TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Cristian Noop

Universaalse logimisrakenduse loomine

Bakalaureusetöö (9 EAP)

Juhendaja: Mirjam Paales

Tartu 2020

Universaalse logimisrakenduse loomine

Lühikokkuvõte:

Käesoleva bakalaureusetöö eesmärgiks on luua universaalne logimise rakendus, mis koguks süsteemi erinevatest rakendustest kokku tegevuslogid ja võimaldaks kogutud logisid efektiivselt hallata. Universaalne rakendus luuakse klienditellimusena, seetõttu on lähteülesandel arvestatud eelkõige kliendi vajadustest. Töös on välja toodud kliendi nõuded rakendusele, mis tehnoloogiaid rakenduse koostamisel kasutati ja mis oli töö tulemus.

Võtmesõnad:

Grails, ActiveMQ, RabbitMQ, JqGrid, Liquibase

CERCS: P175 Informaatika

The Creation of a Universal Logging Application

Abstract:

The goal of this Bachelor's thesis is to create a universal application, that gathers action logs from several applications, saves them and offers an effective way for administrating the data. The universal solution is developed for a client, so the client's needs are a priority when creating the application. The thesis shows the client's requirements for the application, presents the technologies used for development and describes the outcome of the thesis.

Keywords:

Grails, ActiveMQ, RabbitMQ, JqGrid, Liquibase

CERCS: P175 Informatics

Sisukord

Sissejuhatus

Universaalsed infotehnoloogilised lahendused on tänapäeval võtmeteguriteks, mis loovad eeliseid majandusettevõtete konkureerimisel. Olemasolevate rakenduste uuendamine ja juurutamine nõuavad küll lisakulutusi, kuid vähendavad võimalikke turvariske ning tagavad ettevõtetele kaasaegsed, mugavad ja töökindlad süsteemid.

Käesoleva bakalaureusetöö eesmärk on luua universaalne logimise rakendus, mis kogub süsteemi erinevatest rakendustest kokku tegevuslogid ja võimaldab kogutud logisid efektiivselt hallata. Rakendus luuakse anonüümseks soovinud jääda kliendile. Rakenduse universaalsus seisneb selles, et samasugust lahendust on võimalik kasutada ka teistes süsteemides ja tegevuslogisid saatvate rakenduste arv ei ole piiratud.

Töö esimeses peatükis selgitatakse vajaminevaid mõisteid ja termineid. Teises peatükis toob autor välja kliendi nõuded. Kolmas peatükk tutvustab erinevaid tehnoloogiaid, millega töö autor arendusprotsessi jooksul kokku puutus. Neljandas peatükis kirjeldab autor loodud lahendust. Lisa I on loodud rakenduse tegevuslogi valdkonnamudel. Lisa II demonstreerib sõnumi vastu võtmise teenust. Lisa III on lähtekood tegevuslogi salvestamiseks. Lisa IV demonstreerib klassi, mis võimaldab suhtlust andmebaasi, tagarakenduse ja kasutajaliidese vahel.

1. Mõisted ja terminid

Pistikmoodul (inglise k. *plugin*) – „riistvara- või tarkvaramoodul, mis on kergesti paigaldatav ja lisab senisele süsteemile võimalusi“ [1].

Silt – „dokumendi sisuelemendile lisatav vormingukood märgistuskeeltes“ [2].

Kodek – „andmeid või signaale kodeeriv ja dekodeeriv funktsionaalüksus“ [3].

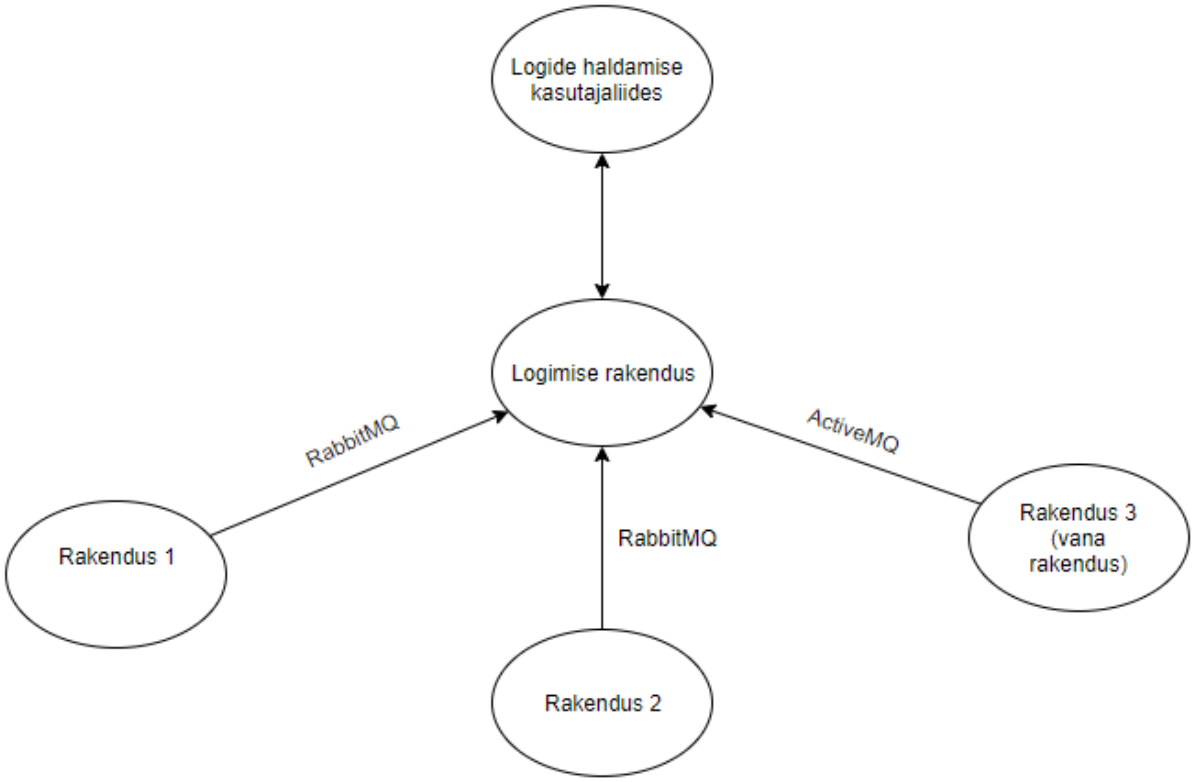
Utiliit – „tarkvarainstrument sageda abifunktsiooni täitmiseks“ [4].

Koodisüst – „programmi täitmise käiku muutva koodi salajane lisamine programmi koostisse“ [5].

API – „reeglid ja vahendid rakendusprogrammi suhtluseks“ [6].

2. Kliendi nõuded

Käesoleva ülesande teostamisel tuli kliendi vana süsteemi rakendust täiendada, mis salvestas andmeid juba olemasolevast süsteemist. Idee oli luua keskne rakendus logide haldamiseks nagu on kujutatud joonisel 1, kus rakenduste tegevuslogid saadetakse läbi sõnumikesksete vahevarade logimise rakendusse, läbi mille on võimalik kõiki logisid hallata. Kliendile oli väga oluline, et süsteemi uuele versioonile viimine oleks sujuv, seega rakendus peab toetama nii uutest rakendustest kui ka vanadest rakendustest saabuvaid logisid.



Joonis 1. Keskse logimise rakendusega süsteemi skeem.

Autori koostöö kliendiga sujus efektiivselt, alustades kliendi vajaduste välja selgitamisest ning arendustöö etappide kokku leppimisega. Algne idee ja nõuded lepiti kokku läbi internetikõne ning edaspidine suhtlus toimus kirja teel läbi kliendi *Rocket.Chat*'i platvormi implementatsiooni.

Järgnevalt on kirjeldatud kliendi poolt esitatud nõuded:

1. Olemasolev logimise rakendus on üle viidud Grails 4 platvormile.
2. Andmebaasis on tabel, mis hoiab tegevuslogisid:
    a. Tegevuslogid on iga kuu kaupa partitsioneeritud.
3. Vanast süsteemist saabuvad tegevuslogid salvestatakse uude ja vanasse andmebaasi.

4. Uuest süsteemist võetakse vastu tegevuslogid ning salvestatakse uude andmebaasi.
5. Administraatori vaade:
    a. Rakenduse ülemises menüüs on nupp nimega „Admin“, mis viib kasutaja administraatori vaatesse.
    b. Administraatori vaates on tabel, milles kuvatakse andmebaasis olevaid parameetreid.
    c. Administraatori vaates on tabel, läbi mille saab andmebaasis olevaid parameetreid hallata.
6. Tegevuslogi vaade:
    a. Rakenduse ülemises menüüs on nupp nimega „Admin“, mis viib kasutaja tegevuslogi haldamise vaatesse.
    b. Tegevuslogide vaates on tabel, kus kuvatakse tegevuslogisid uuest andmebaasist.
    c. Tegevuslogide vaates on tabel, läbi mille saab andmebaasis olevaid tegevuslogisid hallata.

3. Kasutatud tehnoloogiad

Töö autor oli tehnoloogiate valikutes piiratud põhjusel, et klient nõudis rakenduse loomiseks kindlate tehnoloogiate kasutamist. Kliendi süsteem hõlmab palju erinevaid rakendusi ning rakenduste tehnoloogiate ühtlustamise ja töökindluse eesmärgil on kasutatavad pistikrakendused saadaval läbi privaatse hoidla. Järgnev peatükk tutvustab lühidalt tehnoloogiaid, millega autor töö käigus kokku puutus.

3.1 Grails raamistik

Grails on veebiarenduseks loodud avatud lähtekoodiga raamistik. Raamistik põhineb Groovy programmeerimiskeelel - Java platvormile loodud objektorienteeritud programmeerimiskeel, mille eesmärgiks on parendada arendajate produktiivsust läbi lihtsa süntaksi [7, 8].

Groovy süntaks sarnaneb tugevalt Java süntaksiga ning on sujuvalt integreeritav Java programmidega. Seetõttu on võimalik Grails raamistikus kasutada nii Groovy kui ka Java programmeerimiskeeli ning nendes keeltes kirjutatud raamistikke [8].

3.1.1 Struktuur

Grails raamistiku projektid on range struktuuriga, see tähendab, et raamistik ei lase kindlat tüüpi failide asukohta määrata, vaid eeldab, et kõik failid on dokumentatsiooni põhjal kirjeldatud kaustades ja ka vastava failinimega. Järgnevas loetelus on selgitatud Grails raamistiku projekti struktuuri dokumentatsiooni [9] põhjal:

- ***grails-app*** - Projekti failide juurkaust.
    - ***conf*** - Konfiguratsioonifailide kaust, mis võimaldab käitusaegset konfiguratsiooni muuta.
    - ***controllers*** - Rakenduse kontrollerite kaust. Kontrollerid käitlevad päringuid ning loovad neile vastuseid.
    - ***domain*** - Rakenduse valdkonnamudelite kaust.
    - ***i18n*** - Kaust internatsionaliseerimiseks mõeldud failidele (i18n).
    - ***services*** - Rakenduse teenuste kaust. Grails raamistiku loojad ja haldajad ei poolda äriloogika kontrolleritesse paigutamist ning eeldavad, et äriloogika paigutatakse teenustesse.

    - ***taglib*** - Kohandatud siltide kaust. Grails võimaldab Groovy failidena luua kohandatud silte, mida on võimalik .gsp-faililaiendiga vaadete failides kasutada.
    - ***utils*** - Grailsile spetsiifilised utiliitide kaust. Kausta kasutatakse kustomiseeritud kodekite klasside hoidmiseks.
    - ***views*** - GSP (inglise k. *Groovy Server Page*) failide kaust. GSP on serveripoolne vaate viimistlemise tehnoloogia, mis põhineb *Groovy*'l. GSP failid on faililaiendiga ".gsp".
- ***src/main/groovy*** – Muude rakenduses kasutavate failide kaust.
- ***src/test/groovy*** – Üksuste ja integratsiooni testid.

Grails raamistikul loodud rakendusi on võimalik kiiresti ja vähese vaevaga kasutada. Range struktuuriga raamistik on eelseadistatud otsima kindlat tüüpi faile kindlatest kohtadest, mis omakorda vähendab aega, mis arendaja peaks kulutama rakenduse seadistamisele.

Autori poolt loodav rakendus järgib eelnevalt kirjeldatud struktuuri, millele oli veel lisatud kaustad *grails-app/rabbit-consumers* ja *config*.

3.1.2 GORM

GORM (inglise k. *Grails Object Relational Mapping*) on Grailsi objekt-relatsioonilise kaardistamise implementatsioon. Objekt-relatsiooniline kaardistamine (edaspidi ORM) võimaldab määrata objekti mudeli ja andmebaasi skeemi vahelist kaardistamist [8]. Käesolevas töös on objektideks Groovy failide klassid. Üheks tänapäeval populaarseimaks ORM raamistikuks on Hibernate, millel põhineb ka GORM [8].

3.1.2.1 Kaardistamine

GORM-i kasutamisel on vaja täpsustada objekti ja andmebaasi kaardistamist, mida tehakse *domain* kaustas olevate Groovy failide abil. Joonis 2 on näide lihtsast valdkonnamudelist, kus on defineeritud üks muutuja „name", mis vastab andmebaasis nimega samanimelisele väljale.

```
class Client {
    String name
}
```

Joonis 2. Näide lihtsast valdkonna mudelist

Vaikimisi eeldab GORM, et andmebaasi nimi on sama valdkonnamudeli klassi nimega [8]. Lisaks määrab GORM igale klassile väljad “id” ja “version” ilma, et arendaja seda valdkonnamudelis määrama peaks. GORM võimaldab ka automaatselt hallata valdkonnamudelite loomis- ja uuendamisaegu väljadel nimedega “dateCreated” ja “lastUpdated” vastavalt [8].

3.1.2.2 Püsivate objektide manipuleerimine

ORM raamistikud võimaldavad arendajal püsivaid objekte manipuleerida - luua, lugeda, uuendada ja kustutada. C. Richardson [10] on kirjutanud, et GORM kasutab objektide manipuleerimiseks lihtsamat lahendust kui teised ORM raamistikud. Tavapärased ORM raamistikud võimaldavad arendajal läbi mingi API objekti kasutada meetodeid objektide manipuleerimiseks (vt joonis 3).

```
public class HibernateSaveEntityExample
{
    public static void main(String[] args)
    {
        Session sessionOne = HibernateUtil.getSessionFactory().openSession();
        sessionOne.beginTransaction();

        //Loome uue Client objekti
        Client client = new Client();
        client.setClientId(1);
        client.setName("Tamm");

        //Salvestame client objekti
        sessionOne.save(client);

        sessionOne.getTransaction().commit();
        HibernateUtil.shutdown();
    }
}
```

Joonis 3. Näide objekti salvestamisest kasutades Hibernate raamistikku.

GORM lisab aga koodisüstiga objekti manipuleerimiseks vajalikud meetodid vastavasse valdkonnamudeli klassi. Seega saab meetodeid kasutada läbi valdkonnaobjekti klassi (vt joonis 4).

```
public class GORMSaveEntityExample
{
    public static void main(String[] args)
    {
        //Loome uue Client objekti
        Client client = new Client(name: "Tamm");

        //Salvestame client objekti
        client.save()

    }
}
```

Joonis 4. Näide objekti salvestamisest kasutades GORM-i.

Antud töö raames loodavas rakenduses kasutatakse GORM-i, et tagada kogu rakenduse suhtlus andmebaasiga. Nii rakendusse tulevate logide salvestamine kui ka läbi kasutajaliidese andmebaasiga suhtlemine on teostatud läbi GORM raamistiku. Autor leiab, et võrreldes teiste ORM-idega on GORM-i kasutamine väga intuitiivne.

3.2 JavaScript

JavaScript on interpreteeritud, objektorienteeritud keel, mis kasutab *first-class* funktsioone. Peamiselt tuntakse seda kui veebilehtede loomisel kasutatavat keelt ning keele süntaks sarnaneb meelega Java ja C++ programmeerimiskeeltele, et lihtsustada JavaScripti omandamist ja kasutamist [11].

Interpreteeritud keele omaduseks on programmi koodi tõlkimine iga kord, kui programmi jooksutatakse [12].

First-class funktsioonid on funktsioonid, mida on võimalik kasutada kui muutujat. Tänu sellele omadusele on võimalik näiteks funktsiooni parameeter või muutujale määratud väärtus asendada funktsiooniga [13].

Käesolevas töös kasutas autor ainult JavaScripti ja jQuery't, s.t. ei kasutatud JavaScript'i raamistikke. Raamistiku kasutamiseks ei olnud põhjust, sest kasutajaliidese peamiseks ülesandeks on tabelite kuvamine ja andmebaasiga suhtlemine läbi tabelite.

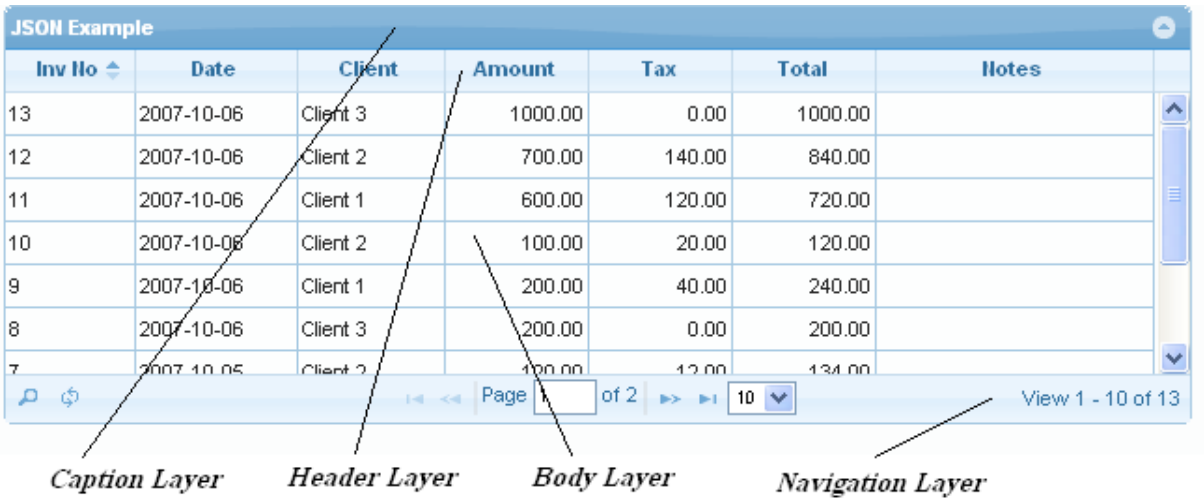
3.3 Ajax

Ajax on termin, mis kirjeldab erinevaid tehnoloogiaid, läbi mille luuakse asünkroonseid rakendusi [14]. Asünkroonsete rakenduse omaduseks on uuendada veebilehel kuvatavat infot ilma lehte uuesti laadimata.

3.4 JqGrid

JqGrid on JavaScriptile loodud pistikmoodul. Moodul on loodud veebilehel andmete kuvamiseks tabelina. Tabel koosneb järgnevast neljast põhilisest kihist (vt. joonis 5), mis on dokumentatsiooni [15] põhjal kirjeldatud järgnevalt:

1. Tabeli pealkirja kiht (inglise k. *Caption Layer*) - kuvab tabeli pealkirja.
2. Tulba pealkirja kiht (inglise k. *Header Layer*) - hoiab endas tulpade informatsiooni: pealkiri, laius, filter, sorteerimine *etc*.
3. Tabeli keha kiht (inglise k. *Body Layer*) - kuvab päritud andmed tulpadele vastavate mudelite põhjal.
4. Navigatsiooni kiht (inglise k. *Navigation Layer*) - kuvab lisainformatsiooni ja toiminguid.



Joonis 5. JqGrid tabeli kihtide struktuuri kirjeldus [15].

JqGrid kasutab Ajax päringuid, et saada informatsioon, mille põhjal luuakse tabel. Päritud informatsioon peab olema esitatud JqGrid'i dokumentatsioonis kirjeldatud tulba mudeli kujul. Järgnev on joonisel 6 kujutatud lihtsa JqGrid'i tabeli struktuuri päringu struktuuri kirjeldus dokumentatsiooni põhjal [15]:

- *colNames* – Tulpade pealkirjad tabelis kuvatavas järjekorras.

- *colModel* – Tulpade mudelid. Tulba mudelite järjekord vastab tulpade järjestusele tabelis. Tulpade pealkirjade ja mudelite arv peab olema sama.
    - *name* – Tulba kuvatav nimi.
    - *hidden* – Tõeväärtus, kas veerg kuvatakse tabelis või mitte.
    - *editable* – Tõeväärtus, kas veergu on võimalik muuta. Rea muutmisel või loomisel ei lisata mittetõese väärtusega tulpade andmeid andmebaasi päringusse.
    - *hidedlg* – Tõeväärtus, kas tulpa on võimalik tabelist peita.
    - *searchoptions* – Tulba filtri valikud.
        - *sopt* – Filtri operatsioonid – võrdub („eq“), ei võrdu („ne“), algab („bw“), ei alga („bn“) jne.
    - *width* – Tulba vaikelaius pikslites.
    - *template* – Tulba mall, ehk tulba elemendi sisu HTML-ina.
- *caption* – Tabeli pealkiri.

```
{
    "colNames":["Tulp 1 pealkiri","Tulp 2 pealkiri"],
    "colModel":[
        {
            "name":"tulp1"
            ,"hidden":false
            ,"editable":true
            ,"searchoptions": {
                    "sopt":["bw","bn","eq","ne","ew","en","nu","nn"]
            }
        },
        {
            "name":"tulp2",
            "hidden":false,
            "editable":false,
            "hidedlg":true,
            "template":"tulp2Template",
            "searchoptions": {
                    "sopt":["eq"]
            },
            "width":35
        }
    ],
    "caption":"Näidistabeli pealkiri"
}
```

Joonis 6. JqGrid'i tabeli struktuuri päringu näide

Käesolevas rakenduses ning ka kliendi teistes rakendustes kasutatakse tabelite kuvamiseks ja andmebaasi haldamiseks JqGrid'i pistikrakendust. JqGrid töötab kiiresti ka suuremate andmehulkade puhul ja pakub võimalust suurt osa tabelist kohandada vastavalt soovidele ning lisada funktsionaalsusi.

3.5 ActiveMQ ja RabbitMQ

ActiveMQ ja RabbitMQ on mõlemad sõnumikesksed vahevarad (inglise k. *Message-oriented middleware)*. Sõnumikeskne vahevara on tarkvara, mis võimaldab erinevatel tarkvara rakendustel omavahel suhelda, luues ühtse süsteemi. Sõnumikeskse vahevara üldine struktuur koosneb allikobjektist, tarbivast objektist ja järjekorrast. Allikobjektist edastatakse soovitud sõnum vahevara järjekorda, kus sõnum jääb ootama, kuni tarbija on sõnumi töötlemiseks valmis.

V. M. Ionescu [16] on kirjutanud, et suurimaks erinevuseks ActiveMQ ja RabbitMQ tarkvarade vahel on RabbitMQ võime luua ühendus erinevaid protokolle kasutavate platvormide vahel - e.g. MSMQ protokoll kasutades programmeerimiskeelt C# ja STOMP protokoll kasutades programmeerimiskeelt Ruby. Platvormid, mis kasutavad ActiveMQ'd, peavad mõlemad kasutama sama protokolli.

3.6 PostgreSQL

PostgreSQL on avatud lähtekoodiga objekt-relatsiooniline andmebaasisüsteem, mis kasutab SQL-i ja täiendab seda omapoolt lisatud teenustega [17]. PostgreSQL on olnud aktiivselt arenduses aastast 1986, kui California Ülikoolis Berkeleys alustati projektiga „POSTGRES", millest hiljem arenes välja PostgreSQL [17]. PostgresSQL on arendajate poolt hästi hinnatud tänu oma usaldusväärsusele, laialdastele lisateenustele, heale skaleeruvusele ja pidevatele uuendustele [17]. Eelmainitud põhjustel on ka antud töö klient otsustanud luua oma andmebaasid PostgreSQL-iga ja soovib ka logimisrakenduse andmeid hoida PostgreSQL andmebaasis.

3.7 Liquibase

Liquibase on avatud lähtekoodiga andmebaasi haldamiseks, versioneerimiseks ja muutuste rakendandamiseks loodud lahendus [18].

Andmebaasi haldamiseks ja muudatuste läbiviimiseks kasutatakse skripte nimega *changeSet*, mis antud bakalaureuse töös on kirjutatud Groovy failidena. Skriptid omakorda kirjeldavad milliseid SQL faile andmebaasi muudatuste läbiviimiseks kasutatakse. Liquibase jälgib rakendatud skriptide seisu ning võimaldab seeläbi ka andmebaasi versioneerimist [19].

3.8 Cron

Cron on UNIX-i utiliit, mis võimaldab jooksutada kasutaja poolt täpsustatud käske soovitud intervallidel. Töö autor kasutab antud projektis Cron'i läbi Quartz teegi. Quartz teek sisaldab Java klassi CronTrigger, mis võimaldab arendajal kasutada Cron'i planeerimisvõimalusi Javas [20].

4. Praktilise töö tulemus

Praktilise töö tulemusena uuendati kliendi endist logimisrakendust. Esmalt viidi vastavalt dokumenteeritud muudatustele [21] endine Grails 3 raamistikuga rakendus üle Grails 4 raamistikule, mis hõlmas endas peamiselt rakenduse konfiguratsiooni muutmist. Seejärel täiendati rakendust vastavalt kliendi nõuetele, et tagada ühilduvus uutest ja ka vanadest rakendustest tulevate logidega. Järgnevad peatükid kirjeldavad peamisi valminud tulemusi.

4.1 Andmebaasi loomine

Kliendiga koostöös lepiti kokku uue andmebaasi mudel sissetulevate logide jaoks. Tabeli loomiseks kasutati Liquibase'i pistikmoodulit. Uuele andmebaasi tabelile loodi ka vastav valdkonnamudel (Lisa I), et rakenduses saaks kasutada GORM raamistikku andmete salvestamiseks, lugemiseks ja muutmiseks.

```
CREATE TABLE log.LOG_ACTION (
    id                  BIGSERIAL PRIMARY KEY,
    version             BIGINT,
    application         VARCHAR(50)                 NOT NULL,
    action_time         TIMESTAMP WITH TIME ZONE    NOT NULL,
    action              VARCHAR(255)                NOT NULL,
    code_id             VARCHAR(30),
    node                VARCHAR(50),
    parameter           TEXT,
    output              TEXT,
    tab                 VARCHAR(50),
    entity_id           BIGINT,
    entity              VARCHAR(50),
    ip                  VARCHAR(20)
);
```

Joonis 7. Tegevuslogide andmebaasi loomine

Rakendus kogub andmeid mitmelt erinevalt rakenduselt, mistõttu on andmebaasi loodud automaatne välja „id" määramine. Väli „id" täidetakse automaatselt järgmise kõige suurema arvuga, mida andmebaasis ei esine.

4.2 Logide vastuvõtmine ja salvestamine

Käesoleva töö üheks osaks oli teistest rakendustest logide vastuvõtmine läbi sõnumikesksete vahevarade RabbitMQ ja ActiveMQ. Rakendusel oli varasemalt olemas teenus, mis võttis läbi ActiveMQ kliendi vanast süsteemist vastu logisid ning salvestab need kliendi vanasse andmebaase. Autor lõi uue RabbitMQ sõnumeid vastu võtva teenus (Lisa II), mis asub kaustas *grails-app/rabbit-consumers*. RabbitMQ teenuses võetakse vastu sissetulev logi JSON-i formaadis ning salvestatakse (Lisa III) kasutades GORM raamistikku andmebaasi.

Sissetulevas sõnumis on JSON-iga kirjeldatud kõik „LOC_ACTION“ tabelis (vt. joonis 7) määratletud väljad peale „id“ ja „version“ väljade, mis lisatakse andmebaasi GORM-i poolt andmete salvestamisel automaatselt.

```
{
    "application"    : "application",
    "action_time"    : "2012-04-21T18:25:43-05:00",
    "action"         : "action",
    "code_id"        : "code_id",
    "node"           : "node",
    "parameter"      : "parameter",
    "output"         : "output",
    "tab"            : "tab",
    "entity_id"      : "entity_id",
    "entity"         : "entity",
    "ip"             : "127.0.0.1"
}
```

Joonis 8. Lihtne näide läbi RabbitMQ teenuse tulevast sõnumist.

Praktilise töö loomise ajal esines RabbitMQ pistikmoodulis viga - kasutajal ei piisa rakenduse tööle saamiseks ainult välist konfiguratsiooni *application.yml* failis muuta, vaid tuleb muuta ka järjekorra nimi (inglise k. *queue name*) teenuse klassis.

4.3 Kasutajaliides

Antud töö käigus valmis lihtne kasutajaliides, mis võimaldab kasutajal hallata logisid ja parameetreid.

Joonis 9. Parameetrite tabel

Joonisel 9 on kujutatud parameetrite tabelit. Tabel on loodud kasutades JqGrid'i pistikmoodulit. Tabel võimaldab hallata olemasolevaid parameetreid, mis asuvad rakenduse andmebaasis, aga ei võimalda kasutajal lisada uusi parameetreid. Lisaks toetab tabel kõikide tulpade kaudu sorteerimist ja „Nimetus" väljal ka filtreerimist. Analoogne tabel on lisatud ka tegevuslogide jaoks, kus on võimalik kõiki väljasid sorteerida ja filtreerida.

Joonis 10. Lehekülje ülemine menüü

Joonisel 10 on kujutatud lehe ülemine menüü. Lisatud on nupp nimega „TEG LOG", mis viib kasutaja tegevuslogide haldamise tabeli vaatesse. Nupp „Administreerimine" avab enda alla menüü, kust on võimalik valida kas „Parameetrid" või „Kasutaja vaikeseaded". „Parameetrid" nupp viib kasutaja parameetrite tabeli vaatesse. „Kasutaja vaikeseaded" vaade hetkel puudub.

4.4 Testimine

Käesoleva töö eesmärk on luua kliendile algne funktsionaalne logimise rakendus, mida hiljem arendatakse vastavalt vajadusele edasi. Kuna kliendi süsteem rakendused on pidevalt arenduses, ei ole lõputöö kirjutamise hetkel testimise tulemused saadaval. Rakendus testitakse kliendi poolt kui kõik vajalikud esialgsed funktsionaalsused on valmis. Testimise peamine eesmärk on veenduda kuidas logimisrakenduse sõnumikesksed vahevarad on konfigureeritud korrektselt ning kas andmed salvestatakse õigesti.

Töö autor leidis, et suurimaks väljakutseks oli kliendi rakenduste ülesehitustest, olemasolevast koodist ja isetehtud pistikmoodulitest aru saamine. Kuigi autori jaoks uute tehnoloogiate õppimine ja kasutamine ei tundunud tihti dokumentatsiooni põhjal raske, siis olemasolevasse keerukasse süsteemi uute funktsionaalsuste lisamine muutis ülesande keerukaks.

5. Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli luua universaalne logimise rakendus, mis koguks süsteemi erinevatest rakendustest kokku tegevuslogid ja võimaldaks kogutud logisid efektiivselt hallata. Universaalne rakendus on loodud klienditellimusena, seetõttu on lähteülesandel arvestatud eelkõige kliendi vajadustest. Rakenduse loomisel tuli arvestada, et samasugust lahendust oleks võimalik kasutada ka teistes süsteemides ja tegevuslogisid saatvate rakenduste arv ei oleks piiratud. Samuti tuli arvestada kliendi vana süsteemi rakendusega, mis salvestas andmeid juba olemasolevast süsteemist. Uuele süsteemile sujuva ülemineku tagamiseks oli oluline, et rakendus toetaks nii uutest rakendustest, läbi RabbitMQ tulevaid logisid, kui ka vanadest rakendustest, läbi ActiveMQ tulevaid logisid.

Töö autor kasutas rakenduses GORM-i, et tagada kogu rakenduse suhtlus andmebaasiga. Nii rakendusse tulevate logide salvestamine kui ka läbi kasutajaliidese andmebaasiga suhtlemine toimub läbi GORM raamistiku. Antud töös on kasutatud ainult JavaScript'i ja jQuery't, s.t ei kasutatud JavaScript'i raamistikke. Autor kasutas JqGrid'i, mis töötab piisavalt kiiresti ka suuremate andmehulkade puhul ja pakub võimalust suurt osa tabelist kohandada vastavalt soovidele ning lisada funktsionaalsusi.

Töö käigus ei saanud kõik kliendi esitatud nõuded täidetud, seega jätkub peale töö esitamist autori poolt rakenduse arendus edasi. Töö raames jäi implementeerimata automaatne tegevuslogide tabeli partitsioneerimine kuu kaupa, kasutades Cron utiliiti.

Töö lõpptulemusena valmis rakendus andmete logimiseks, mis kogub andmeid kahe erineva sõnumikeskse vahevara kaudu. Lisaks loodi ka lihtne kasutajaliides tegevuslogide ja parameetrite andmebaasi tabeli haldamiseks.

6. Viidatud allikad

[1] *Standardipõhine Tarkvaratehnika Sõnastik.* https://stats.cyber.ee/term/search?utf8=%E2%9C%93&lang=2&q=plugin. (29.03.2020)

[2] *Andmekaitse ja Infoturbe Leksikon*. https://akit.cyber.ee/term/4981-tag-1. (29.03.2020)

[3] *Andmekaitse ja Infoturbe Leksikon*. https://akit.cyber.ee/term/5847-codec (13.04.2020)

[4] *Andmekaitse ja Infoturbe Leksikon*. https://akit.cyber.ee/term/3926-utility-1 (13.04.2020)

[5] *Andmekaitse ja Infoturbe Leksikon*. https://akit.cyber.ee/term/562-code-injection (13.04.2020)

[6] *Andmekaitse ja Infoturbe Leksikon*. https://akit.cyber.ee/term/3088-api (13.04.2020)

[7] *Grails*. https://grails.org/ (13.04.2020)

[8] *Groovy*. https://groovy-lang.org/ (13.04.2020)

[9] Grails Documentation. *Grails.* https://grails.org/documentation.html (13.04.2020)

[10] Richardson, C. (2009) 'ORM in Dynamic Languages', Communications of the ACM, 52(4), lk 48–55

[11] About JavaScript. *MDN Web Docs.* https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript (29.03.2020)

[12] Compiled versus interpreted languages. *IBM Knowledge Center* https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zappldev/zappldev_85.html (29.03.2020)

[13] First-class Function. *MDN Web Docs* https://developer.mozilla.org/en-US/docs/Glossary/First-class_Function (29.03.2020)

[14] Ajax. *MDN Web Docs* https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX (15.04.2020)

[15] How it Works. *JqGrid Wiki* http://www.trirand.com/jqgridwiki/doku.php?id=wiki:how_it_works (15.04.2020)

[16] V. M. Ionescu, "The analysis of the performance of RabbitMQ and ActiveMQ," *2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER)*, Craiova, 2015, lk 132-137.

[17] About. *PostgreSQL*https://www.postgresql.org/about/ (29.04.2020)

[18] How Liquibase Works. *Liquibase.* https://www.liquibase.org/get_started/how-lb-works.html (29.04.2020)

[19] S. Shmeltzer Introduction to Liquibase and Managing Your Database Source Code, *Shay Shmeltzer's Oracle Development Tools Tips*, 2017. https://blogs.oracle.com/shay/introduction-to-liquibase-and-managing-your-database-source-code (29.04.2020)

[20] CronHowTo. *Ubuntu.* https://help.ubuntu.com/community/CronHowto

[21] Upgrading from Grails 3.3.x. *Grails*. https://docs.grails.org/latest/guide/upgrading.html (09.08.2020)

Lisad

I. Tegevuslogi valdkonnamudel

```
package ee.smit.hklog

import ee.smit.core.repository.domain.DomainClass

class LogAction implements DomainClass {
    Long id
    String application
    Date actionTime
    String action
    String codeId
    String node
    String parameter
    String output
    String tab
    Integer entityId
    String entity
    String ip

    static constraints = {
        id nullable: true
        application nullable: false
        actionTime nullable: false
        action nullable: false
        codeId nullable: true
        node nullable: true
        parameter nullable: true
        output nullable: true
        tab nullable: true
        entityId nullable: true
        ip nullable: true
    }

    static mapping = {
        table name: 'log_action', schema: 'log'
        id generator: 'sequence', params: [sequence:
'log.log_action_id_seq']
    }

    @Override
    boolean isNew() {
        return false
    }
}
```

II. RabbitMQ logi vastu võtmise teenus

```
package ee.smit.hklog

import com.budjb.rabbitmq.consumer.MessageContext
import groovy.util.logging.Slf4j

@Slf4j
class SosLogPrimary1Consumer {

    /**
     * Peavad olema sama mis välises konfiguratsioonis
     */
    private static final String CONNECTION_NAME = 'primary1'
    private static final String QUEUE_NAME = 'LogIn'

    /**
     * NB! Väline keskne konfiguratsiooni ei tööta
(https://github.com/budjb/grails-rabbitmq-native/issues/134),
     * mistõttu laadime välisest konfiguratsioonist väärtused ka siia
     */
    static rabbitConfig = [
            connection: CONNECTION_NAME,
            queue     : QUEUE_NAME
    ]

    SosLogRabbitmqInService sosLogRabbitmqInService

    void handleMessage(String jsonMessage, MessageContext messageContext) {
        sosLogRabbitmqInService.processMessage(jsonMessage)
    }
}
```

III. RabbitMQ kaudu logi salvestamise teenus

```
package ee.smit.hklog


import grails.converters.JSON
import grails.gorm.transactions.Transactional
import org.grails.web.json.JSONObject

@Transactional
class SosLogRabbitmqInService {

    LogActionSaveService logActionSaveService

    void processMessage(String jsonMessage) {
        try {
            log.info("Saabus tegevuse logi: ${jsonMessage}")
            JSONObject json = JSON.parse(jsonMessage)

            if (log.infoEnabled) {
                String logMessage = "Algas tegevuslogi salvestamine; json:
${json}"

                if (log.isDebugEnabled()) {
                    log.info("${logMessage} ${json}")
                } else {
                    log.info(logMessage)
                }
            }

            LogAction logAction
            try {
                logAction = logActionSaveService.save(json)
            }
            catch (e) {
                log.error("Viga tegevuslogi salvestamisel: ${jsonMessage}", e)
            }

        } catch (Exception e) {
            log.error("Viga tegevuslogi töötlemisel: ${jsonMessage}", e)
        }
    }
}
```

IV. Tegevuslogide tabeli kontroller

```
package ee.smit.hklog.ui

import ee.smit.core.RequestUtil
import ee.smit.core.exception.NoModificationsException
import ee.smit.core.exception.ValidationExceptionHandler
import ee.smit.core.rest.listeners.RESTExceptionHandler
import ee.smit.grid.enums.JqGridOperationEnum
import ee.smit.grid.model.JqGridModel
import ee.smit.grid.model.builder.JqGridModelBuilder
import ee.smit.grid.model.builder.column.JqGridColumnModelBuilder
import ee.smit.grid.model.builder.column.JqGridColumnModelExtension
import ee.smit.grid.query.command.JqGridQueryRequest
import ee.smit.grid.traits.JqGridSaveControllerTrait
import ee.smit.hklog.LogAction
import ee.smit.hklog.LogActionJsonMapperBootStrap
import ee.smit.hklog.LogActionSaveService
import ee.smit.hklog.LogActionUIQueryService
import ee.smit.hklog.UserRoleEnum
import grails.compiler.GrailsCompileStatic
import grails.converters.JSON
import grails.plugin.springsecurity.SpringSecurityUtils
import grails.plugin.springsecurity.annotation.Secured
import org.grails.web.json.JSONObject
import org.springframework.http.HttpStatus

@SuppressWarnings(['MethodReturnTypeRequired'])
@GrailsCompileStatic
class LogActionGridViewController implements JqGridSaveControllerTrait,
ValidationExceptionHandler, RESTExceptionHandler {

    static defaultAction = 'list'

    //Lehe kuvamine
    def list() {
        log.info('Kasutaja avas tegevuslogide tabi')
        render(view: '/logActionGridView/list')
    }

    //Tabelis vaikimisi nähtavad väljad
    private List<String> DEFAULT_VISIBLE_COLUMNS_IN_ORDER = [
            'id',
            'application',
            'action',
            'codeId',
            'node',
            'parameter',
            'output',
            'tab',
            'entityId',
            'entity',
            'ip',
    ]

    //Tabeli väljad mida ei kuvata
    private Set<String> SKIPPED_FIELD_LIST = [

    ] as Set<String>

    //Tabelis muudetavad väljad
```

```
    private Set<String> EDITABLE_FIELDS = [

    ] as Set<String>


    LogActionSaveService logActionSaveService
    LogActionUIQueryService logActionUIQueryService

    // Tabelisse andmete pärimine
    def dataJSON(JqGridQueryRequest searchCommand) {
        List<LogAction> queryResult =
logActionUIQueryService.doQuery(searchCommand)
        Map pagingMap = searchCommand.createPagingMap(queryResult)
        render pagingMap as JSON
    }

    //Andmete mudeli päring
    def gridModelJSON() {
        render(getJqGridModel() as JSON)
    }

    //Mudel
    private JqGridModel getJqGridModel() {
        return getJqGridModelBuilder().build()
    }

    //Tabeli kõikide väljade nimed
    Set<String> getAllFieldNamesForGrid() {
        return LogActionJsonMapperBootStrap.logActionMap(new
LogAction()).keySet() as Set<String>
    }

    //Tabelist salvestamine
    @Secured(closure = {
        SpringSecurityUtils.ifAnyGranted(UserRoleEnum.ROLE_LOG_ADMIN.name())
    })
    def saveFromGrid() {
        JSONObject json = isValidJson(RequestUtil.getJSONFromRequest(request))
        log.info("Kasutaja algatas Tegevuslogi ${json.id} salvestamise
veebivormilt")
        JqGridOperationEnum operationEnum = findJqGridOperation(json)

        json = initializeJsonForSaving(json, false)

        Map result = [:]

        try {
            result.logAction = logActionSaveService.saveByJson(json) as
LogAction
            result.message = 'OK'
        }

        catch (NoModificationsException nmEx) {
            log.debug("Ignoreerime UI'st muutes NoModificationException'it,
Tegevuslogi ${json.id} salvestamisel")

            result.message = nmEx.message
            result.logAction = nmEx.domainEntity as LogAction
        }

        result.status = HttpStatus.ACCEPTED.value()
```

```
        render(result as JSON)
    }

    //Mudeli loomine
    private JqGridModelBuilder getJqGridModelBuilder() {
        JqGridColumnModelBuilder columnModelBuilder = new
JqGridColumnModelBuilder(LogAction)
                .setViewOnlyMode(false)
                .setEditableOnlyFields(EDITABLE_FIELDS)

        JqGridModelBuilder gridModelBuilder = new
JqGridModelBuilder(LogAction, columnModelBuilder)
                .setAllFieldNames(getAllFieldNamesForGrid())

.setDefaultVisibleColumnsInCorrectOrder(DEFAULT_VISIBLE_COLUMNS_IN_ORDER)
                .addSkippedFieldNames(SKIPPED_FIELD_LIST)
        return gridModelBuilder
    }
}
```

V. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, **Cristian Noop**,

(autori nimi)

1. Annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose **Universaalse logimisrakenduse loomine**,

   (lõputöö pealkiri)

mille juhendaja on **Mirjam Paales**,

(juhendaja nimi)

reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Cristian Noop

10.08.2020