username,
                                      @Value("${database.purk.password}") String password,
                                      @Value("1") int poolSize) {

        HikariDataSource dataSource = (HikariDataSource) DataSourceBuilder.create()
                .url(url)
                .driverClassName(driver)
                .username(username)
                .password(password)
                .build();

        dataSource.setMaximumPoolSize(poolSize);

        return dataSource;
    }
}
```

Joonis 9. Andmebaasi ühenduse konfiguratsioon.

SQL päringud asuvad eraldi *jobQuery.properties* konfiguratsiooni failis. Nii saab vajadusel kasutaja SQL päringuid muuta. SQL päringu muutmisel peab aga olema ettevaatlik, et selle päringu veerude nimed ei muutuks, sest siis *RowMapper* ei suuda seda objektiks teha. Selles failis määrata ära parameetri nimi ja selle SQL(Joonis 10).

```
deviceExportQuery=\
SELECT manufacturer_code,\
       serial_number,\
       model_number,\
       utc_number\
FROM AS_SCOPE_REAL_METERS
```

Joonis 10. Töö SQL jobQuery.properties failis.

Päring ridu pannakse kokku Java objektideks *RowMapper'*is. Selle jaoks tehakse valmis klass, mis implementeerib *RowMapper* liidest ja kirjutatakse üle *mapRow* meetodi. Selles meetodis tehakse valmis uus *DeviceDTO* objekt, kuhu pannakse andmebaasist tulevaid andmeid. Andmed asuvad *ResultSet'is*, kust saab need kätte kasutades meetodi *getString*, mille sisendiks on veerunimi, mida soovitakse kätte saada. (Joonis 11). [17]

```
public class DeviceRowMapper implements RowMapper<DeviceDTO> {
    @Override
    public DeviceDTO mapRow(ResultSet rs, int rowNum) throws SQLException {
        DeviceDTO deviceDTO = new DeviceDTO();
        deviceDTO.setManufacturerCode(rs.getString( columnLabel: "manufacturer_code"));
        deviceDTO.setSerialNumber(rs.getString( columnLabel: "serial_number"));
        deviceDTO.setModelNumber(rs.getString( columnLabel: "model_number"));
        deviceDTO.setUtcNumber(rs.getString( columnLabel: "utc_number"));
        return deviceDTO;
    }
}
```

Joonis 11. DeviceRowMapper klass.

4.6 Andmete transformeerimine

Rakenduse äriloogika asub *ItemProcessor*'is. See on komponent, mis võtab vastu üht objekti, transformeerib seda ja siis tagastab uut objekti. Andmete töötlemine käib ühekaupa mitte tükkidena nagu andmete lugemine või kirjutamine. Protsessori klassi tegemiseks implementeeritakse *ItemProcessor* liidest ja määratakse ära sissetuleva ja väljatuleva objekti tüüpi. [17] Protsessori töö algab *process* meetodist, kuhu tuleb sisse *DeviceDTO* objekt, mis hoiab enda sees seadme andmeid. Nendest andmetest tehakse valmis *Device* objekt. Seadme ja selle parameetri kokkupanemise loogikat tehakse eraldi meetodites, et kood oleks loetavam. Valmis tehtud objekt liigub edasi andmekirjutajasse. (Joonis 12).

```
public class DeviceProcessor implements ItemProcessor<DeviceDTO, Device> {

    @Override
    public Device process(DeviceDTO item) { return this.assembleDeviceBlock(item); }

    private Device assembleDeviceBlock(DeviceDTO deviceDTO) {
        Device device = new Device();
        String modelNumber = deviceDTO.getModelNumber();
        if (deviceDTO.getManufacturerCode() != null)
            device.setMRID(deviceDTO.getManufacturerCode().substring(0, 2) + deviceDTO.getSerialNumber());
        device.setModel(modelNumber);
        device.setMfgSerialNumber(deviceDTO.getSerialNumber());
        device.setElectronicId(deviceDTO.getSerialNumber());
        device.setBadgeId(deviceDTO.getUtcNumber());
        device.setMake(deviceDTO.getManufacturerCode());
        device.setType("Meter");
        if ("ARVESTI PUUDUB".equals(modelNumber) || "KOMBIARVESTI PUUDUB".equals(modelNumber)) {
            device.setVirtualInd("Y");
        } else {
            device.setVirtualInd("N");
        }
        device.setClassName("Electric");
        device.setDeviceFunctionType("N");
        device.getParameter().add(assembleDeviceParameter(deviceDTO));

        return device;
    }

    private Parameter assembleDeviceParameter(DeviceDTO deviceDTO) {
        Parameter parameter = new Parameter();
        parameter.setName("Meter Config Code");
        parameter.setValue(deviceDTO.getModelNumber());

        return parameter;
    }

}
```

Joonis 12. Seadme andmete protsessor.

4.7 Andmete kirjutamine

Andmed, mis tulevad protsessorist, kogutakse kokku ja siis neid kirjutab maha andmekirjutaja. Selle jaoks tehakse valmis abstraktne klass, mis implementeerib *ItemWriter* liidest ja kirjutab üle *write* meetodi. [17] *Write* meetodis sissetulnud andmeid tükeldatakse väiksemateks hulkadeks. Igat tekitatud hulka pannakse uude sõnumisse ja siis sõnum salvestatakse failina maha. Selline funktsionaalsus oli tehtud selle jaoks, et kasutaja saaks valida, mitu tükki andmeid peaks üks sõnum sisaldama. Faili salvestamisel lisatakse failile nime, mis koosneb kasutaja poolt määratud nimest ja faili järjekorranumbrist. Tehtud abstraktset klassi laiendatakse alamklassiga, kus määratakse ära sissetuleva objekti tüüpi ja selle lisamist SOAP sõnumisse. (Joonis 13).

```
@Override
public void write(List<? extends T> items) throws Exception {
    List<List<T>> output = Lists.partition((List<T>)items, itemsPerFile);
    for(List<T> partition : output) {
        SDPSyncMessage message = createSdpSyncMessage(partition);
        writeSdpSyncMessageFile(pageCount, outputDir, fileName, message);
        pageCount++;
    }
}
```

Joonis 13. Faili kirjutamise meetod.

4.8 Rakenduse ehitamine/paigaldamine

Ettevõttes peavad kõik uued rakendused kasutama paigalduseks Jenkinsit. Paigalduse protsess on määratud *Jenkinsfile* failis. Paigalduse protsess on jagatud osadeks. Esimeses osas ehitatakse rakendust ja käivitatakse teste. Antud osa käivitatakse alati, kui arendaja paneb oma koodi GITi üles.(Joonis 11).

Teises osas käivitatakse SonarQube analüüsi. SonarQube on tööriist, mis analüüsib koodi kvaliteeti. Sonarqube suudab tuvastada vigu, halba koodi, turvaauke ja dubleeritud koodi. [19] Antud tööriist on ettevõttes kohustuslik ning see peab olema lisatud kõikides uutes rakendustes(Joonis 11).

Kolmas osa käivitatakse ainult siis, kui arendaja paneb koodi peaharusse. Selles sammus laetakse üles ehitatud rakendus ja selle konfiguratsiooni failid Jenkinsi serverisse, kus hiljem saab kasutaja sealt need kätte(Joonis 14).

```
pipeline {
  agent any
  stages {
    stage('Build') {
      tools {
        gradle 'Gradle 5.6.1'
      }
      steps {
        script {
          sh '''chmod a+x gradlew
            ./gradlew clean build'''
          junit 'build/test-results/test/TEST-*.xml'
        }
      }
    }
    stage('Sonar') {
      tools {
        gradle 'Gradle 5.6.1'
      }
      steps {
        script {
          sh '''./gradlew sonarqube -Dsonar.branch.name=${BRANCH_NAME} -PprojectVersion=${BRANCH_NAME}.${BUILD_NUMBER}'''
        }
      }
    }
    stage('Publish') {
      when {
        branch 'master'
      }
      steps {
        archiveArtifacts 'build/libs/*.jar'
        archiveArtifacts 'build/resources/main/*.properties'
      }
    }
  }
}
```

Joonis 14. Jenkinsfile rakenduse paigaldamiseks.

4.9 Rakenduse testimine

Rakenduse komponentide testimiseks kasutatakse Unit teste. Unit testide ülesanne on valideerida, et iga programmi ühik töötab nii nagu vaja. Unit testid on kasulikud, kuna need aitavad avastada varakult programmis vigu, millega saab kokku hoida palju aega. Lisaks Unit testid dokumenteerivad koodi. [20]

Rakenduse ühik testimiseks kasutatakse JUnit5 raamistiku ja komponentide sõltuvuste eemaldamiseks kasutakse Mockito-t. Testide lihtsustamiseks kasutatakse *assertAll* meetodi, mis lubab käivitada ühes testis mitu *Assert*'i [21]. Näitena on toodud välja seadme andmelugeja päringu *RowMapper*'i test. Alguses *When* meetodiga määratakse ära *resultSet*'i meetodi käitumist. Järgmisena käivitatakse testitavat meetodi ja siis kontrollitakse saadud tulemust(Joonis 15).

```
@RunWith(MockitoJUnitRunner.class)
public class DeviceRowMapperTest {

    @InjectMocks
    DeviceRowMapper deviceRowMapper;
    @Mock
    ResultSet resultSet;


    @Test
    public void mapRow_mapsSqlResultToDeviceDTO() throws SQLException {
        when(resultSet.getString( columnLabel: "manufacturer_code")).thenReturn("LANDISGYR");
        when(resultSet.getString( columnLabel: "serial_number")).thenReturn("43904582");
        when(resultSet.getString( columnLabel: "model_number")).thenReturn("DC450");
        when(resultSet.getString( columnLabel: "utc_number")).thenReturn("0003302158");

        DeviceDTO deviceDTO = deviceRowMapper.mapRow(resultSet,  rowNum: 1);
        assertAll( heading: "Map Device sql request result values to DeviceDTO",
                ()->assertEquals( expected: "LANDISGYR", deviceDTO.getManufacturerCode(),  message: "ManufacturerCode should be correct"),
                ()->assertEquals( expected: "43904582", deviceDTO.getSerialNumber(), message: "SerialNumber should be correct"),
                ()->assertEquals( expected: "DC450", deviceDTO.getModelNumber(), message: "ModelNumber should be correct"),
                ()->assertEquals( expected: "0003302158", deviceDTO.getUtcNumber(), message: "UtcNumber should be correct")
        );
    }
}
```

Joonis 15. *DeviceRowMapper* klassi Unit test.

4.10 Rakenduse kasutamine

Rakendust saab käivitada konsooli kaudu. Enne rakenduse käivitamist tuleb rakendust ära konfigureerida. Andmebaasiühenduseks peab lisama andmebaasi URL-i, kasutajat ja parooli. Lisaks kasutaja saab määrata ära maksimum andmebaasiühenduste arvu(Joonis 16).

```
database.purk.jdbc-url=<url>
database.purk.driver-class-name=oracle.jdbc.OracleDriver
database.purk.username=<username>
database.purk.password=<password>
database.purk.datasource.poolSize=1
```

Joonis 16. Andmebaasiühenduse parameetrid.

Konfiguratsiooni failis on võimalik määrata ära, mis tööd soovitakse käivitada. Selle jaoks tuleb väärtustada töö muutujat väärtusega *true.* Samal ajal ei saa käivitada mitu tööd. See tähendab seda, et ainult 1 töökäivitamise parameeter tohib olla *true*.(Joonis 17).

```
run.producerSync.job=true
run.highlimitSync.job=false
run.consproSdpSync.job=false
run.servicePointDeviceAssociation.job=false
run.deviceSync.job=false
run.serviceDeliveryPoint.job=false
```

Joonis 17. Töö käivitamise parameetrid.

Failis saab konfigureerida tööparameetreid. Tööl on võimalik määrata ära mitu rida päritakse andmeid ühe tükina ja mitu objekti salvestatakse ühte faili. Lisaks saab lisada faili nime ja kirjutamise asukohta. Samuti saab konfigureerida mõned sõnumi päiseelemendi väärtusi nagu näiteks allikas ja sõnumi ID(Joonis 18).

```
jobs.deviceSync.chunkSize=1000
jobs.deviceSync.itemsPerFile=1000
jobs.deviceSync.outputDir=<output path>
jobs.deviceSync.fileName=device_SDPSyncMigration
jobs.deviceSync.source=AssetSuite
jobs.deviceSync.messageId=Devices from AssetSuite
```

Joonis 18. Tööparameetrid.

Rakenduse JAR faili tuleb käivitada konsoolist. Selle jaoks peab olema installeeritud Java 11. Käivitamise käsul peavad olema lisatud parameetritena konfiguratsiooni failide asukohad, mida käivitatav rakendus hakkab kasutama.(Joonis 19).

```
C:\>java -jar flexsync-migration-0.0.1.jar --spring.config.location=file:jobQuery.properties --spring.config.location=file:application.properties
```

Joonis 19. Rakenduse käivitamine.

Rakendus logib maha hetkel ära töödeldud ridade arvu ja kui rakendus lõpetab oma tööd, siis konsoolis näidatakse lõpetatud töö statistikat

5 Kokkuvõte

Lõputöö raames sai tehtud valmis Spring Boot migratsiooni rakendus, mis kasutab andmete töötlemise protsessiks Spring Batch-i. Tehtud rakendusega tehti võrguettevõttes ära ka edukalt mõõtepunkti andmete migratsiooni. Tänu sellele võeti kasutusele uus mõõteandmete haldussüsteem. Migratsiooni protsessil probleeme ei tekkinud. Antud migratsiooni rakendust hakatakse kasutama ka tulevikus, siis kui läheb vaja migreerida uuesti mõõtepunkti andmeid arendus või test keskkonnas.

Tehtud rakenduse kood on tänu heale raamistikule väga hästi struktureeritud. Rakenduse komponendid ei sõltu üksteisest. Vajadusel saab neid komponente lihtsasti taaskasutada. Uue töö konfigureerimine on samuti lihtne. Uues töös võib kasutada ära ka teiste tööde samme. Rakenduse paigaldus on automatiseeritud, kasutades selleks uusi tehnoloogiaid. See võimaldab arendajatele kiiresti muudatusi sisse viia.

Tulevikus võib teha rakendust paremaks. Hetkel tööde käivitamine ja taas käivitamine on manuaalne, mida võiks automatiseerida. Samuti rakendust võib panna Docker konteinerisse, et see ei sõltuks keskkonnast, kus seda tööle pannakse. Vajadusel saab rakendusele lisada ka uusi andmelugejaid ja kirjutajaid, mis loeksid ja kirjutaksid andmeid kasutades teisi tehnoloogiaid.

Lõputöö tehtud rakenduse ja tehnoloogia kirjelduste järgi võib teha ka teisi andmete töötlemiseks rakendusi. Ettevõttes hakati seda tehnoloogiat kasutama ka teistes andmete migratsioonides. Autori arvates võiks seda raamistiku kasutada kõikide keeruliste SQL skriptide asemel andmebaasi andmete korrastamisel.

Kasutatud kirjandus

[1] F. Curbera, R. Khalaf, M. J. Duftler, W. Nagy, N. Mukhi ja S. Weerawarana, „Unraveling the web services web: an introduction to SOAP, WSDL, and UDDI,“ *IEEE Internet Computing,* 7 August 2002.

[2] Oracle, „Using SOAP Over JMS Transport,“ Oracle, [Võrgumaterjal]. Available: https://docs.oracle.com/middleware/1212/wls/WSGET/jax-ws-jmstransport.htm#WSGET3431. [Kasutatud 20 4 2020].

[3] C. F. D'Cruz, „Synchronous Implementation using Web Services,“ [Võrgumaterjal]. Available: https://www.enterpriseintegrationpatterns.com/patterns/messaging/ComposedMessagingWS.html. [Kasutatud 20 April 2021].

[4] D. A. Chappell, M. Richards ja M.-H. Richard, Java Message Service, 2nd Edition, O'Reilly Media, Inc., 2009.

[5] J. Ellingwood, „How To Use SFTP to Securely Transfer Files with a Remote Server,“ Digitalocean, 13 August1 2013. [Võrgumaterjal]. Available: https://www.digitalocean.com/community/tutorials/how-to-use-sftp-to-securely-transfer-files-with-a-remote-server. [Kasutatud 10 April 2020].

[6] L. Vogel, „Introduction to Java programming - Tutorial,“ Vogella, [Võrgumaterjal]. Available: https://www.vogella.com/tutorials/JavaIntroduction/article.html. [Kasutatud 10 April 2020].

[7] „TIOBE Index for April 2021,“ Tiobe, April 2021. [Võrgumaterjal]. Available: https://www.tiobe.com/tiobe-index/. [Kasutatud 10 April 2021].

[8] „Kotlin docs,“ Kotlin Foundation, 2021. [Võrgumaterjal]. Available: https://kotlinlang.org/docs/home.html. [Kasutatud 10 April 2021].

[9] „Spring Batch intro,“ Spring, 2021. [Võrgumaterjal]. Available: https://docs.spring.io/spring-batch/docs/current/reference/html/spring-batch-intro.html. [Kasutatud 10 April 2021].

[10] „Introduction to spring framework,“ GeeksforGeeks, 2019. [Võrgumaterjal]. Available: https://www.geeksforgeeks.org/introduction-to-spring-framework/. [Kasutatud 10 April 2021].

[11] M. Heckler, Spring Boot: Up and Running, O'Reilly Media, Inc., 2021.

[12] „2021 Java Technology Report,“ JRebel, 2021. [Võrgumaterjal]. Available: https://www.jrebel.com/blog/2021-java-technology-report. [Kasutatud 10 April 2021].

[13] „Spring Boot Reference Documentation,“ Spring, 2021. [Võrgumaterjal]. Available: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#using-boot-configuration-classes. [Kasutatud 10 April 2021].

[14] „Build automation,“ TrustRadius, [Võrgumaterjal]. Available: https://www.trustradius.com/build-automation. [Kasutatud 15 April 2021].

[15] Baeldung, „Ant vs Maven vs Gradle,“ Baeldung, 9 June 2020. [Võrgumaterjal]. Available: https://www.baeldung.com/ant-maven-gradle. [Kasutatud 1 5 2021].

[16] R. M. Rao, Spring Batch Essentials, Packt Publishing, 2015.

[17] Spring Batch - Reference, Spring, 2021.

[18] Baeldung, Baeldung, 2020. [Võrgumaterjal]. Available: https://www.baeldung.com/hikaricp. [Kasutatud 15 April 2021].

[19] K. Lomror, „Sonarqube: What it is and why to use it?," LoginRadius, 11 July 2020. [Võrgumaterjal]. Available: https://www.loginradius.com/blog/async/sonarqube/. [Kasutatud 15 April 2021].

[20] „Unit testing guide," Guru99, [Võrgumaterjal]. Available: https://www.guru99.com/unit-testing-guide.html. [Kasutatud 15 April 2021].

[21] „Class Assertions," Junit, [Võrgumaterjal]. Available: https://junit.org/junit5/docs/5.0.0-M2/api/org/junit/jupiter/api/Assertions.html#assertAll-org.junit.jupiter.api.Executable...-. [Kasutatud 15 April 2021].

[22] L. S. Sterling, The Art of Agent-Oriented Modeling, London: The MIT Press, 2009.

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

Mina, Jaan Kostõgov

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose Rakendus mõõteandmete haldussüsteemi andmete migreerimiseks, mille juhendaja on Jaanus Pöial ja kaasjuhendaja Andre Soop.
   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.

15.05.2021

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 - Funktsionaalsed ja mittefunktsionaalsed nõuded

Mittefunktsionaalsed nõuded:

- Peab olema võimalik määrata, mitme mõõtepunkti adnmed lähevad ühte faili.
- Migratsiooni sõnumid peavad olema salvestatud XML failidena.
- Peab olema võimalik analüüsida ja teha aruannet migreeritavate andmete alusel.
- Peab olema võimalik määrata kuupäeva alates millest migreeritakse andmeid.

Uue MDM-I tööks on vaja migreerida järgmisi andmeid:

- Aktiivsed mõõtepunktid.
- Kõik arvestid, mis on seotud mõõtepunktidega, koos konfiguratsioonidega.
- Arveldusesüsteemiga sünkroniseeritud mõõtepunktide parameetrid.
- Mõõtepunktide sündmused. Näiteks katkestused.
- Mõõtepunktide tüübid.
- Kui mõõtepunktil puudub arvesti, siis tuleb sellele lisada virtuaalne arvesti.
- Mõõtepunktide arvestide ajalugu.
- Arvutada maksimaalne mõõdiku limiit.