TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Svetlana Ušakova 204203IAPM

Digitaalse arhiivilahenduse kavandamine ja realiseerimine

Magistritöö

Juhendaja: Erki Eessaar

PhD

Tallinn 2022

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: Svetlana Ušakova

03.05.2022

Annotatsioon

Tänapäeval on digitaalsete arhiveerimist vajavate andmete hulk väga suur ning andmemahu kasv kiireneb. Digitaalsete andmete säilitamisele ei ole ühtset lähenemisviisi. Peaaegu igas organisatsioonis tekib säilitamist vajavaid konfidentsiaalseid andmeid. Organisatsioon peab otsustama, kas realiseerida arhiiv täielikult organisatsiooni sees või kasutada juba mõnda olemasolevat teenusepakkuja pakutavat lahendust.

Käesoleva töö eesmärgiks on disainida ja realiseerida üldine arhiivilahendus, mis vastab kõikidele ettevõtte X poolt esitatud nõuetele ning kasutab tänapäevaseid tehnoloogiaid ja praktikaid. Loodud lahendus peab olema skaleeritav, laiendatav ja idempotentne. Süsteem peab võimaldama kontrollida, et arhiveeritud andmeid pole muudetud. Loodav lahendus peab kasutama PostgreSQL andmebaasi, toetama JSONL andmete formaati ning tulema toime konkreetse andmete struktuuriga.

Töö näol on tegemist disaini tegevusuuringuga. See tähendab, et luuakse tehniline tehis, võetakse see kasutusele, jälgitakse kasutamist ning tehakse selle alusel täiendusi. Töö tulemusena loodi arhiivi lahendus, mis võimaldab salvestada ja säilitada JSONL formaadis sündmuste pakke. Iga sündmuste pakk sisaldab andmeid ühe või mitme sündmuse kohta. Andmete muutumatuse tagamiseks kasutatakse plokiaheldust. Loodud rakendus on realiseeritud mikroteenuste arhitektuuri põhjal ning selleks kasutatakse Micronaut raamistikku. Mikroteenuste omavaheline suhtlemine toimub asünkroonse sõnumivahetuse kaudu, mille jaoks kasutatakse Apache Kafkat. Andmebaasisüsteemiks on PostgreSQL ning andmeid hoitakse andmebaasis JSONB tüüpi veergudes.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 90 leheküljel, 9 peatükki, 50 joonist, 16 tabelit.

Abstract

Design and Implementation of a Digital Archive Solution

Nowadays the amount of digital data that needs to be archived is very large and the volume of data is growing in a rapid rate. Almost every organization has confidential data that needs to be securely archived. There is no single approach to digital data preservation. It is up to the organization to decide as to whether the archive will be fully implemented within the organization or whether the services of external parties will be used.

An organization X requires an archive solution to store organizational data. The main goal of the work is to implement a general archival solution that meets all the requirements of the organization X. The solution must meet such requirements as flexibility, scalability, ease of maintenance, data integrity, and idempotence. The solution must make it possible to check that the archived data is not updated. The solution must support JSONL data format and use PostgreSQL database for storing the data.

As a result of the work, a software has been created for receiving data in JSONL format and storing it in a database. The archive application is implemented by using the microservice architecture. It consists of four independent microservices: *Publisher, Loader, Manager,* and *Jobs*. *Publisher* receives events from a client and publishes these to a queue. *Loader* hashes and stores events in the database. *Jobs* checks data immutability. *Manager* manages clients and permissions of clients as well as creates and register databases and schemas. The application is developed in Java by using Micronaut framework. Communication between the microservices is implemented by using asynchronous messaging with Apache Kafka. Events with metadata are stored in a PostgreSQL database. The received events are stored in the columns that have JSONB type. The event data is chained by using SHA-256 hasing algorithm to ensure data immutability. The archive verifies the immutability of the stored data and saves verification results.

The thesis is in Estonian and contains 90 pages of text, 9 chapters, 50 figures, 16 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakendusliides
CASE	*Computer-aided software engineering*, modelleerimisvahend
DICOM	*Digital Imaging and Communications in Medicine*
HDF	*Hierarchical Data Format*
HTTP	*Hypertext Transfer Protocol*, hüperteksti edastusprotokoll
HTTPS	*HTTP over SSL, Hypertext Transfer Protocol Secure*, hüperteksti edastusprotokoll üle turvasoklite kihi
JSON	*JavaScript Object Notation*, vorming andmete esitamiseks ja vahetamiseks
JSONB	*JSON Binary*, kahendkujul JSON, andmetüüp PostgreSQLis
JSONL	*JSON Lines*, formaat andmete tekstilisel kujul kirjete kaupa esitamiseks
JVM	*Java Virtual Machine*, Java virtuaalmasin
MARS	*Multi-Agent Research and Simulation*
NAS	*Network Attached Storage*, võrgumälu
Plokk	Antud töö mõttes andmebaasi kirje
REST	*Representational State Transfer*, veebirakenduste tarkvaraarhitektuuri laad
RPC	*Remote Procedure Call*, kaugprotseduuri väljakutse
SHA	*Secure Hash Algorithm*, turvaline räsialgoritm
SHA-256	256 bittine SHA-2 perekonda kuuluvad räsifunktsioon
SQL	*Structured Query Language*, populaarne andmebaasikeel
SSL	*Secure Sockets Layer*, informatsiooni turvalisuse tagamiseks kasutatav protokoll

Sündmuste pakk	Kogum ühte tüüpi sündmuseid, mille andmeid tuleb süsteemis arhiveerida
Töö spetsifikatsioon	Kirjeldus, milliste sündmuste andmete muutumatust tuleb kontrollida
Töö	Konkreetsete sündmuste muutumatuse kontrollimine
UML	*Unified Modeling Language*, paljude erinevate valdkondade modelleerimiseks kasutatav standardiseeritud modelleerimiskeel, mille abil luuakse visuaalseid mudeleid e diagramm e skeem
URL	*Uniform Resource Locator*, internetiaadress
UTC	*Universal Time Coordinated*, maailmaaeg

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tänapäeval toimub pidev andmevahetus erinevate süsteemide vahel, arvutisüsteemides säilitatavate andmete hulk on väga suur ning andmemahu kasv kiireneb. Peaaegu igal organisatsioonil on konfidentsiaalseid andmeid, mida peab turvaliselt arhiveerima.

Digitaalsete andmete säilitamine tähendab tegevuste jada, mis tagab pideva tõrkekindla juurdepääsu digitaalsetele andmetele teatud perioodi jooksul [1]. Tavaliselt koosnevad arhiiviandmed vanematest andmetest, mis on organisatsiooni jaoks olulised või mille säilitamist nõuavad konkreetse riigi seadused, kuna erinevates riikides ja tööstusharudes kehtivad tavaliselt erinevad eeskirjad andmete säilitamise, sh nende säilitamise aja kohta.

Digitaalsete andmete säilitamisele ei ole ühtset lähenemisviisi. Organisatsioon peab ise otsustama, kas arhiiv realiseeritakse täielikult organisatsiooni sees või kasutatakse selleks väliste osapoolte teenuseid. Mõlemal variandil on oma puudused ja eelised. Näiteks selleks, et luua arhiivi organisatsiooni siseselt, peab organisatsioonis olema piisavalt vahendeid ning erialaoskustega ja kogemustega töötajaid. Organisatsioon peab kogu arhiivi kasutamise vältel tagama ise selle hoolduse ja ülalhoiu, mis on täiendav kulu. Samas on valmimise järel selline lahendus teenusepakkujate suvast sõltumatu ning vastab kõikidele nõuetele. Väliste osapoolte teenuste kasutamisega võivad esineda suhtluse-, turvalisuse- ja usaldusprobleemid ning täieliku kontrolli puudumine, mis on väga tähtis, kui arhiivis hoitakse tundlikke andmeid. Samuti võib see olla ettenähtamatult kallis ja hinnatõusud pole ennustatavad. Näiteks pilveteenuste puhul on raske ette näha kulusid, sest sageli pole täpselt teada, kui palju andmeid peab säilitama ja kui suures mahus teenust on selleks vaja.

1.1 Taust ja probleem

Ettevõte X loob tarkvara, platvormi ja terviklahendusi maailma suurimatele teenuspakkujatele ning pakub teenuseid rohkem kui 150-le tuntud brändile, mille klientideks on sajad miljonid kasutajad üle maailma.

Platvormi kasutamisel tekkinud andmete säilitamiseks vajab ettevõte digitaalset arhiivi. Ettevõttel on vaja arhiveerida erinevates riikides tekkinud andmeid. Igas riigis on omad regulatsioonid – seadused ja piirangud andmete töötlemisele ja säilitamisele. Seega peab olema võimalik kohandada arhiivi omadusi sõltuvalt riigist, mille andmeid seal hoitakse.

Ettevõtte X jaoks realiseeritud arhiivilahendus peab vastama sellistele nõuetele nagu paindlikkus, horisontaalne skaleeritavus, hoolduse lihtsus, andmete muutumatus ja aheldamine, SQL-andmebaasi ja JSON andmete esitamise formaati kasutamine. Sõltuvalt teenusepakkujast võib olla vaja arhiivi puhul kasutada lisateenuseid. Üheks selliseks lisateenuseks on andmete muutumatuse kontroll.

Iga riik määrab oma piirangud andmete hoidmisele ning riikide nõuded andmete säilitamisele võivad aja jooksul muutuda. Seega võib tekkida vajadus hakata arhiivis pakkuma uusi lisateenuseid, mida mõne riigi andmete puhul on vaja ja teiste puhul ei ole vaja. See tähendab, et arhiivi rakenduse funktsionaalsed nõuded võivad tulevikus muutuda ning arhiivi pidajal peab olema võimalik neid muudatusi jooksvalt teostada. Selleks peab lahenduse arhitektuur olema piisavalt paindlik. Samuti võib tekkida vajadus hakata hoidma arhiivis uute riikide andmeid või eemaldada mingi riigi andmed. Arhiivilahendus peab võimaldama salvestada erinevates riikides tekkinud andmeid erinevatel serveritel ja paigutada need serverid vajadusel geograafiliselt erinevatesse asukohtadesse – näiteks nii, et iga riigi arhiivi andmed on füüsiliselt selle riigi territooriumil.

Etteantud nõuetele vastav arhiivilahendus puudub. Töö eesmärgiks on pakkuda välja ja realiseerida selline arhiivilahendus. Lahendus töötatakse välja ühe ettevõtte soove silmas pidades, kuid tulemus võib sobida kasutamiseks ka teistele rahvusvahelistele teenusepakkujatele.

1.2 Töö kirjeldus

Töö peamine eesmärk oli disainida ja realiseerida üldine arhiivilahendus, mis vastab kõikidele ettevõtte poolt esitatud nõetele, kasutades tänapäevaseid tehnoloogiaid ja praktikaid. Töö sisuks on loodava süsteemi analüüsimine e selle kohta nõuete kogumine, rakenduse ning andmebaasi disainimine ja realiseerimine, tulemuste valideerimine ja tulemuste analüüsimine.

Antud töö tulemusena luuakse arhiivilahendus. Arhiivilahendus on mõeldud kasutamiseks ettevõte sees, kuna see on tihedalt seotud ettevõte teiste süsteemiga, mis täiendavad loodud süsteemi. Näiteks süsteemi kasutajaliidesed on realiseeritud eraldi rakendusena, mida antud töös ei kirjeldata. Selleks, et süsteemi oleks võimalik kasutada iseseisva süsteemina väljaspool ettevõtet, peab lahendust kohandama ja täiendama. Selliste laienduste kavandamine ja realiseerimine ei ole käesoleva töö eesmärk.

2 Metoodika

Käesolev peatükk annab ülevaade töö objektist, töö protsessist ja kasutatud tööriistadest.

2.1 Ülevaade objektist

Arhiivirakendus on loodud ettevõte X jaoks. Ettevõte X valmistab tarkvara erinevatele teenuspakkujatele üle maailma. Ettevõtte X on suurettevõte, mille nime ja konkreetset tegevust konfidentsiaalsuse nõudest lähtuvalt ei avaldata.

2.2 Ülevaade töö protsessist

Töö esimese sammuna leiti funktsionaalsed ja mittefunktsionaalsed nõuded. Nende alusel disainiti ja realiseeriti antud töös kirjeldatud arhiivilahendus. Arendustöö toimus kahenädalaste iteratsioonidena, kasutades agiilset tarkvara arendusmetoodikat Scrum. Kogu arendus võttis aega umbes 1.5 aastat.

Töö näol on tegemist disaini tegevusuuringuga (*action design research*), mis tähendab tehnilise tehise (arhiivilahenduse) arendamist ning samaaegselt kasutusele võtmist ning kasutamise jälgimise tulemuste ja tagasiside alusel tehise täiendamist [2].

See tähendab, et kõigepealt loodi esialgne lahenduse versioon, mida käivitati ja testiti lokaalses masinas koodi testide abil. Seejärel võeti algne lahendus kasutusele ning parandati leitud vead ja täiendati lahendust.

2.3 Tööriistade kirjeldus

Rakenduste loomiseks on olemas palju erinevaid programmeerimiskeeli, raamistikke ja tööriistu. Antud rakenduse jaoks valiti kasutamiseks Micronaut raamistik. Micronaut on 2022. aasta seisuga suhteliselt noor avatud lähtekoodiga JVM-põhine raamistik, mis sobib mikroteenustel põhinevate rakenduste ehitamiseks ning toetab erinevaid programmeerimiskeeli, sealhulgas ka Javat [3]. Süsteemi ehitamiseks ja haldamiseks kasutati projektijuhtimise tööriista Maven. Mikroteenused suhtlevad omavahel kasutades asünkroonset sündmusvoo töötlemist võimaldavat vahendit Apache Kafka. Sündmuste

andmete valideerimiseks e reeglitele vastavuse kontrollimiseks kasutatakse Bean Validation raamistiku abi.

Andmebaasi disainimiseks ja olemi-suhte diagrammide ning andmebaasi füüsilise disaini diagrammide koostamiseks kasutati CASE tarkvara Enterprise Architect. Süsteemi modelleerimiseks kasutatakse visuaalset standardiseeritud modelleerimiskeelt UML – loodi paketidiagramme, kasutusjuhtude diagramme, olemi-suhte diagramme ning andmebaasi disaini diagramme (klassidiagrammidena) ja jadadiagramme. Rakenduse testimiseks kirjutati ühiktestid JUnit raamistiku abil. Päringute saatmiseks ja rakenduse töö testimiseks kasutati Postmani. Andmebaasi skeemide migreerimiseks ja muudatuste haldamiseks kasutati avatud lähtekoodiga vahendit Liquibase ning andmebaasisüsteemiks on avatud lähtekoodiga PostgreSQL.

Rakenduse koodi hoidmiseks ja versioonide haldamiseks kasutatakse Githubi hoidlat. Vahendite valikut põhjendatakse pikemalt jaotises 7.1.2.

2.4 Tööjaotus

Antud lahenduse mudelid ja lähtekood on loodud autori poolt. Kuid nagu igas suurettevõttes tavaks, siis arutati kõik võimalikud lahendused, arhitektuur ja tehnoloogiad läbi meeskonna juhiga või mentoriga.

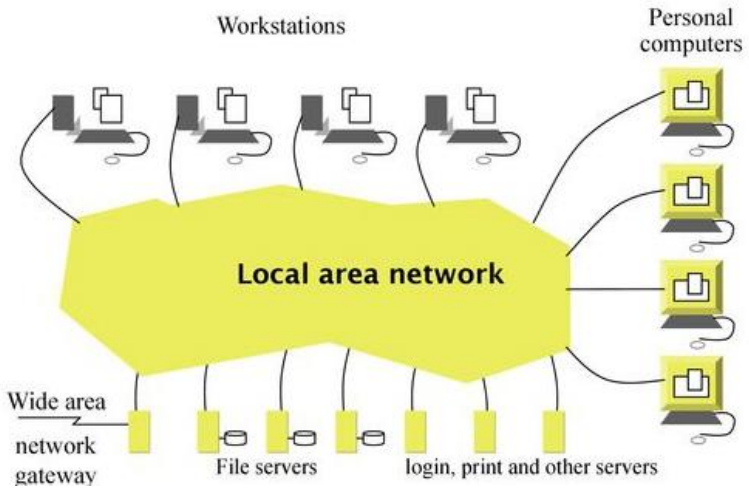
3 Teoreetiline taust

Selles peatükis antakse lühiülevaade töös kasutatavaid mõistetest, mille tundmine on kirjatöö ja realiseeritud süsteemi mõistmiseks vajalik.

3.1 Hajussüsteemid

Hajus arvutisüsteem (edaspidi hajussüsteem) on arvutivõrku ühendatud autonoomsete arvutite kogum, mis on varustatud vajaliku tarkvaraga (vt **Joonis 1**). Hajussüsteemi elemendid suhtlevad ja koordineerivad toiminguid, et süsteem paistaks lõppkasutajate jaoks kui mittehajus e tsentraliseeritud süsteem.

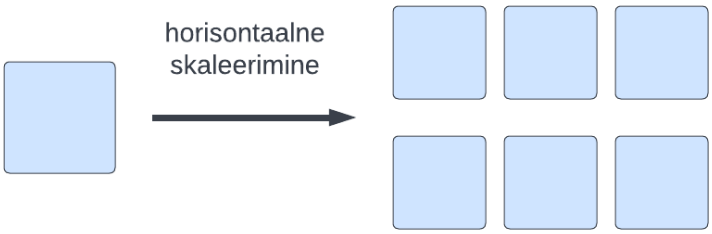
Hajussüsteemi erinevad sõlmed käituvad üksteisest sõltumatult ning võivad paikneda samas kohas või olla füüsiliselt eraldatud ja isegi erinevates geograafilistes asukohtades. Tavaliselt loovad sõlmed ühenduse võrguga, omavad kohalikku mälu ning suhtlevad sõnumite teel. [4]



Joonis 1 Hajussüsteemid [5]**.**

Hajussüsteemide tugevad küljed on järgmised [4].

- Horisontaalse skaleerimise võimalikkus – Skaleerimine on süsteemile ressursside lisamine, et suurendada süsteemi võimsust. Süsteeme saab skaleerida vertikaalselt ja horisontaalselt. Vertikaalne skaleerimine tähendab füüsiliste ressursside lisamist olemasolevasse masinasse. Horisontaalne skaleerimine tähendab, et süsteemi lisatakse uusi sõlmi (vt **Joonis 2**). Kui töökoormus suureneb ning tekib vajadus süsteemile ressursse lisada, siis hajussüsteeme on võimalik skaleerida horisontaalselt ning peaaegu puuduvad piirangud skaleerimisele ehk kasutatavate masinate arv pole piiratud. Vertikaalset skaleerimist on lihtne teha, kuid mingist hetkest alates hakkavad selle kulud kiiresti (eksponentsiaalselt) kasvama ja üritavad horisontaalse skaleerimise kulud, mis kasvavad lineaarselt [6] .



Joonis 2 Horisontaalne skaleerimine.

- Tõrkekindlus – Kuna süsteem koosneb mitmetest võrgu kaudu ühendatud sõlmedest, siis ühe sõlme rike ei põhjusta terve süsteemi riknemist, kuna hajussüsteemi ülejäänud sõlmed võivad asendada riknenud sõlme ning võtta üle nende ülesandeid. See muudab süsteemi kasutajale kättesaadavamaks e parandab süsteemi käideldavust.

- Paralleelsus – Hajutatud süsteeme saab kavandada paralleelsust toetama, mis tähendab, et keeruka ülesande täitmine on jagatud osadeks ja neid osi täidavad erinevad sõlmed samaaegselt.

- Tõhusus – Hajussüsteemid on tõhusad, kuna selle erinevad osad saavad omavahel töökoormust jagada.

- Paindlikkus – Hajussüsteemi on lihtsam laiendada – selleks saab süsteemi lisada uusi sõlmi.

Hajussüsteemidega kaasnevad ka ohud, mis on järgmised [7].

- Rikke käsitlemine – Kui ühed sõlmed riknevad ja teised jätkavad tööd, siis vigade tuvastamine ja nendest taastumine võib olla hajussüsteemide puhul keerulisem kui tsentraliseeritud süsteemi puhul.

- Võrguühenduse probleemid – Võrguühendus ei ole alati usaldusväärne ja stabiilne, mille tulemuseks on sõnumite kaotsiminek.

- Tarkvaradisaini keerukus – Rakendamise, hoolduse ja tõrkeotsinguga võivad kaasneda raskused, mis muudavad süsteemi projekteerimise keeruliseks.

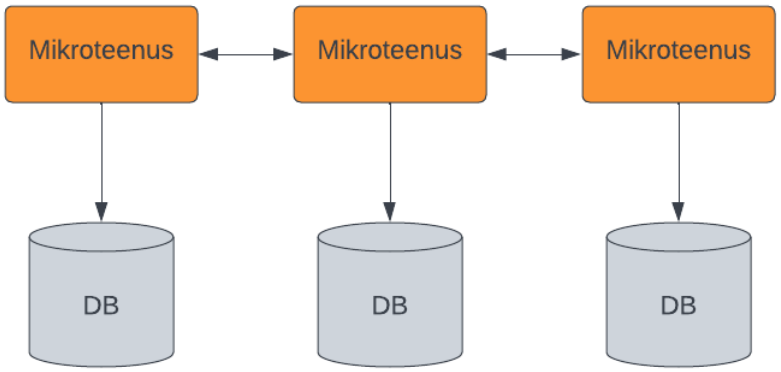
3.2 Monoliitarhitektuur

Mikroteenuste arhitektuuri mõistmiseks peab kõigepealt tegema selgeks, mis on monoliitarhitektuur, sest mikroteenused vastanduvad sellele. Monoliitarhitektuuris on süsteem ehitatud ühtse tervikuna. Monoliitarhitektuuri puhul on rakendus tavaliselt päris suur ja keerukas, ühendab endas kõik tarkvaralt soovitud funktsioonid ning paikneb ühes serveris. Monoliitsüsteemis esineb palju seoseid ja sõltuvusi erinevate osade vahel ning ühe koodi osa muutmine võib mõjutada süsteemi teisi osi. Samuti nõuab iga muudatus terve monoliidi ümberehitamist ja ülespanemist e paigutamist (*deploy*). Sageli kasvavad sellised süsteemid aja jooksul väga suurteks ja keerukateks ning mida suurem on süsteem, seda raskem on teha selles muudatusi, säilitada head struktuuri ning süsteemi edasi arendada. Monoliitsüsteemi skaleerimine võtab rohkem ressursse kui mikroteenuste arhitektuuril põhineva süsteemi skaleerimine, sest on võimalik skaleerida ainult tervet süsteemi ning ei saa skaleerida ainult vajalikke süsteemi osi. Skaleerimist, mille käigus suurendatakse serveri ressursse, näiteks lisatakse rohkem mälu või arvutusvõimsust, nimetatakse vertikaalseks skaleerimiseks. Vertikaalne skaleerimine ei ole kõige efektiivsem lahendus, sest sellel on piiranguid, kuna ühe arvuti maksimaalne võimsus on piiratud. [8] Monoliite on tavaks skaleerida vertikaalselt, kuid hästi disainitud (selgelt piiritletud süsteemi osad) monoliiti saab skaleerida ka horisontaalselt [9]. Horisontaalne skaleerimine suurendab monoliidi maksumust ja keerukust [9].

Monoliitne süsteem ei ole normaliseeritud süsteem [10], sest selle elementide vahelisest suurest sõltuvuste hulgast tulenevalt on selles süsteemis kombinatoorsed efektid. Seetõttu ei ole süsteemi arendus stabiilne kuna süsteemi muutmise keerukus ei sõltu mitte ainult muutuse enese keerukusest, vaid ka süsteemi suurusest. Mida suuremaks muutub süsteem, seda rohkem on normaliseerimata süsteemis sõltuvusi ja seda rohkem tuleb süsteemi mingi osa muutmiseks teha kaskaadselt muudatusi süsteemi teistes osades. Süsteemi muutmise vajadus on aga paratamatu, kuna kõik süsteemid evolutsioneeruvad.

3.3 Mikroteenused

Mikroteenuste arhitektuur on tarkvaraarendamise lähenemine, kus üks rakendus on jaotatud mitmeteks väiksemateks lõdvalt seostatud autonoomselt arendatavateks teenusteks (vt **Joonis 3**). Need teenused on tavaliselt lihtsalt muudetavad ning täidavad suhteliselt väikeseid funktsioone ning kõik teenuse funktsioonid käivad „ühe asja kohta“. Iga teenuse realiseerimise jaoks on võimalik kasutada erinevaid tehnoloogiaid, raamistikke, programmeerimiskeeli ja andmebaasisüsteeme. Kuna mikroteenused on iseseisvad või omavahel nõrgalt seotud, siis see võimaldab muuta ühe teenuse lähtekoodi ilma, et see mõjutaks oluliselt süsteemi tervikuna. See võimaldab erinevatel arendajatel ja meeskondandel arendada süsteemi samaaegselt. Süsteemi, mis koosneb erinevatest mikroteenustest, on võimalik horisontaalselt skaleerida. See tähendab, et ühele serverile ressursside lisamise asemel (vertikaalne skaleerimine), lisatakse süsteemi uusi servereid. See on mugav ja efektiivne, sest serverite arv pole piiratud ning rakenduse võimsus pole piiratud ühe arvuti maksimaalse võimsusega. Samuti on võimalik vastavalt vajadusele skaleerida ainult vajalikke mikroteenuseid, mis on ka kulude mõttes efektiivsem kui terve süsteemi korraga skaleerimine. [11] Mikroteenustel põhinev süsteem on suurema normaliseeritusse astmega kui monoliitsüsteem. Igal teenusel on oma kindel ülesanne. See vastab normaliseeritud süsteemide otstarbe lahususe (*separation of concerns*) teoreemile [10], mille kohaselt peab normaliseeritud süsteemi igal tegevuselemendil (*action entity*) olema ainult üks ülesanne e põhjus seda elementi muuta. Mikroteenus on süsteemi makrotaseme tegevuselement. Normaliseeritud süsteemis peavad tegevuselemendid suutma töötada koos teise tegevuselementide erinevate versioonidega. Kuna mikroteenuste vahel on vähe sõltuvusi, siis on seda kergem saavutada.



Joonis 3 Mikroteenuste arhitektuur.

Mikroteenuste kasutamine võib olla kasulik ja see pakub eeliseid, kuid sellega kaasnevad ka kulud. **Tabel 1** on välja toodud mikroteenuste eelised ja puudused [12].

Tabel 1 Mikroteenuste eelised ja puudused.

Eelised	**Puudused**
Mikroteenused on üksteisest vähe sõltuvad komponendid, mis annab võimaluse arendada samaaegselt sama süsteemi erinevaid osi.	Mikroteenused peavad pidevalt omavahel andmeid vahetama. Sellist süsteemi on monoliidist raskem kavandada ja realiseerida.
Kuna mikroteenused on autonoomsed, siis on neid võimalik paigutada (*deploy*) üksteisest sõltumatult.	Pidevat teenuste vahelist suhtlemist on keeruline programmeerida.
Tehnoloogia mitmekesisus. Iga teenus võib olla realiseeritud kasutades erinevaid programmeerimiskeeli, raamistikke ja andmevahetustehnoloogiaid.	Teenuste haldamiseks peab olema kompetentne meeskond. Liiga palju erinevaid tehnilisi vahendeid muudab ka süsteemi haldamise keerukamaks ja kulukamaks.

3.4 Suhtlus tarkvarakomponentide vahel

Tarkvarakomponentide sh mikroteenuste omavaheliseks suhtlemiseks ja üksteisega andmete vahendamiseks on olemas erinevaid lähenemisi [13]. Mida nimekirjas edasi, seda vähem on suhtlevad osapooled üksteisest sõltuvad ja seda eelistatum on selline suhtlus mikroteenuste vahel.

- Failide edastamine – Rakendused jagavad faile, mis sisaldavad teiste rakenduste tarbitavat informatsiooni. Kuna failide formaat võib erineda, siis peavad rakendused ise neid teisendama või tuleb selleks kasutada mingeid tööriistu.

- Jagatud andmebaas – Erinevad rakendused hoiavad andmeid ühes andmebaasis. Jagatud andmebaasi kasutamisega mikroteenuste vaheliseks suhtlemiseks kaasneb oht, et mikroteenused kaotavad oma põhiomadused nagu sõltumatus ja skaleeritavus.

- Kaugprotseduuri kutsumine (RPC – *remote procedure call*) – Iga rakendus pakub kasutamiseks rutiinidest koosneva liidese, mis võimaldab teistel rakendustel selle rakendusega suhelda. RPC idee seisneb selles, et teisele rakendusele päringu saatmine näeb välja nagu samas rakenduses asuva kohaliku rutiini väljakutsumine [13]. 2022. aasta alguse seisuga on olemas palju RPC põhjal ehitatud raamistikke. Üks selline on gRPC, mis kasutab JSON asemel kahendformaadis andmete kodeerimist (Protocol Buffer formaadis). Kahendformaadis esitatud andmed on mahult väiksemad kui tekstilised andmed ning nende saatmine üle võrgu on tänu sellele palju kiirem [14].

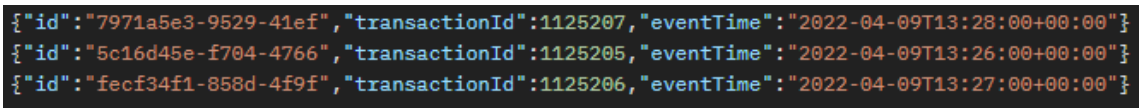
- REST kasutajaliides – Rakendused suhtlevad üksteisega otse ja sünkroonselt HTTP kaudu. Üks rakendus saadab päringu teisele läbi REST liidese ning teine kas teeb midagi saadud informatsiooniga või saadab tagasi mingi vastuse. Selline lähenemine võib põhjustada päringute paljusust [11]. Põhimõtteliselt kutsutakse samuti välja teise osapoole pakutav teenus ja oodatakse selle täitmist (vaata RPC), kuid väljakutse näeb teistmoodi välja.

- Sõnumivahetus – Rakendused ühenduvad ühise sõnumsidesüsteemiga ning vahetavaid andmeid sõnumite abil. Antud lähenemine aitab edastada andmeid kiiresti, usaldusväärselt ja asünkroonselt. Sõnumite saatmine ei nõua, et mõlemad

rakendused töötaksid samaaegselt. Sõnumi saatja ei pea ootama adressaadilt vastust, vaid saab peale saatmist jätkata oma tööd ning adressaat tegeleb saadud sõnumiga siis, kui tal on selleks võimalus. See lahendab erinevaid hajussüsteemide probleeme [13].

3.5 JSONL

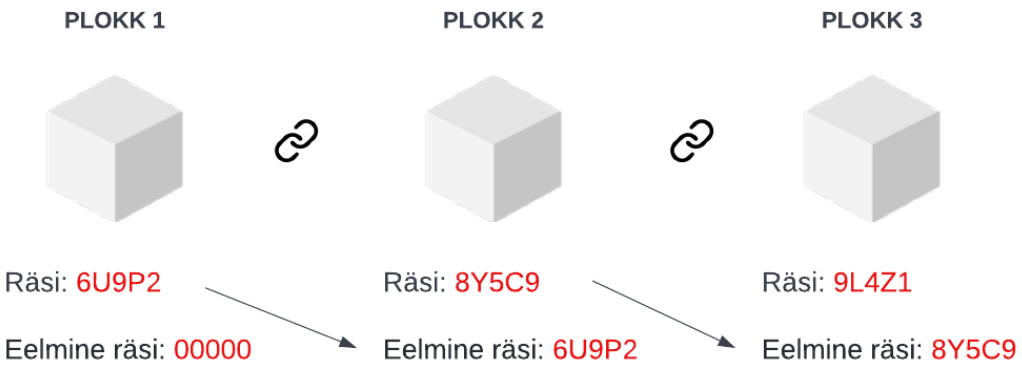
JSONL on JSON Lines teksti formaat, mida nimetatakse ka reavahetusega eraldatud JSON-ks. JSONL vormingus peab iga rida olema valiidne JSON väärtus, mis on kodeeritud UTF-8 abil. [15] **Joonis 4** esitab JSONL näite.



Joonis 4 JSONL näide.

3.6 Plokiaheldamine

Plokk on andmete kogum, näiteks andmebaasi kirje. Plokiahelat võib seletada nagu plokkide ahelat, mis on omavahel seotud krüptograafiliste räside abil. Plokke lisatakse plokiahela lõppu ning iga uus lisatud plokk sisaldab sellele eelneva ploki räsi (vt **Joonis 5**). Igast plokist on võimalik jõuda juurplokini järgides ploki eelmist räsi. Kui vaadata joonist, siis plokis on olemas eelmise ploki räsi, mis viitab eelmisele plokile. Eelmine plokk viitab samal viisil üleelmisele jne. Nii võibki juurplokini jõuda. Plokke omavahel siduv räsi genereeritakse krüptograafilise ühesuunalise räsifunktsiooni abil kasutades näiteks SHA-256 algoritmi. Plokiaheldamine tagab andmete muutumatuse, sest kui plokis olevat informatsioon muudetakse, siis muutub ka ploki räsi. Selleks et kontrollida andmete ehk plokkide muutumatust, peab ploki räsi uuesti arvutama. Kui ühes plokis muutub räsi, siis järgmiste plokkide räsi ümberarvutamisel saadakse vale tulemus, sest eelmise ploki räsi väärtus on muutunud. [16]



Joonis 5 Plokiahela näide.

3.7 X509 sertifikaat

X509 sertifikaate kasutatakse sidepartneri identiteeti kontrollimiseks HTTPS (HTTP over SSL) protokolli kasutamisel. See tähendab, et turvalise ühenduse loomise ajal peab server kontrollima kliendi sertifikaati. Sertifikaati struktuur on määratletud X.509 standardiga. See sisaldab näiteks versiooninumbrit, seerianumbrit, digitaalset allkirja, väljaandja nime, kehtivusaega ja informatsiooni avaliku võtme kohta. [17]

3.8 Digitaalne arhiiv

Arhiiv on inimtegevuse tulemusena tekkinud dokumentide kogum, milles olevaid dokumente säilitatakse nende väärtuslikkuse pärast mingi aja jooksul või ka piiramatu tähtajaga. Arhiiv annab ülevaate minevikus toimunud sündmustest, kuna dokumenteerib üksikisikute või organisatsioonide tegevust. Arhiivides säilitatakse andmeid erinevates formaatides – digitaalselt (elektrooniliselt) või analoogselt. [18]

Digitaalsete andmete säilitamine tähendab tegevuste jada, mis tagab pideva tõrkekindla juurdepääsu digitaalsetele andmetele teatud perioodi jooksul. Need on väärtuslikud andmed, mis on loodud kas indiviidi või organisatsiooni tegevuste tagajärjel ning pole antud ajal süsteemis aktiivses kasutuses. [19]

ISO 15489 on rahvusvaheline standard äriandmete haldamiseks. Vastavalt sellele standardile peab arhiiv järgima järgnevaid põhimõtteid.

1. Andmete terviklikkus – Kõik arhiivis olevad andmed peavad olema kaitstud muutmise eest või muutmise eest ilma dokumenteeritud loata ning süsteem peab olema võimeline rikutud andmeid leidma.
2. Andmete püsivus – Arhiiv peab tagama andmete säilimise teatud aja jooksul.
3. Andmete juurdepääsetavus – Andmeid peab olema võimalik üle vaadata.
4. Andmeturve – Andmed peavad olema kaitstud.
5. Usaldusväärsus – Andmed peavad olema usaldusväärsed.
6. Andmete arusaadavus – Andmed peavad olema mõistetavad.
7. Metaandmed – Metaandmete eesmärk on kirjeldada andmete konteksti, sisu ja struktuuri ning andmete haldamise ajalugu [19] Digitaalses arhiivis peavad olema lisaks andmetele ka nende metaandmed e andmed arhiivis olevate andmete kohta.

3.8.1 Olemasolevad lahendused

Olemasolevate lahenduste leidmiseks tehti Google Scholar [20] otsingusüsteemis järgmiste otsingufraasidega päringuid: *digital archiving, archive design, digital preservation.*

Üks näide olemasolevast arhiivilahendusest on meditsiiniliste piltide salvestamiseks loodud arhiiv, mis on realiseeritud hajutatud failisüsteemi arhitektuuri põhjal [21]. Antud süsteem on ülesehitatud nii, et see oleks võimeline toime tulema suurte andmemahtudega. Süsteem koosneb klientrakendustest, mis saadavad andmeid üle võrgu andmeserveritele. Klientrakenduste arv ei ole piiratud. Kasutajate autentimine toimub sümmeetrilise krüptograafia meetodite abil. Süsteem koosneb ka mitmest andmesõlmest ning kõik sõlmed on ühesuguse prioriteetsusega, mis tagab süsteemi töökindluse ja kättesaadavuse. Andmesõlmed säilitavad ja haldavad klientide poolt saadetud andmeid. Peale andmesõlme on süsteemis realiseeritud ka koopia (*replica*) serverid, mis teevad koopiaid andmesõlmede poolt salvestatud andmetest. Andmeid hoitakse failis, mis võtab enda alla 80% füüsilise seadme vabast püsimälust. [22]

Teine leitud arhiivilahendus on realiseeritud MARS raamistiku jaoks. MARS on interaktiivne platvorm ekspertidele nagu ökoloogid, bioloogid, keemikud, mis pakub võimalust simuleerida erinevaid keerulisi stsenaariume (nt bakterite paljunemine). Süsteem on realiseeritud mikroteenuste arhitektuuri põhjal ning pakub detsentraliseeritud juurdepääsu andmetele, mis tagab parema töökindluse ja skaleeritavuse. Süsteem võtab

vastu erinevaid faile, kus võivad olla näiteks geoandmed, mudelid või aegread. Andmeid saadetakse, uuendatakse ja küsitakse HTTP kaudu. Andmete hoidmise jaoks kasutatakse NAS Synology [23], mis annab kasutajatele võimaluse küsida andmeid üle võrgu, sest andmeid hoitakse pilves. [24]

Kolmandaks lahenduseks on MIDAS süsteem. See on veebipõhine digitaalsete andmete arhiveerimissüsteem, mis on mõeldud suurte teaduslike andmekogude töötlemiseks. MIDAS süsteem on võetud kasutusele mitmetes uurimislaborites. Süsteem suudab salvestada erinevas formaadis digitaalseid andmeid nagu näiteks Wordi dokumente, PDF faile ja 3D mudeleid. MIDAS süsteem on ehitatud avatud lähtekoodiga platvormi DSpace põhjal. Teaduslike andmekogumite metaandmete hoidmiseks kasutab süsteem XCEDE standardit. MIDAS toetab hajutatud andmetöötlust, kasutades selleks avatud lähtekoodiga tööriista Condor. Süsteem pakub võimalust määrata erinevatele kasutajatele ja andmekogudele erineva kasutajapoliitika, millesse kuuluvad näiteks õigused andmete üleslaadimiseks ja allalaadimiseks. Süsteem on põhiliselt kirjutatud PHP programmeerimiskeeles ning mõned moodulid on realiseeritud C++ keeles. Andmete hoidmiseks kasutab süsteem PostgreSQL andmebaasi. [25]

Neljas lahendus on prototüüpsüsteem, mille eesmärgiks on meditsiiniliste digitaalsete piltide arhiveerimine. Süsteem on realiseeritud kasutades populaarset pilvandmetöötluse platvormi Microsoft Windows Azure. Süsteem kasutab DICOM serverit, mis teeb andmete salvestamise ja küsimise päringuid. DICOM server on realiseeritud C# programmeerimiskeeles ning põhineb avatud lähtekoodiga DICOM# [26] projektil. Süsteem toetab skaleerimist ning iga süsteemi komponendi eksemplaride arvu võib vajadusel suurendada või vähendada. Süsteemis hoitakse nii arhiveeritud andmeid kui ka metaandmeid SQL Azure andmebaasis.

4 Süsteemianalüüs

Selles peatükis kirjeldatakse nõuded loodavale tarkvarale.

Loodav süsteem peab klientidelt vastu võtma sündmuste pakke, mis koosnevad ühest või mitmest sündmusest. Pakk koosneb ühest või mitmest JSON objektist (sündmusest). Maksimaalne sündmuste arv ühes pakkis on 1000. Iga sündmuste pakk on JSONL formaadis dokument. Iga sündmus on esitatud JSON formaadis ja on üks rida JSONL dokumendis. Kõik ühes pakis olevad sündmused on ühte tüüpi ja lähevad lõpuks andmebaasis ühte ja samasse tabelisse. Kõik sündmuste tüübid peavad olema süsteemis defineeritud. See tähendab, et iga riigis olulise sündmuse tüübi kohta tuleb rakenduses luua täpselt üks klass ja andmebaasis üks tabel iga selle riigi teenusepakkuja kohta, kes seda tüüpi sündmuste andmeid saadab. Enne sündmuste pakkide vastuvõtmist peab süsteem kontrollima, kas klient on süsteemis registreeritud ning kas tal on sündmuste pakkide saatmiseks piisavalt õiguseid. Samuti peab süsteem kontrollima, kas sündmuste pakkide jaoks on andmebaasis olemas vastav tabel. Vastuvõetud sündmused tuleb valideerida, töödelda, aheldada ning andmebaasi salvestada. Sündmuste valideerimine seisneb sündmuste väljade kontrollimises. Kontrollitakse näiteks kohustuslike väljade väärtuste olemasolu, ainult tühikutest koosnevate väljade väärtuste puudumist või väljade väärtuste vastamist mustritele. Süsteem peab kontrollima andmebaasis hoitavate sündmuste terviklikkust. Andmete terviklikkus tähendab, et andmed on sellised nagu need salvestati ja neid ei ole vahepeal muudetud.

4.1 Süsteemi eesmärk

Järgnevalt esitatakse loodava süsteemi eesmärgid.

- Tagada süsteemi talletamiseks saadetud andmete säilimine määratud perioodi jooksul.
- Tagada võimalus kontrollida salvestatud andmete terviklikkust ja vaadata kontrollide tulemusi.
- Tagada võimalus hallata süsteemi kasutajaid ja nende õiguseid.

- Tagada võimalus hallata süsteemi kasutatavaid andmebaase ja skeeme.

4.2 Tükeldus allsüsteemideks

Järgnevalt esitatakse infosüsteemi jaotus allsüsteemideks. Süsteemil on kaks pädevusala e rolli, mille esindajad süsteemi kasutavad – arendaja ja andmebaasi administraator.

Tabel 2 esitab funktsionaalsed allsüsteemid ja nende teenindatavad andmekesksed allsüsteemid e registrid. Iga funktsionaalne allsüsteem vastab mingile süsteemi põhifunktsioonile. Teenindamine tähendab, et see allsüsteem nii loeb kui ka muudab registris olevaid andmeid.

Tabel 2 Funktsionaalsed allsüsteemid.

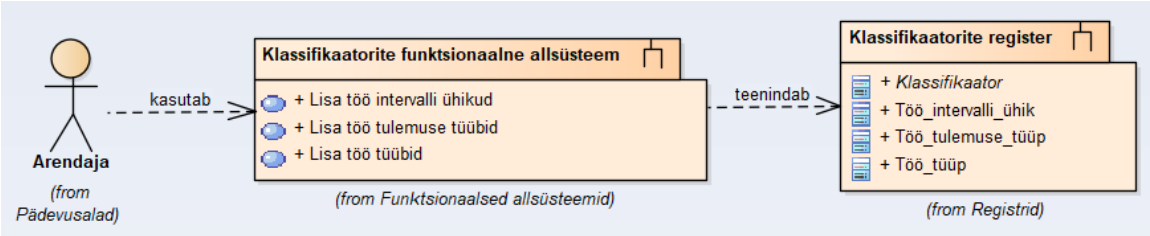
Funktsionaalne allsüsteem	Register, mida see funktsionaalne allsüsteem teenindab
Klassifikaatorite funktsionaalne allsüsteem	Klassifikaatorite register
Andmebaaside funktsionaalne allsüsteem	Andmebaaside register
Klientide funktsionaalne allsüsteem	Klientide register
Sündmuste pakkide funktsionaalne allsüsteem	Sündmuste pakkide register
Töö spetsifikatsioonide funktsionaalne allsüsteem	Töö spetsifikatsioonide register
Tööde funktsionaalne allsüsteem	Tööde register

4.3 Funktsionaalsed nõuded

Selles jaotises kirjeldatakse süsteemi funktsionaalsed nõuded funktsionaalsete allsüsteemide kaupa. Süsteemi funktsionaalsete nõuete esitamiseks kasutatakse käesolevas töös kasutusjuhtude mudelit.

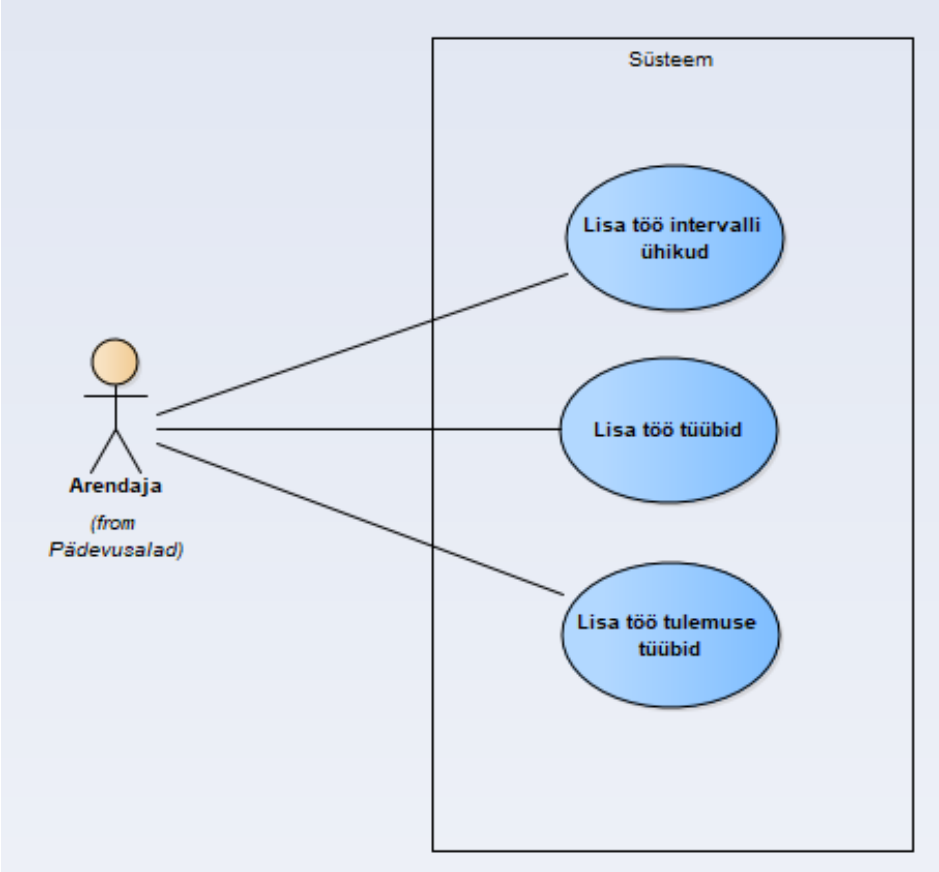
4.3.1 Klassifikaatorite funktsionaalne allsüsteem

Joonis 6 esitab paketidiagrammi kasutades klassifikaatorite funktsionaalse allsüsteemi seosed teiste allsüsteemidega (pädevusalade ja registritega).



Joonis 6 Klassifikaatorite funktsionaalse allsüsteemi paketidiagramm.

Joonis 7 esitab klassifikaatorite funktsionaalse allsüsteemi kasutusjuhtude diagrammi.



Joonis 7 Klassifikaatorite funktsionaalse allsüsteemi kasutusjuhtude diagramm.

Rakenduse haldamise lihtsustamiseks haldab klassifikaatoreid arendaja ise. See tähendab, et süsteem ise salvestab käivitamisel andmebaasi kõik vajalikud klassifikaatorite väärtused, mis on arendaja poolt SQL lauseid sisaldavate skriptidena ära kirjeldatud. Hetkel kasutatakse süsteemis üsna vähe klassifikaatoreid – ajaühikud ning tööde tüübid ja töö tulemused. Need on otseselt seotud süsteemi funktsionaalsusega ja on defineeritud

süsteemi sees. Juhul kui tekib täiendavate klassifikaatori tüüpide kasutusele võtmise vajadus, siis on võimalik viia klassifikaatorite haldamine andmebaasi administraatori ülesandeks. Selleks peab süsteemi lisama kasutajaliidese (saab näiteks realiseerida REST abil) ja realiseerima vastava funktsionaalsuse.

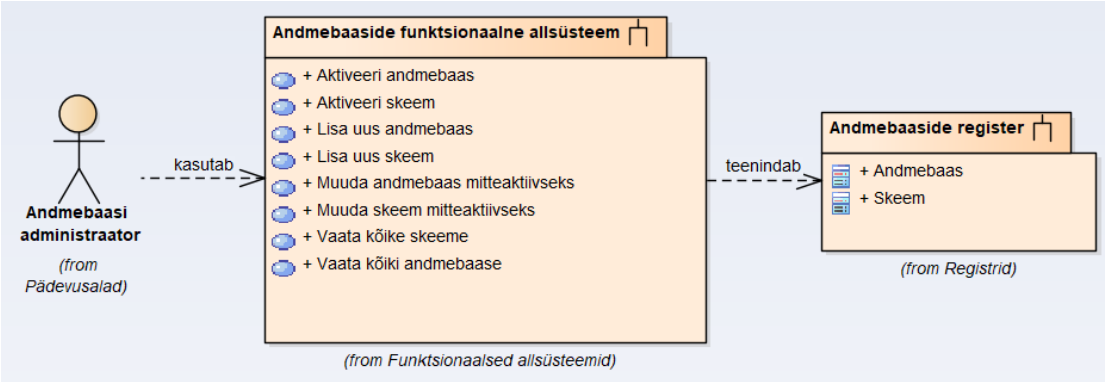
Kasutusjuht: Lisa töö intervalli ühikud
Tegutsejad: Arendaja
Kirjeldus: Süsteemi käivitamisel kontrollib süsteem, kas andmebaasis on töö intervalli ühikud registreeritud ning kui ei ole, siis lisab skripti käivitamise tulemusel töö intervalli ühikud.

Kasutusjuht: Lisa töö tüüp
Tegutsejad: Arendaja
Kirjeldus: Süsteemi käivitamisel kontrollib süsteem, kas andmebaasis on töö tüübid registreeritud ning kui ei ole, siis lisab skripti käivitamise tulemusel töö tüübid. Töö tüüp aitab näidata, milline töö oli teostatud.

Kasutusjuht: Lisa töö tulemuse tüüp
Tegutsejad: Arendaja
Kirjeldus: Süsteemi käivitamisel kontrollib süsteem, kas andmebaasis on töö tulemuse tüübid registreeritud ning kui ei ole, siis lisab skripti käivitamise tulemusel töö tulemuse tüübid. Töö tulemuse tüüp näitab, kas tehtud töö õnnestus või mitte.

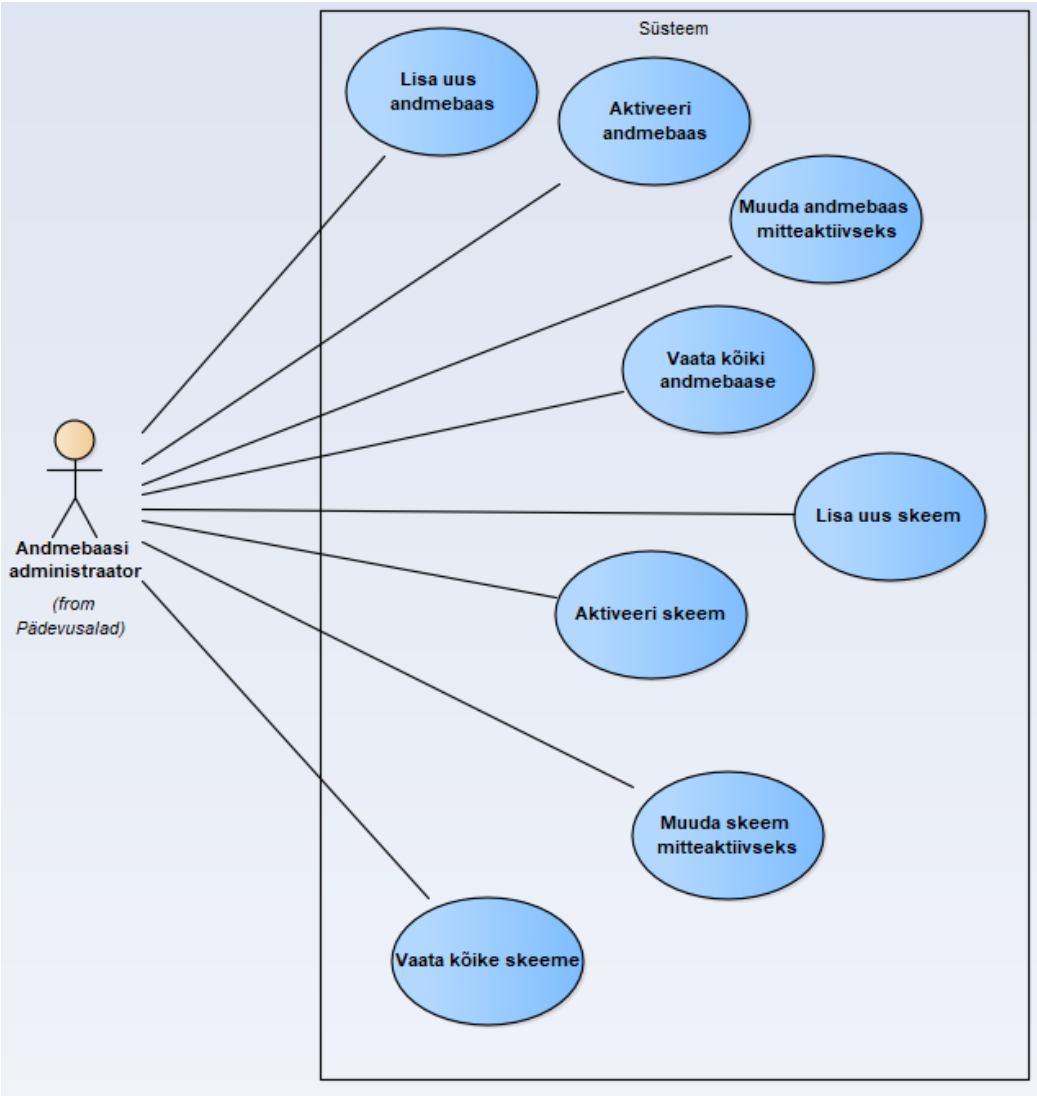
4.3.2 Andmebaaside funktsionaalne allsüsteem

Joonis 8 esitab paketidiagrammi kasutades andmebaaside funktsionaalse allsüsteemi seosed teiste allsüsteemidega.



Joonis 8 Andmebaaside funktsionaalse allsüsteemi paketidiagramm.

Joonis 9 esitab andmebaaside funktsionaalse allsüsteemi kasutusjuhtude diagrammi.



Joonis 9 Andmebaaside funktsionaalse allsüsteemi kasutusjuhtude diagramm.

Kasutusjuht: Lisa uus andmebaas

Tegutsejad: Andmebaasi administraator

Kirjeldus: Andmebaasi administraator lisab süsteemi uue andmebaasi.

Kasutusjuht: Lisa uus skeem

Tegutsejad: Andmebaasi administraator

Kirjeldus: Andmebaasi administraator lisab uue skeemi, mis peab olema seotud ühe süsteemis olemasoleva andmebaasiga. Süsteem käivitab skripti, millega luuakse skeemis tabelid. Iga tabel vastab eraldi sündmuse tüübile, millele vastavate sündmuste andmeid soovitakse arhiveerida.

Kasutusjuht: Aktiveeri andmebaas

Tegutsejad: Andmebaasi administraator

Kirjeldus: Andmebaasi administraator aktiveerib andmebaasi kui see on mitteaktiivne. Aktiivses seisundis andmebaasi on võimalik salvestada sündmuseid (juhul kui ka seotud skeem, milles olevatesse tabelitesse arhiveeritavad andmed pannakse, on aktiivne).

Kasutusjuht: Muuda andmebaas mitteaktiivseks

Tegutsejad: Andmebaasi administraator

Kirjeldus: Andmebaasi administraator muudab andmebaasi mitteaktiivseks kui see on aktiivne. Mitteaktiivsesse andmebaasi ei ole võimalik sündmuseid salvestada.

Kasutusjuht: Aktiveeri skeem

Tegutsejad: Andmebaasi administraator

Kirjeldus: Andmebaasi administraator aktiveerib skeemi kui see on mitteaktiivne. Aktiveeritud skeemi on võimalik salvestada sündmuseid (juhul kui ka seotud andmebaas on aktiivne).

Kasutusjuht: Muuda skeem mitteaktiivseks

Tegutsejad: Andmebaasi administraator

Kirjeldus: Andmebaasi administraator muudab skeemi mitteaktiivseks kui see on aktiivne. Mitteaktiivsesse skeemi ei ole võimalik sündmuseid salvestada.

Kasutusjuht: Vaata kõiki andmebaase

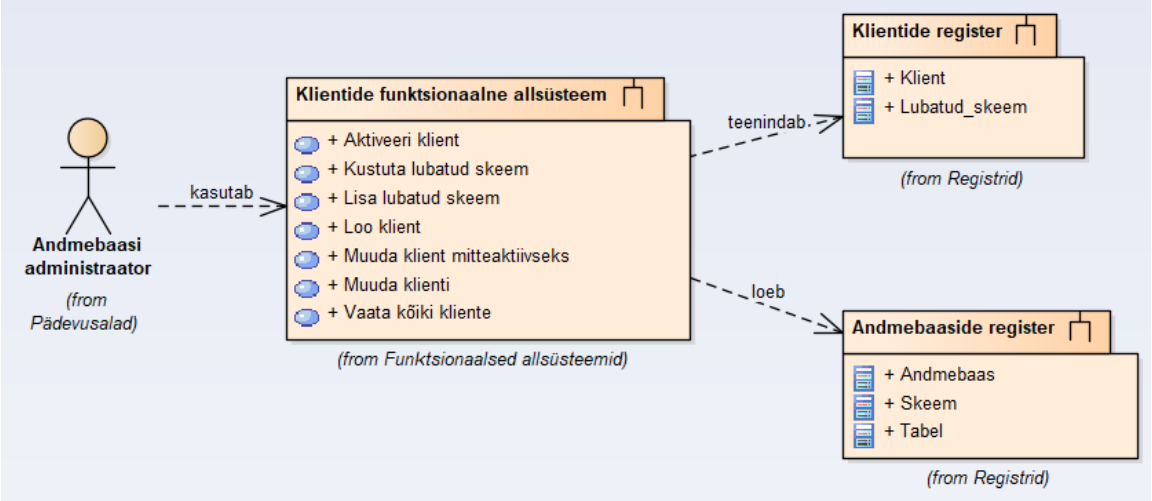
Tegutsejad: Andmebaasi administraator

Kirjeldus: Andmebaasi administraator vaatab kõiki andmebaase. Andmebaase kuvatakse koos andmebaasisüsteemi poolt antud unikaalse identifikaatoriga ning staatusega (aktiivne/mitteaktiivne).

Kasutusjuht: Vaata kõiki skeeme
Tegutsejad: Andmebaasi administraator
Kirjeldus: Andmebaasi administraator vaatab kõiki skeeme. Skeemid kuvatakse koos andmebaasisüsteemi poolt antud unikaalse identifikaatoriga, seotud andmebaasi nimega ning staatusega (aktiivne/mitteaktiivne).

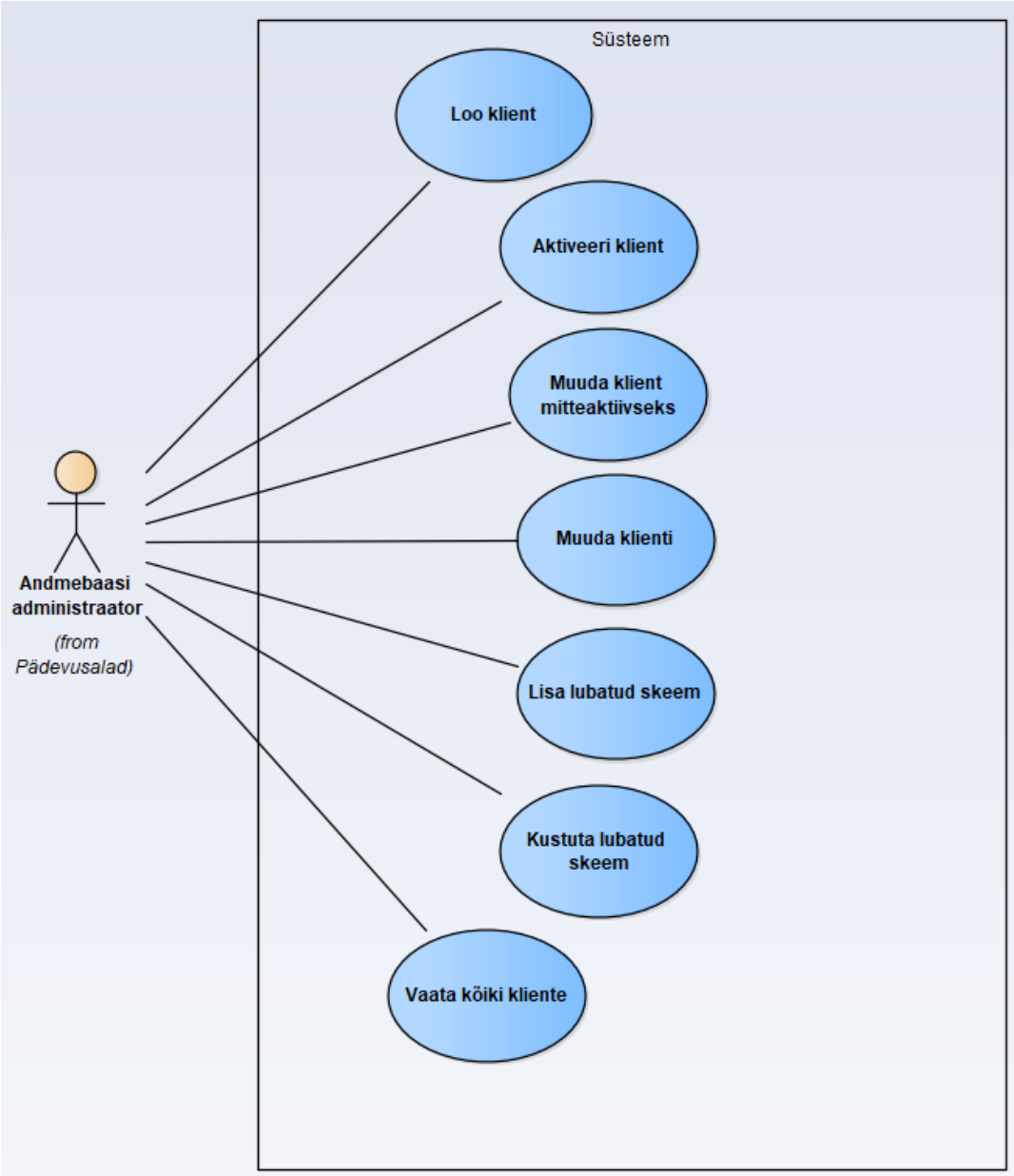
4.3.3 Klientide funktsionaalne allsüsteem

Joonis 10 esitab paketidiagrammi kasutades klientide funktsionaalse allsüsteemi seosed teiste allsüsteemidega.



Joonis 10 Klientide funktsionaalse allsüsteemi paketidiagramm.

Joonis 11 esitab klientide funktsionaalse allsüsteemi kasutusjuhtude diagrammi.



Joonis 11 Klientide funktsionaalse allsüsteemi kasutusjuhtude diagramm.

Kasutusjuht: Loo klient

Tegutsejad: Andmebaasi administraator

Kirjeldus: Andmebaasi administraator lisab süsteemi uue kliendi. Süsteemi saavad saata sündmuste pakke ainult registreeritud ja aktiivsed kliendid. Kliendiks võib olla kas füüsiline isik või teine rakendus.

Kasutusjuht: Aktiveeri klient

Tegutsejad: Andmebaasi administraator

Kirjeldus: Andmebaasi administraator muudab kliendi aktiivseks kui see on mitteaktiivne. Aktiivne klient saab süsteemi sündmuseid saata talle lubatud skeemidesse.

Kasutusjuht: Muuda klient mitteaktiivseks
Tegutsejad: Andmebaasi administraator
Kirjeldus: Andmebaasi administraator muudab kliendi mitteaktiivseks kui see on aktiivne. Mitteaktiivne klient ei saa süsteemi andmeid saata.

Kasutusjuht: Muuda klienti
Tegutsejad: Andmebaasi administraator
Kirjeldus: Andmebaasi administraator muudab kliendi andmeid.

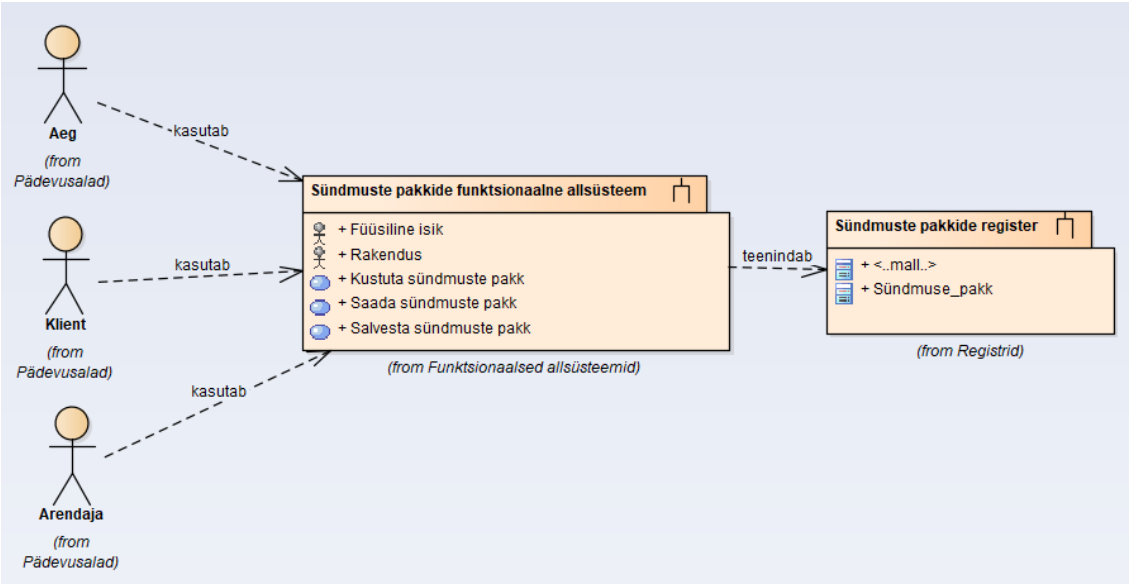
Kasutusjuht: Lisa lubatud skeem
Tegutsejad: Andmebaasi administraator
Kirjeldus: Andmebaasi administraator määrab, et mingit skeemi saab kasutada kliendi andmete arhiveerimiseks. See tähendab, et kliendile lubatakse salvestada andmeid antud skeemi. Klient ei saa saata andmeid talle mittelubatud skeemi.

Kasutusjuht: Kustuta lubatud skeem
Tegutsejad: Andmebaasi administraator
Kirjeldus: Andmebaasi administraator määrab, et mingit skeemi ei saa enam kasutada kliendi andmete arhiveerimiseks. See tähendab, et kliendil ei ole enam rohkem lubatud andmeid sellesse skeemi salvestada. See ei mõjuta seda, kas klient saab (teises süsteemis) neid andmeid vaadata või mitte. Kliendi ja skeemi seose kustutamisel kliendi saadetud andmeid sellest skeemist ei kustutata.

Kasutusjuht: Vaata kõiki kliente
Tegutsejad: Andmebaasi administraator
Kirjeldus: Andmebaasi administraator saab vaadata kõiki kliente. Kliente kuvatakse koos unikaalse identifikaatoriga ja neile lubatud skeemidega koos andmebaasi nimega.

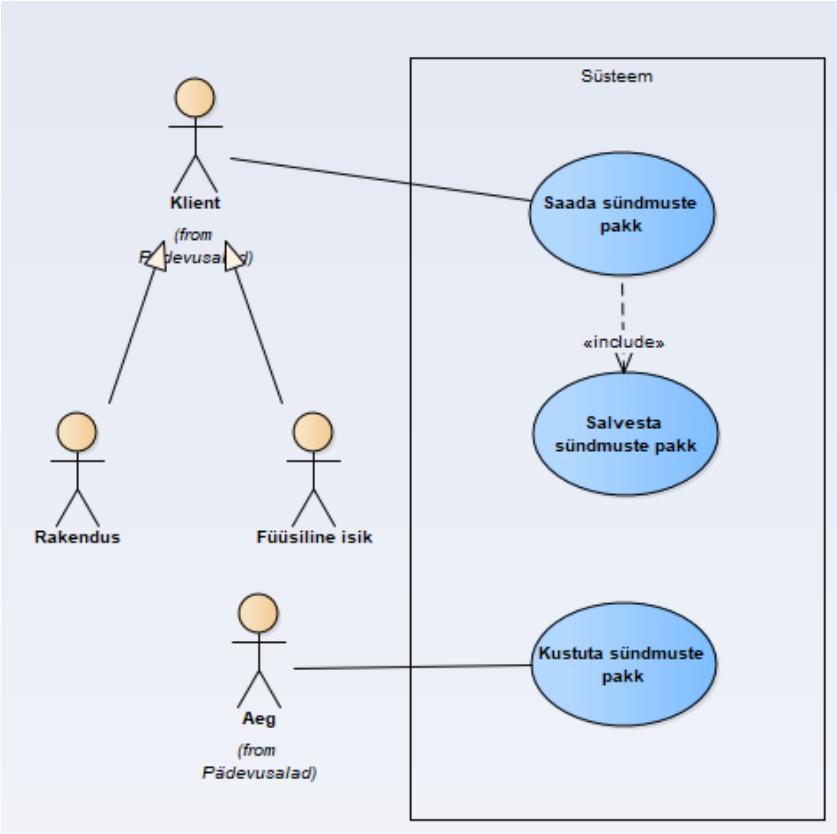
4.3.4 Sündmuste pakkide funktsionaalne allsüsteem

Joonis 12 esitab paketidiagrammi kasutades sündmuste pakkide funktsionaalse allsüsteemi seosed teiste allsüsteemidega.



Joonis 12 Sündmuste pakkide funktsionaalse allsüsteemi paketidiagramm.

Joonis 13 esitab sündmuste pakkide funktsionaalse allsüsteemi kasutusjuhtude diagrammi.



Joonis 13 Sündmuste pakkide funktsionaalse allsüsteemi kasutusjuhtude diagramm.

Kasutusjuht: Saada sündmuste pakk

Tegutsejad: Klient

Kirjeldus: Klient saadab JSONL formaadis sündmuste paki. Pakis on ühe või rohkema sündmuse andmed. Päringu URL peab viitama andmebaasile, skeemile ja tabelile kuhu andmeid saadetakse. Tabel määrab ära süsteemis defineeritud sündmuse tüübi.

Kasutusjuht: Salvesta sündmuste pakk

Tegutsejad: Klient

Kirjeldus: Kliendi poolt saadetud sündmuste pakid salvestatakse andmebaasi sündmuse tüübile vastavasse tabelisse.

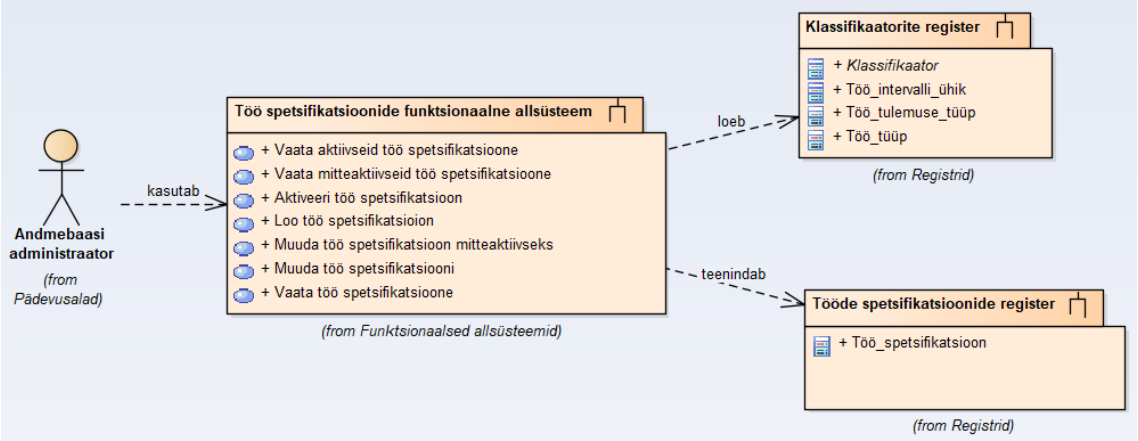
Kasutusjuht: Kustuta sündmuste pakk

Tegutsejad: Aeg

Kirjeldus: Iga sündmuse tüübi jaoks on määratud säilitamisaeg. Kui sündmuste paki säilitamisaeg on möödunud, siis antud sündmuste pakk kustutatakse. See tähendab andmete kustutamist sündmuse tüübile vastavast tabelist.

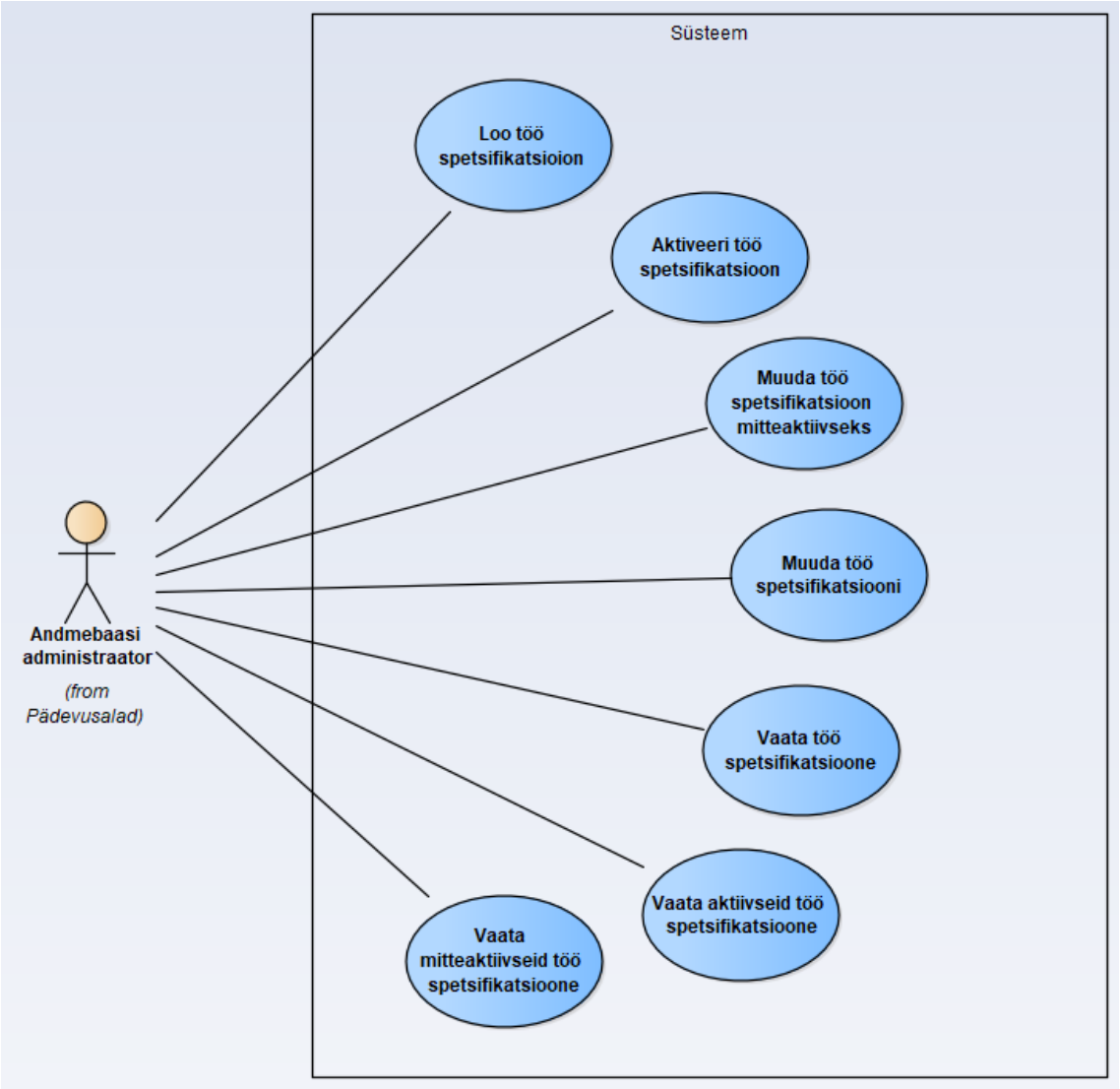
4.3.5 Töö spetsifikatsioonide funktsionaalne allsüsteem

Joonis 14 esitab paketidiagrammi kasutades töö spetsifikatsioonide funktsionaalse allsüsteemi seosed teiste allsüsteemidega.



Joonis 14 Töö spetsifikatsioonide funktsionaalse allsüsteemi paketidiagramm

Joonis 15 esitab töö spetsifikatsioonide funktsionaalse allsüsteemi kasutusjuhtude diagrammi.



Joonis 15 Töö spetsifikatsioonide funktsionaalse allsüsteemi kasutusjuhtude diagramm.

Kasutusjuht: Loo töö spetsifikatsioon

Tegutsejad: Andmebaasi administraator

Kirjeldus: Andmebaasi administraator lisab uue töö spetsifikatsiooni. Töö spetsifikatsioon peab määrama sündmuse tüübi (tabeli), milles olevate andmete terviklikkust peab süsteem kontrollima.

Kasutusjuht: Aktiveeri töö spetsifikatsioon

Tegutsejad: Andmebaasi administraator

Kirjeldus: Andmebaasi administraator muudab mitteaktiivse töö spetsifikatsiooni aktiivseks. Süsteem alustab tööd vastavalt töö spetsifikatsioonile. Töö spetsifikatsiooni

aktiivsus või mitteaktiivsus ei sõltu tabelit sisaldava skeemi aktiivsusest või mitteaktiivsusest. See tähendab, et kui skeem on mitteaktiivne, siis töö spetsifikatsioon võib ikka jääda aktiivseks ja selles skeemis olevates tabelites olevate sündmuste andmete muutumatust võib süsteem selle töö spetsifikatsiooni alusel kontrollida.

Kasutusjuht: Muuda töö spetsifikatsioon mitteaktiivseks
Tegutsejad: Andmebaasi administraator
Kirjeldus: Andmebaasi administraator muudab aktiivse töö spetsifikatsiooni mitteaktiivseks. Kui töö spetsifikatsioon muudakse mitteaktiivseks, siis lõpetatakse tööde teostamine selle spetsifikatsiooni alusel. See tähendab, et hetkel selle järgi tehtav töö lõpetatakse ja järgmine ei lähe käima.

Kasutusjuht: Muuda töö spetsifikatsiooni
Tegutsejad: Andmebaasi administraator
Kirjeldus: Andmebaasi administraator muudab töö spetsifikatsiooni andmeid. Süsteem lõpetab töö spetsifikatsiooni alusel toimuvad tööd ning alustab uut tööd vastavalt muudetud töö spetsifikatsioonile.

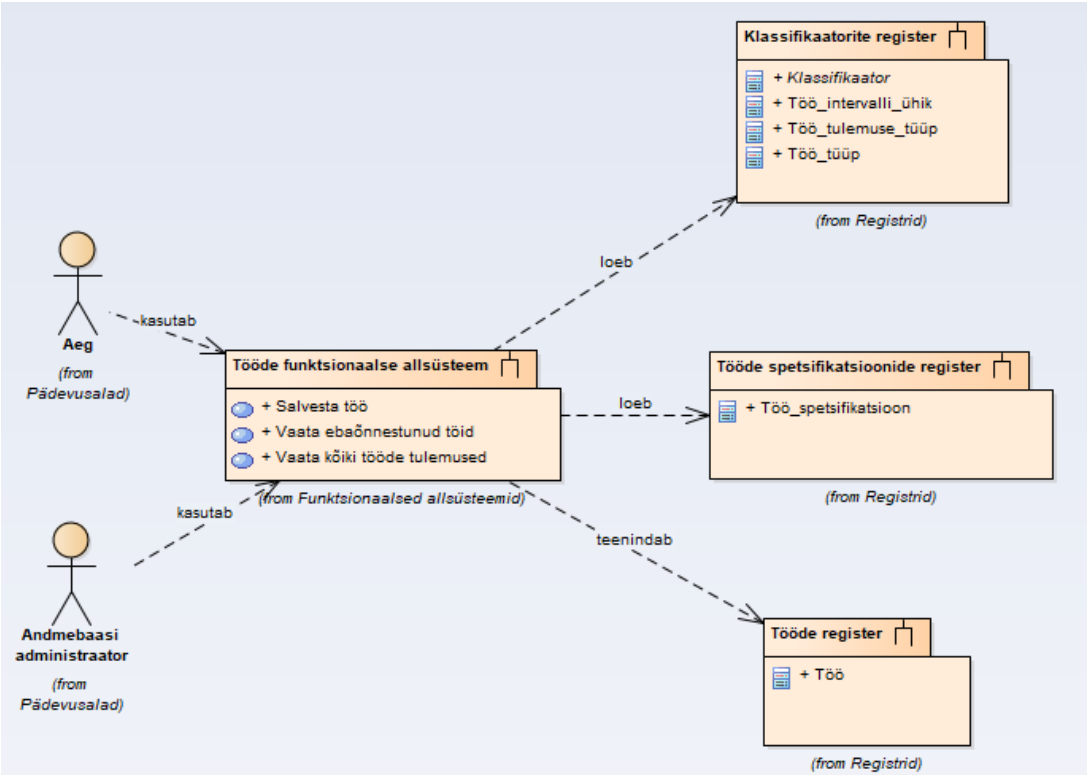
Kasutusjuht: Vaata töö spetsifikatsioone
Tegutsejad: Andmebaasi administraator
Kirjeldus: Andmebaasi administraator saab vaadata töö spetsifikatsioonide nimekirja. Andmebaasi administraator saab iga töö spetsifikatsiooni korral vaadata selle kõiki detailseid andmeid.

Kasutusjuht: Vaata aktiivseid töö spetsifikatsioone
Tegutsejad: Andmebaasi administraator
Kirjeldus: Andmebaasi administraator saab vaadata aktiivsete töö spetsifikatsioonide nimekirja.

Kasutusjuht: Vaata mitteaktiivseid töö spetsifikatsioone
Tegutsejad: Andmebaasi administraator
Kirjeldus: Andmebaasi administraator saab vaadata mitteaktiivsete töö spetsifikatsioonide nimekirja.

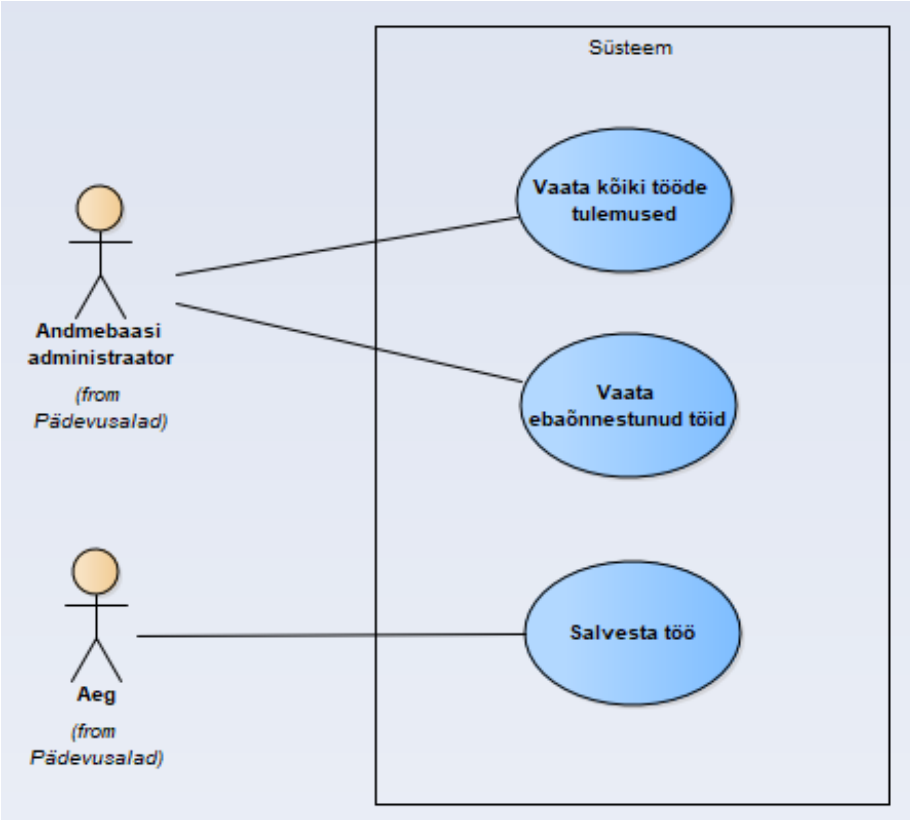
4.3.6 Tööde funktsionaalne allsüsteem

Joonis 16 esitab paketidiagrammi kasutades tööde funktsionaalse allsüsteemi seosed teiste allsüsteemidega.



Joonis 16 Tööde funktsionaalse allsüsteemi paketidiagramm

Joonis 17 esitab tööde funktsionaalse allsüsteemi kasutusjuhtude diagrammi.



Joonis 17 Tööde funktsionaalse allsüsteemi kasutusjuhtude diagramm

Kasutusjuht: Vaata kõikide tööde tulemusi

Tegutsejad: Andmebaasi administraator

Kirjeldus: Andmebaasi administraator vaatab terviklikkuse kontrollimises tehtud tööde nimekirja. Andmebaasi administraator saab iga töö korral vaadata selle kõiki detailseid andmeid.

Kasutusjuht: Vaata ebaõnnestunud tööde tulemusi

Tegutsejad: Andmebaasi administraator

Kirjeldus: Andmebaasi administraator vaatab ebaõnnestunud tööde nimekirja.

Kasutusjuht: Salvesta töö

Tegutsejad: Aeg

Kirjeldus: Süsteem täidab kliendi poolt salvestatud andmete terviklikkuse kontrolli (teeb tööd) ning kontrolli tulemused (tööd) salvestatakse andmebaasi. Töid tehakse mingi perioodilisusega.

4.4 Mittefunktsionaalsed nõuded

Tabel 3 esitab süsteemi mittefunktsionaalsed nõuded.

Tabel 3 Süsteemi mittefunktsionaalsed nõuded.

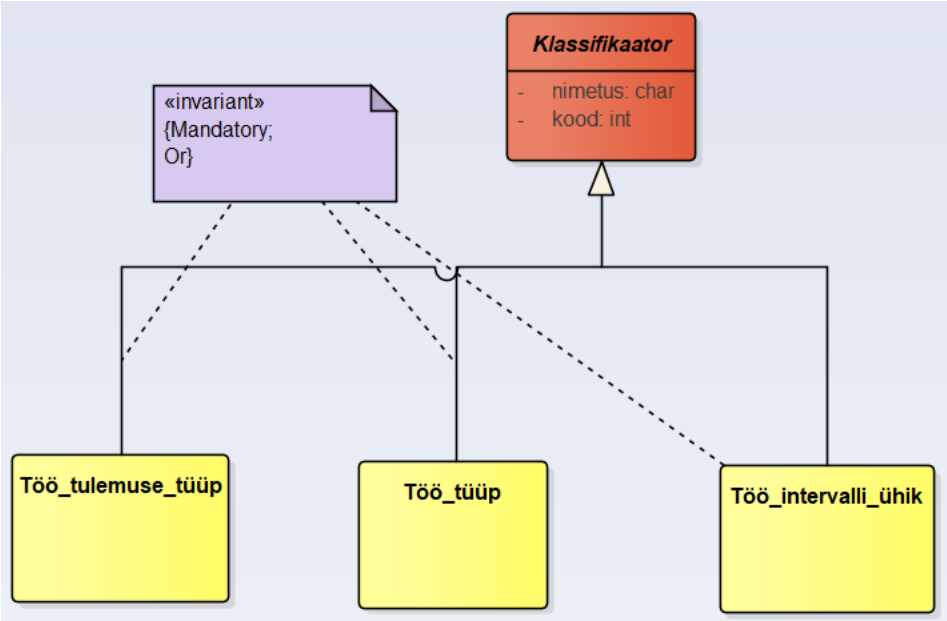
Nõue	**Kirjeldus**
Andmebaasi-süsteem	Arhiivilahenduses peab andmete hoidmiseks kasutama PostgreSQL andmebaasisüsteemis loodud andmebaasi.
Andmete muutumatus	Süsteemis salvestatud sündmuste pakid ning saadetud pakkide metaandmed peavad olema muutumatud. See tähendab, et neid ei saa tabeli loomise skriptis defineeritud säilitamisaja jooksul muuta ega kustutada. Andmete muutumatuse tagamiseks kasutatakse sündmuste aheldamist.
Andmete säilitamise aeg	Arhiivis salvestatud sündmused tuleb pärast seadistatud aega kustutada. Säilitamisaeg on reguleeritud teenusepakkuja poolt ning võib olla iga kliendi jaoks erinev. Sündmused koos metaandmetega kustutatakse *pg_partman* laienduse abil vastavalt säilitamispoliitikale.
Idempotentsus	Kui süsteemi saadetakse ühesugused sündmuste pakid, siis süsteem peab seda tuvastama ja korduseid mitte salvestama. Vastus kliendi salvestamise palvele peab olema samasugune nagu juba salvetatud sündmuste paki puhul ja see vastus peab sisaldama sündmuste paki räsi.
JSONL formaat	Süsteem peab olema võimeline võtma vastu kirjeid JSONL [4] formaadis: nii mitu kirjet korraga kui ka ühekaupa.

Nõue	**Kirjeldus**
Laiendatavus	Süsteem peab olema projekteeritud ja ehitatud nii, et tulevikus oleks võimalikult lihtne lahendust edasi arendada ja uut funktsionaalsust lisada.
Paindlikkus	Arhiivilahendus peab toetama mitut riiki ning mitmeid teenusepakkujaid ühe riigi kohta.
Plokiahel	Süsteemis peavad kõik kliendi poolt saadetud sündmuste pakid olema aheldatud. Sündmuste pakid räsitakse SHA-256 räsimisalgoritmi abil.
Skaleeritavus	Iga riigi jaoks peab olema kasutusel oma andmebaas ja iga selles riigis oleva teenusepakkuja jaoks peab olema andmebaasis oma skeem. Iga riigi andmete teenindamiseks on kasutusel üks või mitu sõlme (arvutit). Peab olema võimalik lisada juurde uusi sõlmi.
Teenusepakkuja andmete struktuur	Antud lahenduse nõudeks on toetada konkreetset andmete struktuuri. Ühes ja samas andmebaasis peab kõikides arhiveeritavaid andmeid sisaldavates skeemides olema ühesugune tabelite komplekt ning ühesugune tabelite struktuur. See tähendab, et kõik ühes riigis olevad teenuspakkujad peavad saatma sama struktuuriga andmeid, kuid iga teenuspakkuja andmed salvestatakse eraldi skeemi.

4.5 Nõuded süsteemi baasandmetele

Selles peatükkis kirjeldatakse detailselt ja mittetehniliselt funktsionaalsete allsüsteemide vajatavate registrite struktuuri. Punasega on tähistatud vastava registri põhiobjekt e põhiolemitüüp, kollasega vastava registri mittepõhiobjektid ning rohelisega on tähistatud teistesse registritesse kuuluvad objektid (vt **Joonis 18**-**Joonis 23**). Lisaks esitatakse diagrammidel kujutatud olemitüüpide ja atribuutide sõnalised kirjeldused (vt **Tabel 4**-**Tabel 15**).

4.5.1 Klassifikaatorite register



Joonis 18 Klassifikaatorite registri olemi-suhte diagramm.

Tabel 4 Klassifikaatorite registri olemitüüpide sõnalised kirjeldused.

Olemitüübi nimi (eesti- ja inglise keeles)	Definitsioon
Klassifikaator *Classifier*	Klassifikaatorid on „mistahes andmed, mida kasutatakse andmebaasis teiste andmete liigitamiseks või andmebaasis olevate andmete seostamiseks väljaspool organisatsiooni vastutusala oleva informatsiooniga.“ [27]

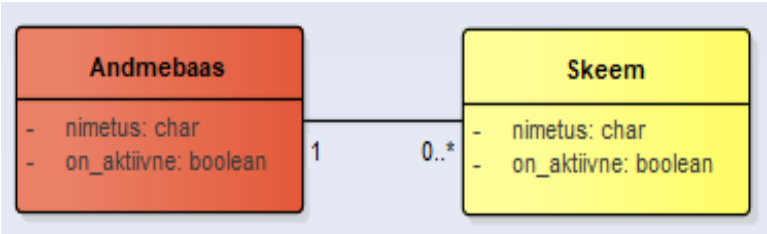
Olemitüübi nimi (eesti- ja inglise keeles)	Definitsioon
Töö_intervalli_ühik *Job_interval_unit*	Töö intervalli ühikut on vaja, et määrata, millise sagedusega töid käivitatakse. Töö intervalli ühikud on klassifikaatorid. Võimalike väärtuste näited on SECONDS ja MINUTES.
Töö_tulemuse_tüüp *Job_result_type*	Töö tulemuse tüüp näitab, milline on kontrolli tulemus. Töö tulemuse tüübid on klassifikaatorid. Võimalike väärtuste näited on COMPLETED ja FAILED.
Töö_tüüp *Job_type*	Töö tüüp näitab, millist töö kontrolli loogikat kasutatakse. Töö intervalli ühikud on klassifikaatorid. Võimalike väärtuste näited on GENERAL ja RANDOM.

Tabel 5 Klassifikaatorite registri atribuutide sõnalised kirjeldused.

Olemitüübi nimi	Atribuudi nimi (eesti- ja inglise keeles)	Atribuudi definitsioon	Näiteväärtus
Klassifikaator	kood *code*	Klassifikaatori väärtust iseloomustav sümbolite jada. Selle eesmärk on võimaldada klassifikaatori väärtusele lühidalt viidata. Kood võib olla tekstiline või arvuline väärtus. Koodi peaks saama võimalikult lihtsalt meelde jätta. Eesmärk on, et kui kasutaja näeb koodi, siis seostub see talle võimalikult kergesti klassifikaatori väärtusega, mida see kood esitab. **{Klassifikaatori tüübi piires unikaalne identifikaator. Tekstiline kood ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}**	s
Klassifikaator	nimetus *name*	Klassifikaatori väärtuse ametlik nimetus.	GENERAL

Olemitüübi nimi	Atribuudi nimi (eesti- ja inglise keeles)	Atribuudi definitsioon	Näiteväärtus
		{Klassifikaatori tüübi piires unikaalne identifikaator. Nimetus ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	

4.5.2 Andmebaaside register



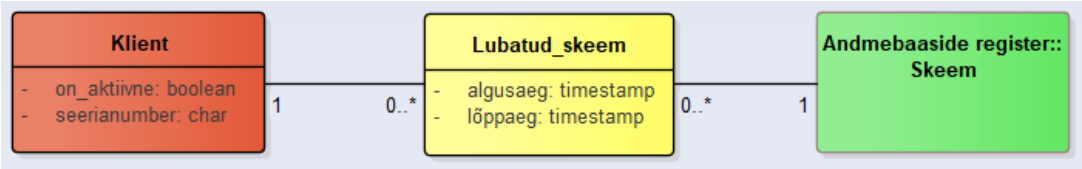
Joonis 19 Andmebaaside registri olemi-suhte diagramm.

Tabel 6 Andmebaaside registri olemitüüpide sõnalised kirjeldused.

Olemitüübi nimi (eesti- ja inglise keeles)	Definitsioon
Andmebaas *Database*	Andmebaas, milles sündmuseid salvestatakse. Iga riigi kohta, milles asuvate teenusepakkujaid andmeid soovitakse salvestada, luuakse eraldi andmebaas.
Skeem *Schema*	Andmebaasi skeem (*SQL-schema*), mille tabelitesse salvestatakse ühe teenusepakkuja andmeid. Skeem on mingil põhjusel kokkukuuluvaks määratud skeemiobjektide kogum. Riigis võib olla rohkem kui üks teenusepakkuja ja iga sellise teenusepakkuja jaoks luuakse riigile vastavas andmebaasis eraldi skeem. Kõikides ühele riigile vastavates skeemides on ühesuguse struktuuriga tabelid, sest ühe ja sama riigi erinevate teenusepakkujate korral on vaja arhiveerida sama tüüpi sündmuste andmeid.

Tabel 7 Andmebaaside registri atribuutide sõnalised kirjeldused.

4.5.3 Klientide register



Joonis 20 Klientide registri olemi-suhte diagramm.

Tabel 8 Klientide registri olemitüüpide sõnalised kirjeldused.

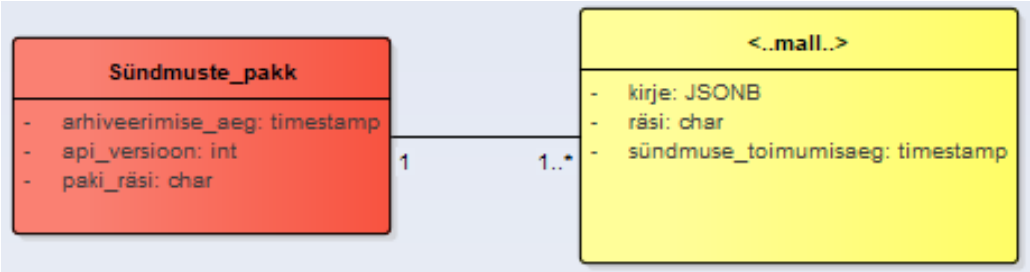
Olemitüübi nimi (eesti- ja inglise keeles)	**Definitsioon**
Klient *Client*	Klient on kas füüsiline isik või rakendus, kes saab arhiivi sündmuste pakke saata.
Lubatud_skeem *Allowed_schema*	Iga kliendi õigus mingisse skeemi sündmuste pakke saata. Kliendil võib olla mitu lubatud skeemi, mis tähendab, et üks klient saab saata sündmuste pakke erinevatesse skeemidesse. Erinevad kliendid saavad saata sündmuste pakke ühte ja samasse skeemi samal ajal. Ühel ja samal kliendil ei tohi olla ühe ja sama skeemi jaoks kahte kattuvat ajaperioodi, kui ta saab sellesse skeemi andmeid saata.

Tabel 9 Klientide registri atribuutide sõnalised kirjeldused.

Olemitüübi nimi	**Atribuudi nimi (eesti- ja inglise keeles)**	**Atribuudi definitsioon**	**Näiteväärtus**
Klient	on_aktiivne *is_active*	Tõeväärtus, mis näitab, kas kliendil on lubatud sündmuste pakke saata (True) või mitte (False). Selleks, et klient saaks saata sündmuste pakke, peab olema andmebaasis aktiivne see klient, tema riigi andmebaas ja kliendile lubatud skeem. **{Registreerimine on kohustuslik}**	True
Klient	seerianumber *serial_number*	Kliendi seerianumber. Andmete saatmisel peab päringule lisama X509 sertifikaadi. Iga sertifikaat sisaldab seerianumbrit. Selleks, et kliendil oleks võimalus andmeid saata, peab	ADFEB2B

Olemitüübi nimi	Atribuudi nimi (eesti- ja inglise keeles)	Atribuudi definitsioon	Näiteväärtus
		süsteemi olema lisatud klient, millel on sama seerianumber nagu sertifikaadis, mida ta kasutab. **{Registreerimine on kohustuslik. Kliendi unikaalne identifikaator. Väärtus ei tohi olla tühi string ja ainult tühimärkidest koosnev string. }**	
Lubatud_ skeem	algusaeg *start_datetime*	Näitab, mis maailmaaja (UTC) ajast saab klient sündmuste pakke saata. Algusaja väärtuseks on õiguse lisamise hetke maailmaaeg. **{Registreerimine on kohustuslik. Väärtus ei tohi olla suurem kui lõppaeg. Ühel ja samal kliendil ei tohi ühel ja samal ajal alata ühe ja sama skeemi kasutamise õigus rohkem kui üks kord.}**	2021-04-25 14:30:00
Lubatud_ skeem	lõpuaeg *end_datetime*	Näitab, mis maailmaaja (UTC) ajani saab klient andmeid saata. Kui lõpuaeg pole teada, siis on lõpuaeg lõpmatus (’infinity’). Sellisel juhul saab klient saata andmeid kuni lubatud skeemile määratakse mingi muu lõpuaeg. **{Registreerimine on kohustuslik. Väärtus ei tohi olla väiksem kui algusaeg. }**	2021-05-25 14:30:00

4.5.4 Sündmuste pakkide register



Joonis 21 Sündmuste pakkide registri olemi-suhte diagramm.

Tabel 10 Sündmuste pakkide registri olemitüüpide sõnalised kirjeldused.

Olemitüübi nimi (eesti- ja inglise keeles)	Definitsioon
Sündmuste_pakk *Event_batch*	Sündmuste pakk koosneb ühest või mitmest sündmusest, mille andmeid soovitakse digitaalses arhiivis säilitada.
<..mall..> <..template..>	Kõik mingit ühte tüüpi sündmuse andmed, mis on saadetud ühe teenusepakkuja poolt. Sisaldab saadetud sündmust JSON formaadis koos sündmuse metaandmetega. Tabeli nimi sõltub saadetava sündmuse tüübist. Oletame, et riigi kõik teenusepakkujad peavad arhiveerima sündmuseid, mis on tüüpi A, B või C. See tähendab, et riigile vastava andmebaasi kõikides skeemides on tabelid nimega A, B ja C ning nende tabelite struktuur on ühesugune ja kirjeldatud selle malli poolt. Nende tabelite andmed on registreeritud andmebaaside registris.

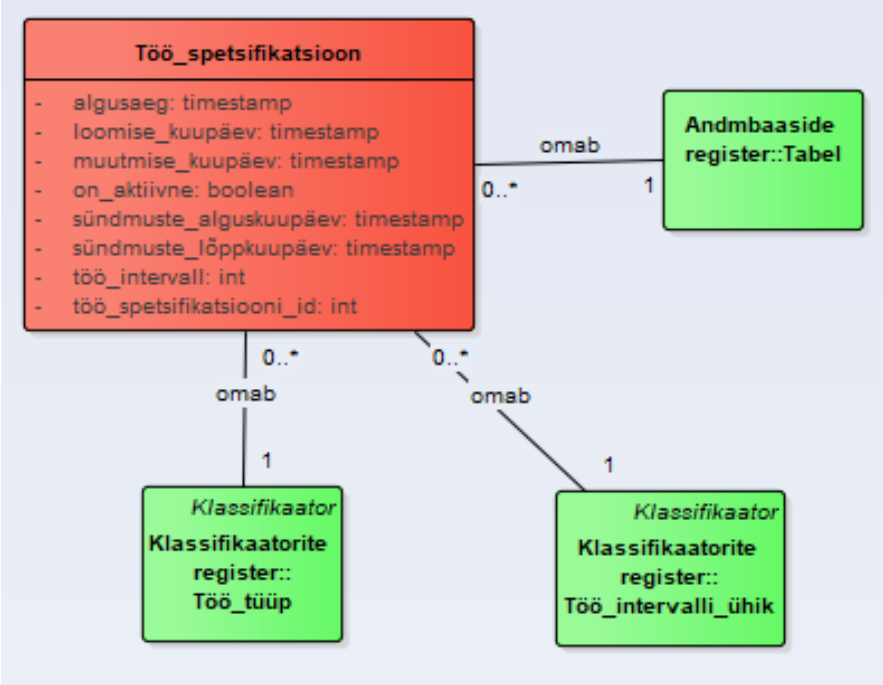
Tabel 11 Sündmuste pakkide registri atribuutide sõnalised kirjeldused.

Olemitüübi nimi	Atribuudi nimi (eesti- ja inglise keeles)	Atribuudi definitsioon	Näiteväärtus
Sündmuste_ pakk	arhiveerimise_aeg *archive_time*	Maailmaaeg (UTC) , millal paki andmed andmebaasi salvestati. **{Registreerimine on kohustuslik.}**	2021-08-25 14:30:00

Olemitüübi nimi	Atribuudi nimi (eesti- ja inglise keeles)	Atribuudi definitsioon	Näiteväärtus
Sündmuste_ pakk	api_versioon *api_version*	Arhiivi rakendusliidese (API) versioon, mis saadetakse URL-s. Näitab, milline oli API versioon, kui sündmus arhiivi saadeti. **{Registreerimine on kohustuslik.}**	1
Sündmuste_ pakk	paki_räsi *batch_hash*	Räsi, mis on arvutatud terve paki kohta, kasutades SHA-256 räsimisalgoritmi. See aitab toetada idempotentsust ning tuvastada korduvaid pakette. **{Registreerimine on kohustuslik. Unikaalne.}**	3ebcbbcfe1fab42c59bcd5bc96ba44719bff75c8d57271c81034a23c5a609c39
<..mall..>	kirje *record*	Sündmuse andmed JSONB formaadis ilma metaandmeteta. Iga sündmus sündmuste pakis salvestatakse eraldi kirjena. Seega kui tuleb sündmuste pakk kus on 1000 sündmust, siis selle malli järgi tekkivasse tabelisse tekib 1000 rida. **{Registreerimine on kohustuslik.}**	{„id“: „81e99494-e73b-40f1-b209-249d6918465e“,“eventTime“: „2022-04-09T13:26:23+00:00“ }
<..mall..>	räsi *hash*	Räsi, mida kasutatakse sündmuste aheldamiseks ning sündmuste terviklikkuse kontrolliks. See arvutatakse	99c29b70086a9fe514524511145a9b852a4bc0f83bfa9efcdf5242ab213849be

Olemitüübi nimi	Atribuudi nimi (eesti- ja inglise keeles)	Atribuudi definitsioon	Näiteväärtus
		iga sündmuse jaoks kasutades SHA-256 räsimisalgoritmi. **{Registreerimine on kohustuslik.}**	
<..mall..>	sündmuse_ toimumisaeg *event_time*	Maailmaaeg (UTC), millal sündmus toimus. See peab sisalduma sündmuse andmetes. **{Registreerimine on kohustuslik.}**	2021-06-01 00:00:00

4.5.5 Töö spetsifikatsioonide register



Joonis 22 Töö spetsifikatsioonide registri olemi-suhte diagrammid.

Tabel 12 Töö spetsifikatsioonide registri olemitüüpide sõnalised kirjeldused.

Olemitüübi nimi (eesti- ja inglise keeles)	Definitsioon
Töö_spetsifikatsioon *Job_specification*	Töö spetsifikatsioonid määravad sündmuste andmete kontrollimiseks tehtavate tööde omadused. Põhimõtteliselt võiksid süsteemis lisaks andmete kontrollimisele toimuda ka muud tüüpi tööd, kuid süsteem seda hetkel ei võimalda, sest nõuded, mille järgi süsteem loodi, seda ei soovinud. Töö spetsifikatsioon määrab, millal kontrolli alustatakse, milliseid sündmuseid millises tabelis kontrollitakse ning mis sagedusega kontrolli teostatakse. Kui riigis on X teenusepakkujat, selles riigis on vaja säilitada andmeid N erinevat tüüpi sündmuse kohta ja nende sündmuste andmeid tuleb kontrollida, siis riigile vastavas andmebaasis tekib X skeemi. Igas sellises skeemis on N tabelit. Süsteemis tuleb defineerida X*N töö spetsifikatsiooni (üks iga tabeli kohta).

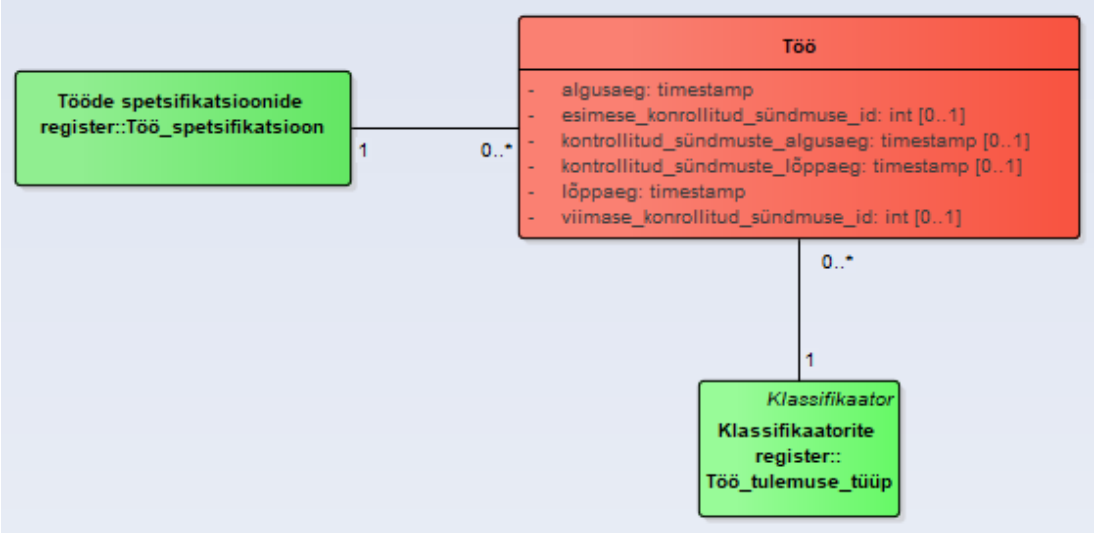
Tabel 13 Töö spetsifikatsioonide registri atribuutide sõnalised kirjeldused.

Olemitüübi nimi	Atribuudi nimi (eesti- ja inglise keeles)	Atribuudi definitsioon	Näiteväärtus
Töö_ spetsifikatsioon	töö_spetsifikatsiooni_id *job_specification_id*	Töö spetsifikatsiooni unikaalne identifikaator. Aitab tagada töö spetsifikatsiooni unikaalsust. **{Registreerimine on kohustuslik.}**	1
Töö_ spetsifikatsioon	algusaeg *starttime*	Maailmaaeg (UTC), millal peab alustama töö spetsifikatsiooni alusel tööde tegemist. **{Registreerimine on kohustuslik. Väärtus ei tohi olla väiksem kui loomise aeg. }**	2021-04-25 14:30:00

Olemitüübi nimi	Atribuudi nimi (eesti- ja inglise keeles)	Atribuudi definitsioon	Näiteväärtus
Töö_ spetsifikatsioon	loomise_aeg *created_at*	Maailmaaeg (UTC) aeg, millal töö spetsifikatsioon andmebaasi salvestati. **{Registreerimine on kohustuslik. Väärtust ei tohi tagantjärgi muuta. Väärtuse aasta peab olema vahemikus 2000-2100. Ei tohi olla suurem kui muutmise kuupäev. }**	2021-06-01 12:00:00
Töö_ spetsifikatsioon	muutmise_aeg *updated_at*	Maailmaaeg (UTC), millal töö spetsifikatsiooni viimati muudeti. **{Registreerimine on kohustuslik. Ei tohi olla väiksem kui loomise aeg.}**	2021-06-01 12:00:00
Töö_ spetsifikatsioon	on_aktiivne *is_active*	Tõeväärtus, mis näitab, kas töö spetsifikatsiooni alusel saab töid teha (True) või mitte (False). **{Registreerimine on kohustuslik.}**	True
Töö_ spetsifikatsioon	sündmuste_algusaeg *event_start_datetime*	Maailmaaeg (UTC), mis näitab, mis päevast ja mis kellaajast alates salvestatud sündmuseid peab tabelis kontrollima. Näiteks kui sündmuste algusaeg on 2021-05-01 12:00:00, siis kontrollitakse neid sündmuseid, mis olid salvestatud 2021-05-01 12:00:00 ja hiljem. **{Registreerimine on kohustuslik. Väärtuse aasta**	2021-05-01 12:00:00

Olemitüübi nimi	Atribuudi nimi (eesti- ja inglise keeles)	Atribuudi definitsioon	Näiteväärtus
		peab olema vahemikus 2000-2100. }	
Töö_ spetsifikatsioon	sündmuste_lõppaeg *event_end_datetime*	Maailmaaeg (UTC), mis näitab, mis päevani ja mis ajani toimunud sündmuseid peab tabelis kontrollima. Näiteks kui sündmuste lõppaeg on 2021-06-01 12:00:00, siis kontrollitakse kõiki sündmuseid, mis olid salvestatud enne 2021-06-01 12:00:00. **{Registreerimine on kohustuslik. Väärtus ei tohi olla väiksem kui sündmuste alguskuupäev. Väärtuse aasta peab olema vahemikus 2000-2100. }**	2021-06-01 12:00:00
Töö_ spetsifikatsioon	töö_intervall *job_interval*	Arvuline väärtus, mis näitab tööde käivitamise sagedust. Selle ühik on määratud seose kaudu töö intervalli ühikuga. **{Registreerimine on kohustuslik. Väärtus peab olema positiivne täisarv. }**	10

4.5.6 Tööde register



Joonis 23 Tööde registri olemi-suhte diagramm.

Tabel 14 Tööde registri olemitüüpide sõnalised kirjeldused.

Olemitüübi nimi (eesti- ja inglise keeles)	Definitsioon
Töö *Job*	Ühe teostatud sündmuste terviklikkuse kontrollimise tulemus. Selle käigus kontrollitakse töö spetsifikatsiooni alusel konkreetses tabelis (konkreetse sündmuse tüübi) salvestatud sündmuste andmete muutumatust.

Tabel 15 Tööde registri atribuutide sõnalised kirjeldused.

Olemitüübi nimi	Atribuudi nimi (eesti- ja inglise keeles)	Atribuudi definitsioon	Näiteväärtus
Töö	algusaeg *start_datetime*	Maailmaaeg (UTC), millal algas töö ehk teiste sõnadega sündmuste kontrollimine. **{Registreerimine on kohustuslik. Väärtus ei tohi olla suurem kui lõppaeg. }**	2021-06-01 12:00:00

Olemitüübi nimi	Atribuudi nimi (eesti- ja inglise keeles)	Atribuudi definitsioon	Näiteväärtus
Töö	esimese_kontrollitud_ sündmuse_id *first_checked_event_id*	Esimese kontrollitud sündmuse unikaalne identifikaator. Näitab, millisest sündmusest alustati kontrollimist. **{Väärtus ei tohi olla suurem kui viimase kontrollitud sündmuse id.}**	100
Töö	kontrollitud_sündmuste algusaeg *checked_events_start_ datetime*	Maailmaaeg (UTC), mis näitab, millisest kuupäevast ja kellaajast alanud sündmuseid kontrolliti. **{Väärtus ei tohi olla suurem kui kontrollitud sündmuste lõppaeg. Väärtuse aasta peab olema vahemikus 2000-2100.}**	2021-01-01 00:00:00
Töö	kontrollitud_sündmuste_ lõppaeg *checked_events_end_ datetime*	Maailmaaeg (UTC), mis näitab, millise kuupäevani ja kellaajani toimunud sündmuseid kontrolliti. **{Väärtus ei tohi olla väiksem kui kontrollitud sündmuste algusaeg. Väärtuse aasta peab olema vahemikus 2000-2100.}**	2021-02-01 00:00:00
Töö	lõppaeg *end_datetime*	Maailmaaeg (UTC), millal töö lõppes ehk lõpetati sündmuste kontrollimine. **{Väärtus ei tohi olla väiksem kui algusaeg.}**	2021-06-01 13:00:00
Töö	viimase_kontrollitud_ sündmuse_id *last_checked_event_id*	Viimase kontrollitud sündmuse unikaalne identifikaator. Näitab, milline oli viimane kontrollitud sündmus. Kui töö käigus leitakse terviklikkuse viga, siis peatub töö	200

Olemitüübi nimi	Atribuudi nimi (eesti- ja inglise keeles)	Atribuudi definitsioon	Näiteväärtus
		esimese terviklikkuse veaga sündmuse juures, töö tulemuse tüüp on FAILED ja selle sündmuse identifikaator salvestatakse viimase kontrollitud identifikaatorina. Kui töö käigus sellist viga ei leita, siis töö tulemuse tüüp on COMPLETED ja salvestatakse viimane töö käigus kontrollitud identifikaator. **{Väärtus ei tohi olla väiksem kui töö käigus kontrollitud esimese sündmuse id.}**	

5 Tehniline lahendus

Selles peatükis kirjeldatakse loodava süsteemi tehnilist lahendust.

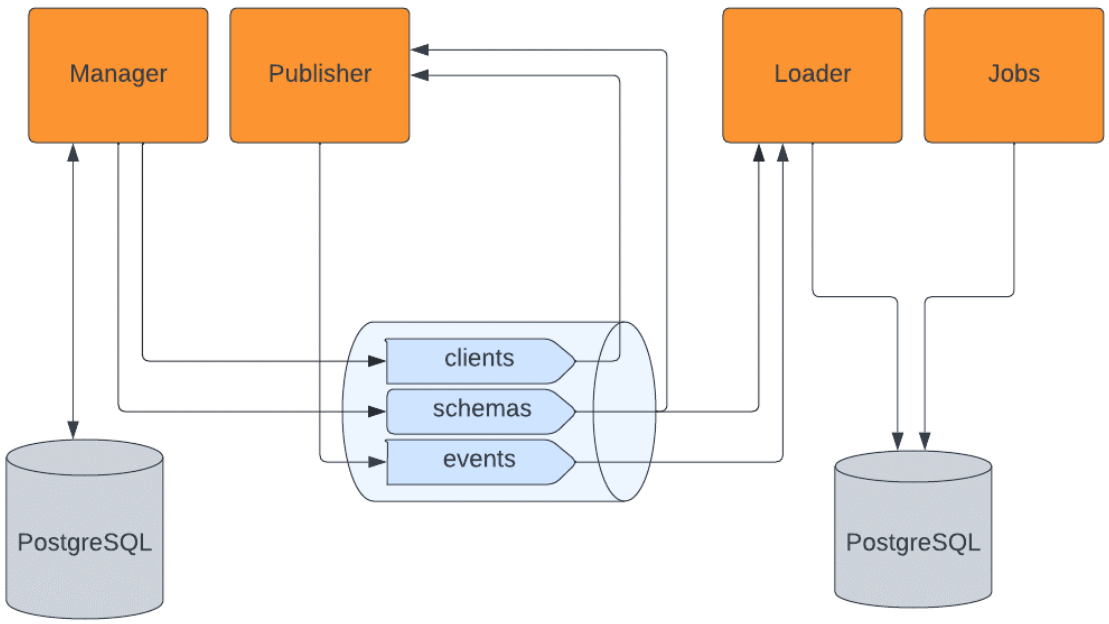
5.1 Tehniline arhitektuur

Arhiivi rakendus on realiseeritud kasutades mikroteenuste arhitektuuri. See võimaldab süsteemi laiendata horisontaalselt, kui kasutajate hulk suureneb või andmete saatmise kiirus suureneb ning tekib vajadus jagada koormust mitme serveri vahel.

Süsteem on jagatud neljaks üksteisest vähe sõltuvaks mikroteenuseks. *Publisher*, *Loader* ja *Manager* mikroteenused suhtlevad omavahel kasutades asünkroonset sündmuste andmete vahetamist. Kui mingis mikroteenuses tekkib uus sündmus, mida on vaja teiste mikroteenustega jagada, siis see saadetakse Kafka ootejärjekorda (*queue*), kust sündmust on võimalik lugeda. Tänu sellele ei pea kõik mikroteenused olema korraga üleval ja töötama.

Loader ja *Jobs* jagavad andmebaasi, aga selle põhjuseks on, et *Jobs* peab kontrollima andmeid, mis on salvestatud *Loader* mikroteenuse poolt.

Joonis 24 esitab arhiivi rakenduse arhitektuuri.



Joonis 24 Arhiivi rakenduse arhitektuuri diagramm.

Tabel 16 näitab, milline mikroteenus realiseerib millised funktsionaalse allsüsteemi kasutusjuhud. Nagu tabelist näha, siis *Manager* mikroteenus realiseerib klassifikaatorite-, andmebaaside- ja klientide funktsionaalsete allsüsteemide funktsionaalsust. Selline lähenemine valiti teadlikult, kuna nende allsüsteemide kasutusjuhte kasutatakse arhiivi seadistamiseks ja ettevalmistamiseks. Mugavuse mõttes otsustati kõiki administreerimisega seotud funktsionaalsuseid hoida ühes kohas. Vajadusel on võimalik jagada *Manager* mikroteenus kolmeks mikroteenusteks ning jagada üks andmebaas samuti kolmeks andmebaasiks.

Tabel 16 Mikroteenuste ja funktsionaalsete allsüsteemide seos koos kasutusjuhtudega.

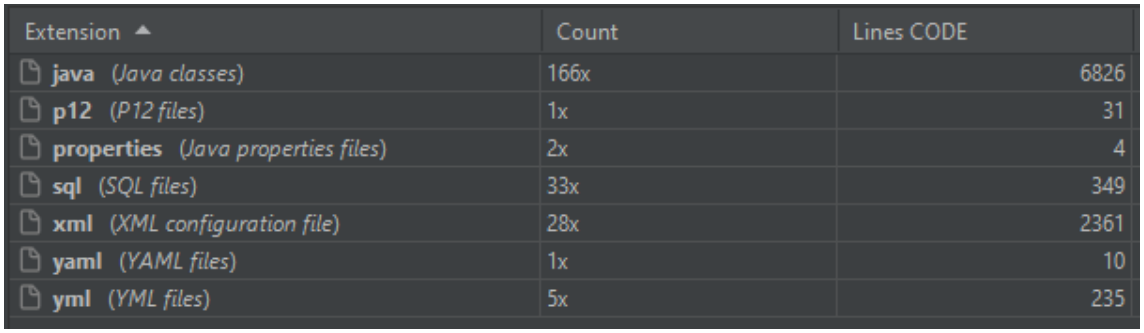
Funktsionaalne allsüsteem	**Kasutusjuht funktsionaalses allsüsteemis**	**Mikroteenus, mis seda kasutusjuhtu realiseerib**
Klassifikaatorite funktsionaalne allsüsteem	Lisa töö intervalli ühikud Lisa töö tüübid Lisa töö tulemuse tüübid	Manager
Andmebaaside funktsionaalne allsüsteem	Lisa uus andmebaas Aktiveeri andmebaas Muuda andmebaas mitteaktiivseks Vaata kõiki andmebaase Lisa uus skeem Aktiveeri skeem Muuda skeem mitteaktiivseks Vaata kõike skeeme	Manager

Funktsionaalne allsüsteem	Kasutusjuht funktsionaalses allsüsteemis	Mikroteenus, mis seda kasutusjuhtu realiseerib
	Vaata kõike tabeleid	
Klientide funktsionaalne allsüsteem	Loo klient Aktiveeri klient Muuda klient mitteaktiivseks Muuda klienti Lisa lubatud skeem Kustuta lubatud skeem Vaata kõike kliente	Manager
Töö spetsifikatsioonide funktsionaalne allsüsteem	Lisa töö spetsifikatsioon Aktiveeri töö spetsifikatsioon Muuda töö spetsifikatsioon mitteaktiivseks Muuda töö spetsifikatsiooni Vaata töö spetsifikatsioone Vaata aktiivseid töö spetsifikatsioone Vaata mitteaktiivseid töö spetsifikatsioone	Jobs
Tööde funktsionaalne allsüsteem	Salvesta töö Vaata kõiki tööde tulemused	Jobs

Funktsionaalne allsüsteem	Kasutusjuht funktsionaalses allsüsteemis	Mikroteenus, mis seda kasutusjuhtu realiseerib
	Vaata ebaõnnestunud töid	
Sündmuste pakkide funktsionaalne allsüsteem	Saada sündmuste pakk	Publisher
	Salvesta sündmuste pakk	Loader

5.2 Rakendus

Selles jaotises kirjutatakse rakenduse realisatsioonist. Rakenduses on kokku 6826 füüsilist koodirida. **Joonis 25** esitab loodud rakenduse failide ja koodiridade arvu. Antud statistika saadi IntelliJ IDEA sisseehitatud pistikprogrammi abil.



Extension	Count	Lines CODE
java (Java classes)	166x	6826
p12 (P12 files)	1x	31
properties (Java properties files)	2x	4
sql (SQL files)	33x	349
xml (XML configuration file)	28x	2361
yaml (YAML files)	1x	10
yml (YML files)	5x	235

Joonis 25 Loodud rakenduse koodi statistika

Süsteem toetab erinevaid riike. Igas riigis on üks või rohkem teenusepakkujat. Igas riigis on üks või rohkem sündmuse tüüpi, millele vastavate sündmuste andmeid tuleb arhiveerida. Iga riigi kõigi teenusepakkujate kohta tuleb koguda sama tüüpi sündmuseid. Erinevat tüüpi sündmuste korral võidakse arhiivis talletada erinevaid andmeid, st vastavate JSON dokumentide struktuur on erinev.

Riigile vastab andmebaas, riigis toimetavale teenusepakkujale vastab skeem selles andmebaasis ja riigis olulisele sündmuse tüübile vastab eraldi tabel kõigis nendes

skeemides. Selles tabelis on muuhulgas JSONB tüüpi veerg vastavat tüüpi sündmuste andmete hoidmiseks.

5.2.1 Kafka teema ülesehitus

Kafka ootejärjekorda nimetatakse teemaks (*topic*). Igal Kafka teemal võib olla mitu tootjat ja tellijat. Tootjateks on rakendused, mis kirjutavad sündmuseid teemasse. Tellijateks on rakendused, mis loevad sündmuseid.

Arhiivis peab olema loodud iga skeemi (riigis tegutseva teenusepakkuja) kohta eraldi teema. Iga teema nimi peab vastama mustrile:

archive_<andmebaasi_nimi>_<skeemi_nimi>,

kus **<andmebaasi_nimi>** ja **<skeemi_nimi>** viitavad, millisesse andmebaasi ja selle skeemi sündmuseid salvestatakse.

Selline lähenemine annab võimaluse saata ja hoida erinevate riikide ja teenuspakkujate andmeid üksteisest eraldatuna.

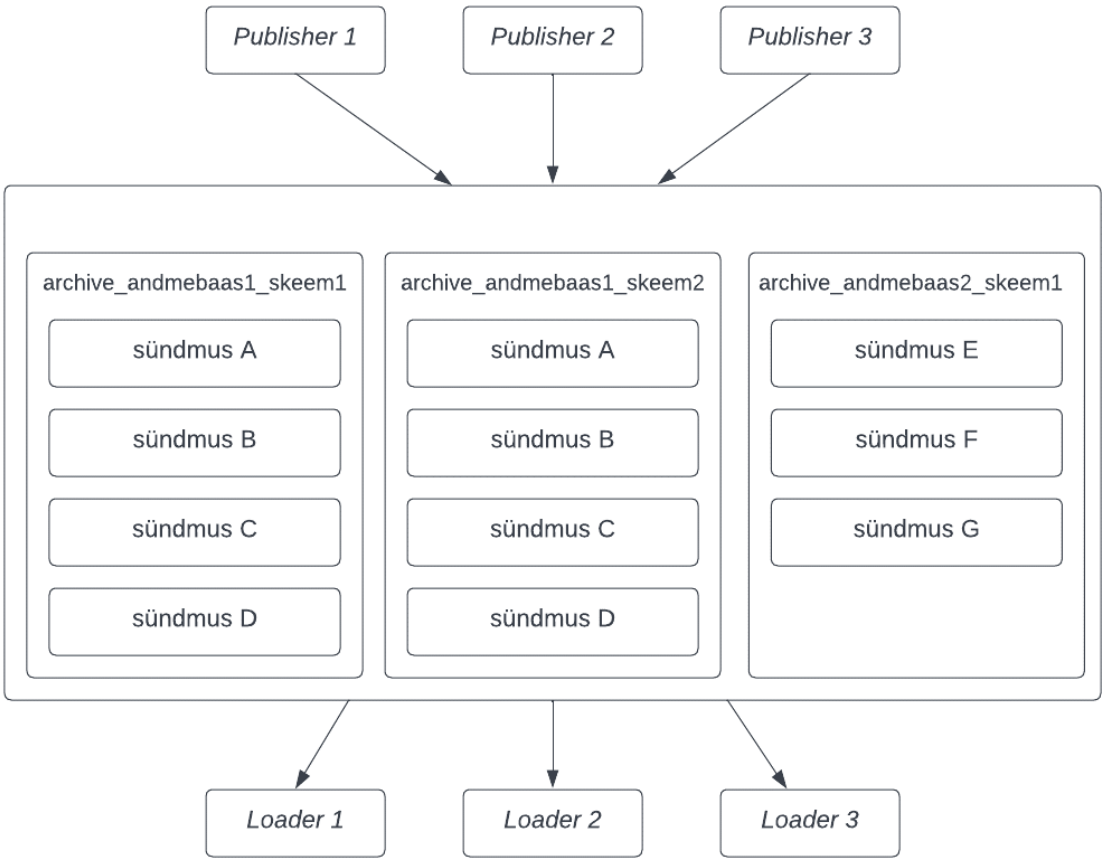
Lisaks sündmuste saatmiseks mõeldud teemadele tuleb luua veel kaks teemat:

1. **archive_clients** – teema, kus hoitakse andmeid klientide kohta, kes saavad arhiivi sündmuste pakke saata,

2. **archive_schemas** – teema, kus hoitakse andmeid kõikide skeemide kohta, kuhu on võimalik sündmuste pakke saata.

Kuna teemades võib olla korraga väga palju andmeid, siis on need jagatud sektsioonideks (*partition*) ning iga sündmuse tüübi jaoks peab olema oma sektsioon.

Joonis 26 illustreerib kolme Kafka teemat ja nende sektsioone. Kaks teemat on skeemide jaoks, mis asuvad sama andmebaasis (nendes on ühe ja sama riigi kahe teenusepakkuja arhiveeritavad andmed). Nendes teemades on neli sektsiooni erinevate sündmuste tüüpide jaoks. Üks teema on teise andmebaasi ja skeemi (riigi ja teenusepakkuja) jaoks, kus on kolm sektsiooni erinevate sündmuste tüüpide jaoks.

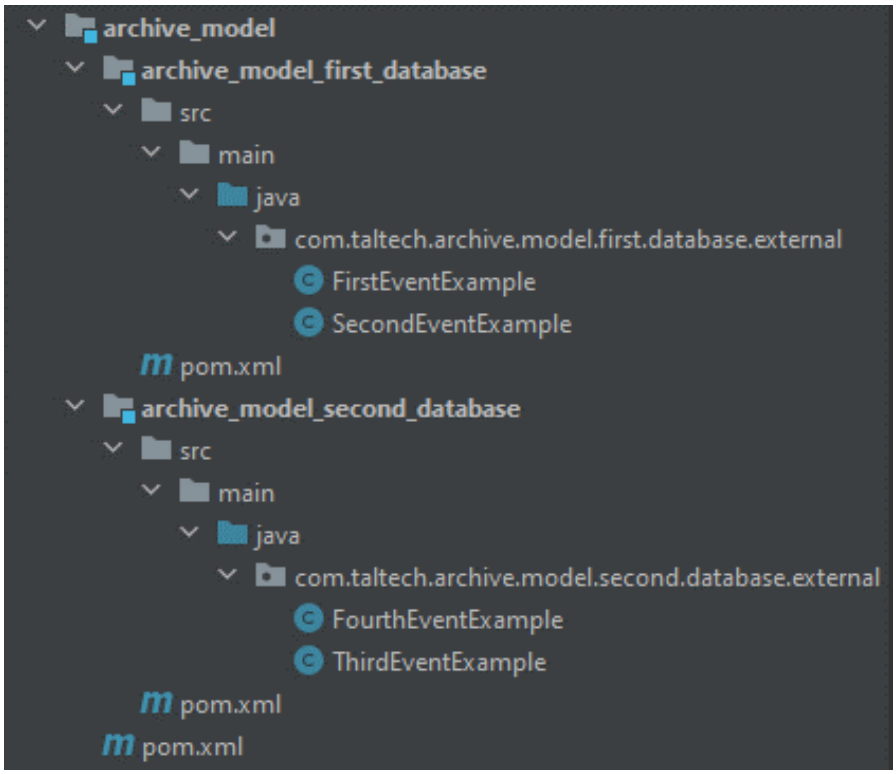


Joonis 26 Kafka teemad ja sektsioonid.

5.2.2 Sündmused

Süsteem võtab vastu ainult neid sündmuseid, millele on süsteemis loodud vastav mudel ehk klass. Iga sündmuse tüübi kohta tuleb luua eraldi klass. Klassid on jagatud erinevatesse rakenduse moodulitesse ning iga moodul vastab ühele andmebaasile. Moodul vastab riigile ja klass vastab sündmuse tüübile. Oletame, et meil on kaks riiki ja kokku kolm teenusepakkujat. Kõigi nende puhul on vaja koguda andmeid sündmuste kohta, mis on tüüpi T (üks ja sama tüüp). Rakenduses luuakse sellisel juhul sündmuse tüübile T vastavalt kaks klassi, üks on ühe riiki moodulis ja teine on teise riiki moodulis. See dubleerimine on vajalik, sest erinevate riikide mudelites on klassidel erineva sisuga annotatsioonid.

Joonis 27 esitab näite, kuidas on moodulid ja klassid organiseeritud.



Joonis 27 Rakenduse moodulid sündmuste klassidega erinevate andmebaaside jaoks.

Joonis 28 näitab, kuidas peab välja nägema andmete klass. Igal klassil peab olema *@ExternalModel* annotatsioon, milles on andmebaasi nimi. *@Id* näitab millisesse sektsiooni tuleb sündmus saata.

```java
/**
 * First event example
 */
@Data
@Builder
@Validate
@ExternalModel(database = "first_database")
@Id(0)
@NoArgsConstructor
@AllArgsConstructor
@FieldDefaults(level = AccessLevel.PRIVATE)
public class FirstEventExample implements Record {

    /**
     * Id
     */
    String id;

    /**
     * Event time
     * Format: yyyy-mm-ddThh:mm:ss+|-hh:mm
     */
    @Pattern(regexp = DATE_TIME_PATTERN)
    @JsonDeserialize(using = DateTimeHandler.class)
    String eventTime;
}
```

Joonis 28 Klassi näide.

5.2.3 Arhiivi seadistamine ja ettevalmistamine

Manager komponendi eesmärk on lihtsustada arhiivi ettevalmistamist ja seadistamist. *Manager* on mõeldud organisatsiooni siseseks kasutamiseks, mis tähendab, et seda kasutavad ainult DevOps-id või andmebaasi administraatorid.

Manager komponent vastutab kõikide arhiivis kasutatavate andmebaasi skeemide ja tabelite loomise eest. Kõik need luuakse Liquibase abil. Liquibase on avatud lähtekoodiga tarkvara, mis võimaldab jälgida, hallata ning rakendada andmebaasiskeemi muudatusi [28].

Manager komponendi eesmärgid on järgmised.

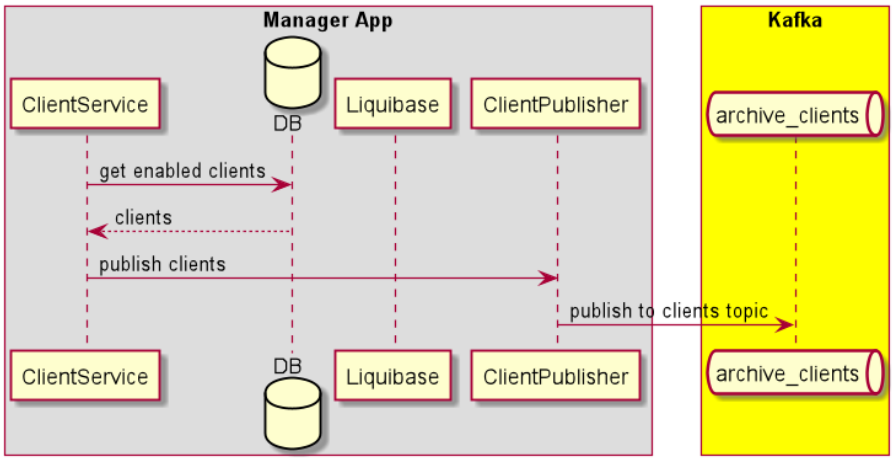
- Klientide ja klientide õiguste haldamine.
- Kliendile kasutuseks lubatud skeemide haldamine (lisamine/kustutamine).
- Uute skeemide ja tabelite loomine.

5.2.3.1 Klientide haldamine

Selleks, et klient saaks andmeid arhiivi saata, peab *Client* tabelisse salvestama kliendi andmed. Samuti peavad iga kliendi jaoks olema defineeritud kõik andmebaasid ja skeemid, kuhu tal on lubatud andmeid saata. Need õigused peavad olema lisatud *Allowed_schema* tabelisse.

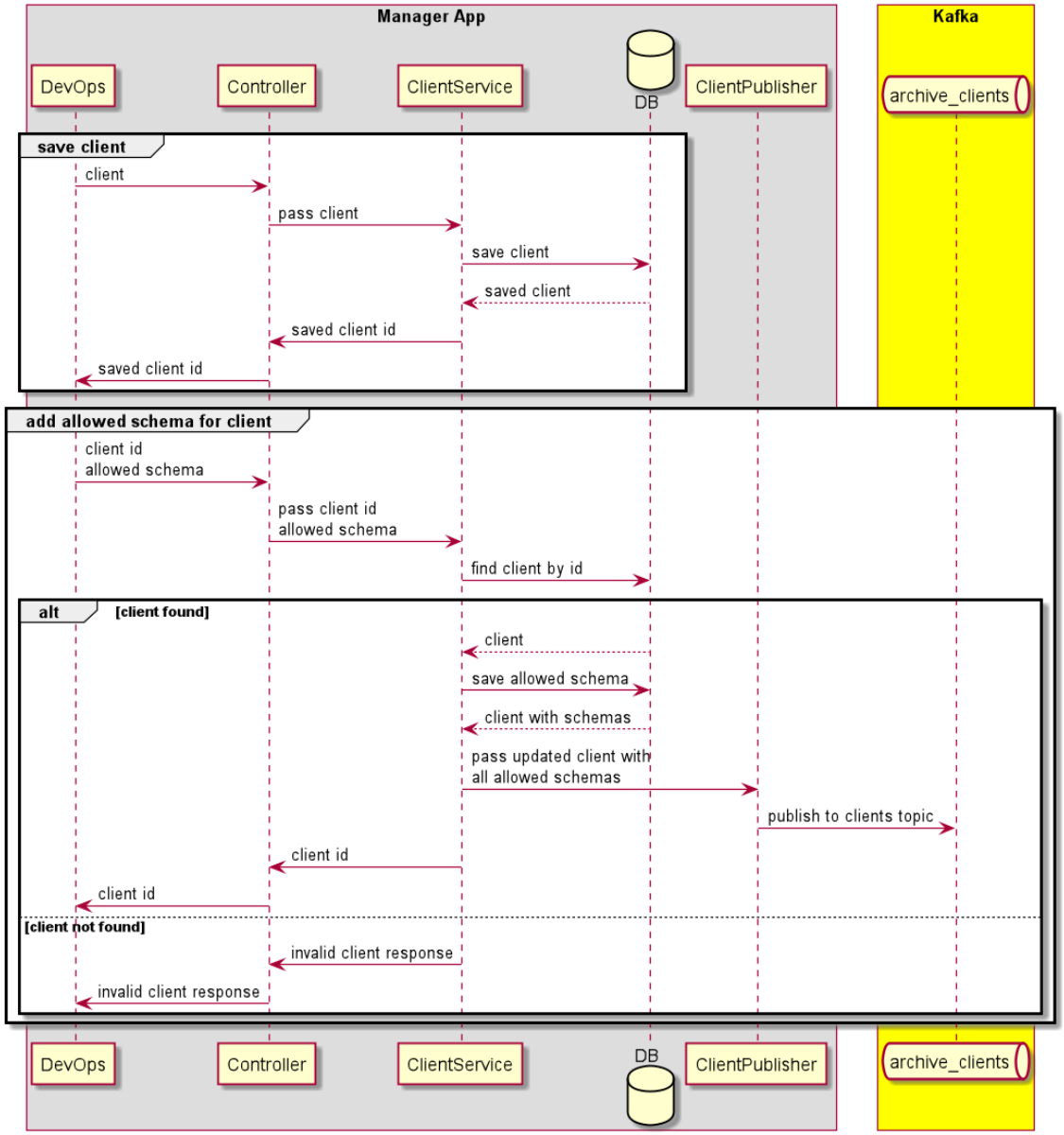
Kõiki kliente koos andmete salvestamiseks lubatud skeemidega hoitakse Kafka teemas *archive_clients*. Selline lähenemine annab võimaluse hoida *Manager* komponenti iseseisva komponendina, mis ei sõltu teistest komponentidest. Samuti on ka teistel komponentidel klientide andmetele ligipääs.

Manager komponendi käivitamisel küsitakse andmebaasist kõik olemasolevad kliendid ja nende õigused ning saadud andmed saadetakse Kafka teemasse. Kui andmeid muudetakse, siis uuendatud andmed saadetakse uuesti Kafka teemasse. **Joonis 29** esitab komponendi käivitamisel andmete Kafkasse saatmise jadadiagrammi.



Joonis 29 Kliendi andmete Kafkasse saatmise jadadiagramm komponendi käivitamisel.

Joonis 30 on esitatud kliendi salvestamine ja salvestatud kliendile õiguste lisamise jadadiagramm.

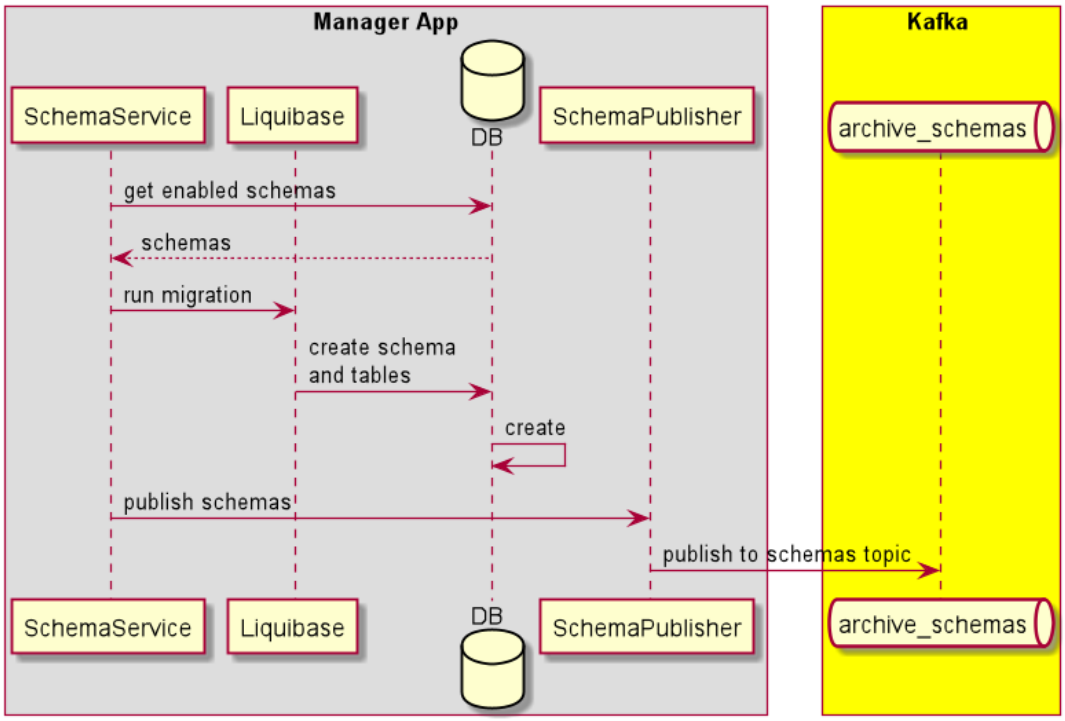


Joonis 30 Kliendi ja kliendi õiguste andmete Kafkasse lisamise jadadiagramm.

5.2.3.2 Andmebaaside, skeemide ja tabelite haldamine

Sündmuste salvestamiseks tuleb andmebaasis luua õiged skeemid ja tabelid. Kõikide skeemide andmeid hoitakse samuti Kafka teemas ning teised komponendid saavad neid andmeid Kafkast lugeda. Sarnaselt klientide andmetega, küsitakse *Manager* komponendi

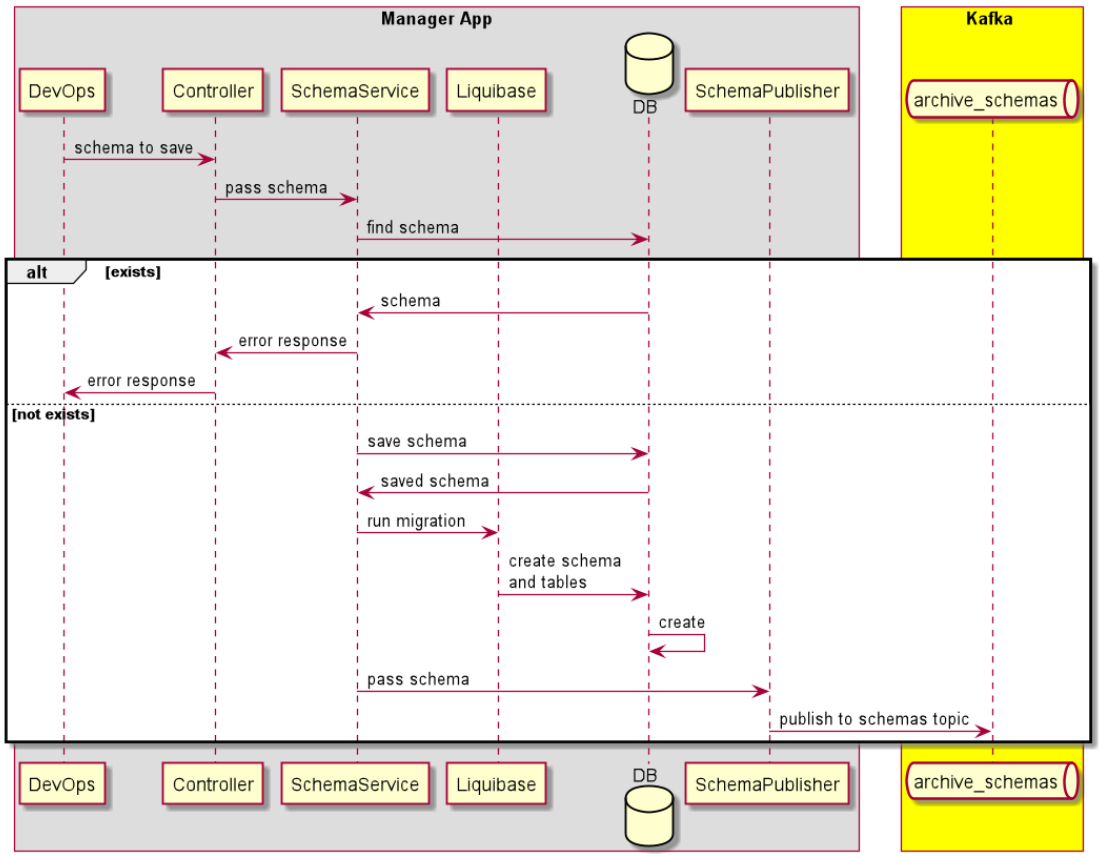
käivitamisel andmebaasist kõik olemasolevad skeemid ning saadud andmed saadetakse Kafka teemasse. **Joonis 31** esitab selle protsessi jadadiagrammi.



Joonis 31 Skeemide andmete Kafkasse saatmise jadadiagramm komponendi käivitamisel.

Kui uus skeem on *Archive_schema* tabelisse lisatud, siis andmebaasis luuakse vastav skeem ning selle skeemi all luuakse tabelid. Skeemide ja tabelite loomise skriptid peavad olema eelnevalt süsteemi lisatud. Skeemide ja tabelite loomine ja muutmine on realiseeritud Liquibase abil.

Joonis 32 esitab skeemi lisamise protsessi jadadiagrammi.

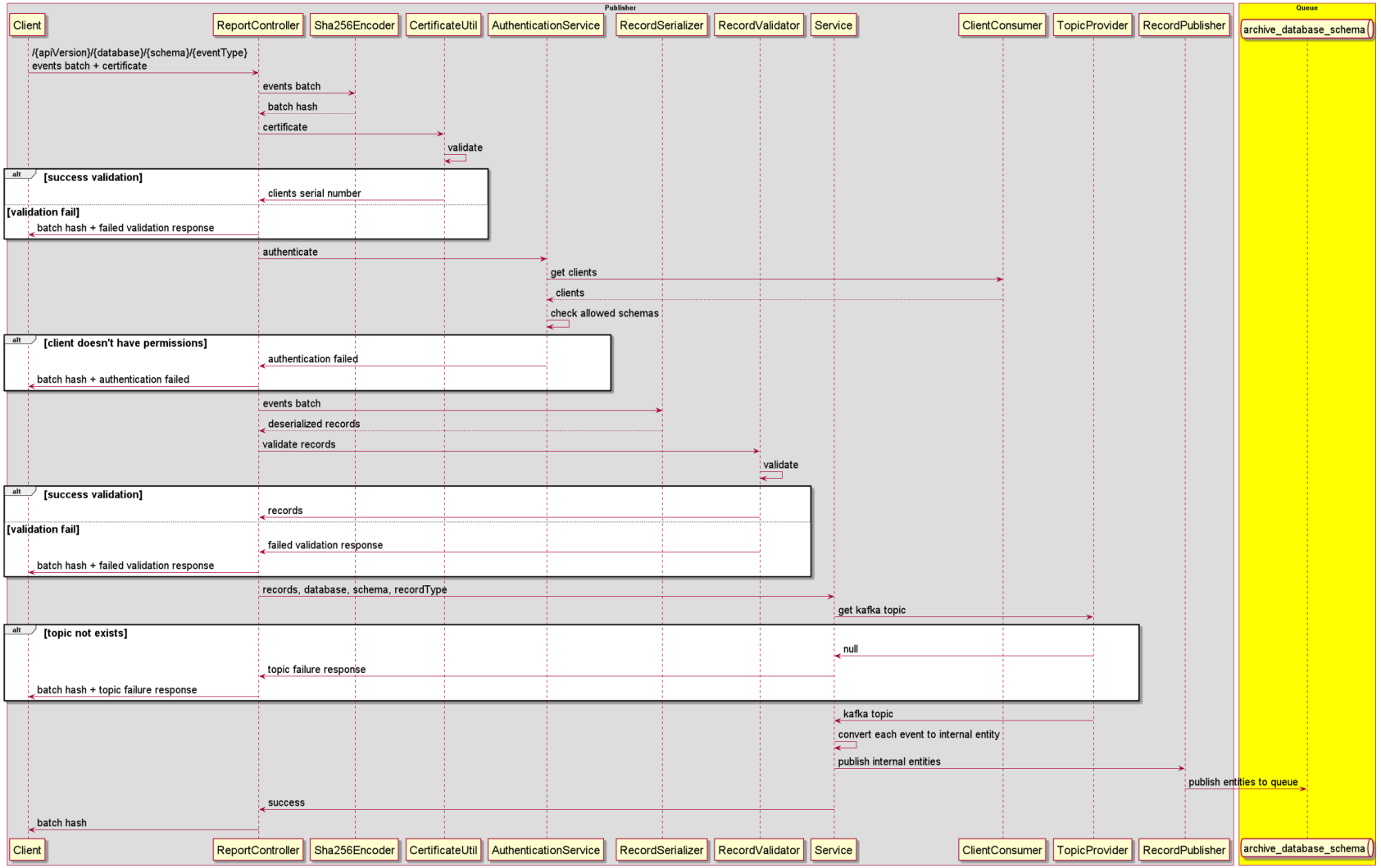


Joonis 32 Skeemide lisamise jadadiagramm

5.2.4 Sündmuste pakkide vastuvõtmine

Publisher mikroteenus vastutab sündmuste pakkide vastuvõtmise, valideerimise (vt jaotis 5.2.5) ja Kafka teemasse saatmise eest. *Publisher* on kliendile kättesaadav *REST* liidese kaudu. Klient saadab selle kaudu päringuid. Lisaks sündmustele JSONL formaadis peab päring sisaldama ka selliseid andmeid nagu andmebaasi- ja skeemi nimi, kuhu andmeid saadetakse ning sündmuse tüüp. Kuna Publisher kasutab HTTPS protokolli, siis peab päringule lisama ka X509 sertifikaadi. *Publisher* kontrollib sertifikaati valiidsust ja kliendi õigusi. Kui sellist klienti ei leidu või sellel puuduvad õigused, siis süsteem tagastab saatjale selle kohta vastuse ning andmeid ei töödelda. Klientide õigused ja lubatud skeemid loetakse Kafka teemadest ning neid hoitakse mälus.

Iga sündmuste paki kohta arvutatakse räsi, mis tagastatakse kliendile nii õnnestunud kui ka ebaõnnestunud sündmuste paki salvestamise korral. Sündmuste pakki räsi aitab tagada idempotentsust ning tuvastada korduseid. Kui salvestataval sündmuste pakil on samasugune räsi nagu juba salvestatud pakil, siis seda ei salvestata. **Joonis 33** eitab sündmuste pakkide vastuvõtmise protsessi jadadiagrammi.



Joonis 33 Sündmuste pakkide vastuvõtmise jadadiagramm

5.2.5 Sündmuste andmete valideerimine

Sündmuste pakkide vastuvõtmise käigus toimub ka sündmuste andmete valideerimine e reeglitele vastavuse kontrollimine. Valideerimisreeglid defineeritakse kliendi poolt iga sündmuse tüübi jaoks eraldi. Ühes riigis olevate teenuspakkujatel on ühesugused sündmuste tüübid ning seega ka ühesugused valideerimisreeglid. Kui sama sündmuse tüübile vastavaid sündmuseid peab arhiveerima erinevates riikides, siis võivad nendel sõltuvalt riigist olla erinevad valideerimisreeglid. Kuna iga riigi jaoks on sündmuse tüübil eraldi klass, siis võib sellega ükskõik milliseid valideerimisreegleid siduda. See mõjutab ainult seda klassi.

Kõigepealt deserialiseeritakse kõik sündmused ning JSON sündmustest saadakse Java objektid. Seejärel toimub sündmuste paki valideerimine. Kõigepealt kontrollitakse, et sündmuste arv pakis ei ületaks maksimaalset lubatud väärtust. Maksimaalne lubatud arv on konfigureeritav, aga vaikimisi väärtus on 1000. Seejärel valideeritakse iga sündmust eraldi. Kui vähemalt ühe sündmuse puhul leidub mingi viga, siis terve seda sündmust sisaldav sündmuste pakk lükatakse tagasi ning kliendile tagastatakse vastav põhjus. Tagastatavas sõnumis peab olema vale sündmuse unikaalne identifikaator ning vea kirjeldus. Sündmuse valideerimiseks kasutatakse Bean Validation 2.0 raamistiku abi. Selle kasutamiseks lisati projekti selle teek ning iga sündmuse väljale, mida peab valideerima, lisati annotatsioonid nagu näiteks @NotNull, @NotEmpty, @NotBlank ja @Min. Kindlasti kontrollitakse, et kõik kohustuslikud väljad on täidetud. Samuti kontrollitakse, et nimed ja nimetused ei koosne ainult tühikutest ning arvulistes väljades olevad väärtused ei oleks negatiivsed. Kõik kuupäevi ja aegu esitavad stringid valideeritakse, et need vastaksid konkreetsetele mustritele.

Joonis 34 illustreerib, kuidas näeb välja valideerimisreeglite rakendamine. Iga välja juures on valideerimiseks annotatsioon. Näiteks *id* väli ei tohi puududa (olla NULL), ei tohi olla tühi string ning peab koosnema 1–50 tähemärgist. *IpAddress* ei ole kohustuslik väli, kuid kui see on määratud, siis selle maksimaalne pikkus võib olla 40 tähemärki. *EventTime* ei tohi puududa (olla NULL), ei tohi olla tühi string ning väärtus peab vastama mustrile.

```java
public class Login implements Record {

    /**
     * Id
     */
    @NotEmpty
    @Size(min = 1, max = 50)
    String id;

    /**
     * Client IP address
     */
    @Size(max = 40)
    String ipAddress;

    /**
     * User's id
     */
    @NotEmpty
    @Size(min = 1, max = 40)
    String userId;

    /**
     * Event time
     * Format: yyyy-mm-ddThh:mm:ss+|-hh:mm
     */
    @NotEmpty
    @Pattern(regexp = DATE_TIME_PATTERN)
    @JsonDeserialize(using = DateTimeHandler.class)
    String eventTime;
}
```

Joonis 34 Objekti valideerimise reeglite esitamise näide.

5.2.6 Sündmuste pakkide salvestamine

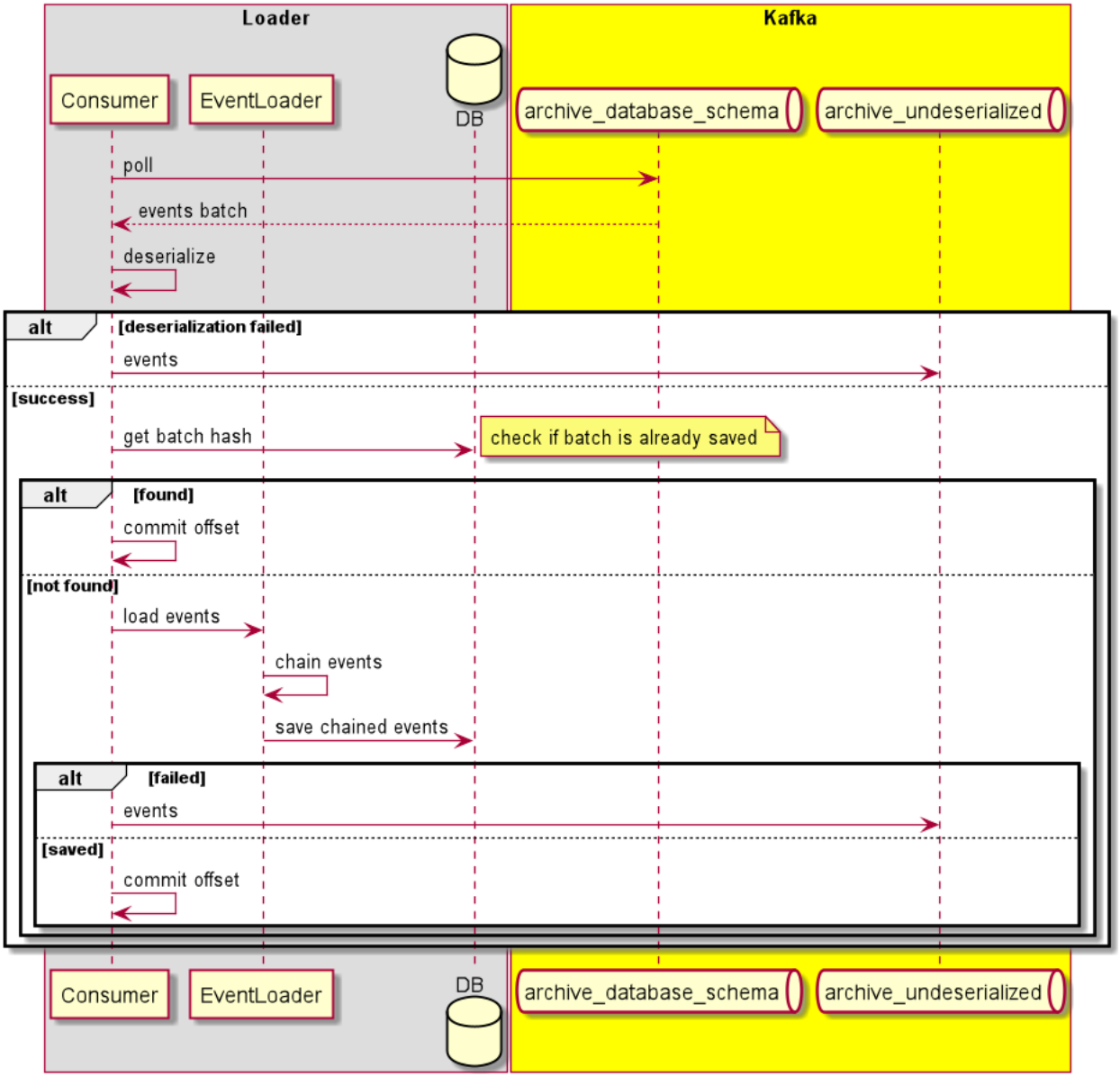
Loader mikroteenuse eesmärgid on järgmised.

- Kafka teemast sündmuste pakkide lugemine, teisendamine (väliste objektide enda sisesteks objektideks muutmine – iga sündmus deserialiseeritakse ning iga sündmuste pakk teisendatakse listiks e loendiks) ja salvestamine andmebaasi.

- Sündmuste aheldamine SHA-256 räsimisalgoritmi abil.

Sarnaselt *Publisher* mikroteenusega loeb *Loader* Kafka teemast skeemid, mis on lubatud andmete salvestamiseks.

Joonis 35 esitab sündmuste lugemise, aheldamise ja salvestamise protsessi jadadiagrammi.



Joonis 35 Sündmuste pakkide salvestamise jadadiagramm.

5.2.6.1 Aheldamine

Andmete töötlemisel aheldatakse kõik sündmuste kirjed. Kõik ühes skeemis olevad ühte tüüpi sündmused moodustavad ühe ahela. See tähendab, et iga teenuspakkuja andmed on eraldi ahelates ning iga sündmuse tüübi (ehk tabeli) kohta on eraldi ahel.

Igas sündmustega tabelis on veerg räsi jaoks. Iga uue sündmuse räsi sisaldab eelmise sündmuse räsi (vt **Joonis 36**). Sündmuste aheldamiseks võetakse iga sündmuse korral selle andmed stringina, lisatakse selle lõppu eelmise sama teenusepakkuja sama tüüpi sündmuse räsi ja saadud väärtuse põhjal arvutatakse SHA-256 räsimisalgoritmi abil selle sündmuse räsi. Samamoodi aheldatakse kõik ühes plokkis olevad andmed. Aheldatud andmed salvestatakse andmebaasi ühe transaktsioonina e tehinguna.

```
public String chainTo(String lastRecordHash) {
    return DigestUtils.sha256Hex( data: getRecordString() + lastRecordHash);
}
```

Joonis 36 Sündmuste räsi arvutamise koodi näide

Joonis 37 esitab viimase räsi küsimise ja sündmuste paki aheldamise koodi näide.

```
String lastRecordHash = eventRepository.getLastHash(database, databaseSchema, tableName);

List<InternalEntity> recordsToSave = new ArrayList<>();

for (InternalEntity record : records) {
    String hash = record.chainTo(lastRecordHash);

    record.setHash(hash);
    lastRecordHash = hash;
    record.setVersion(removeChar(record.getVersion()));

    recordsToSave.add(record);
}
```

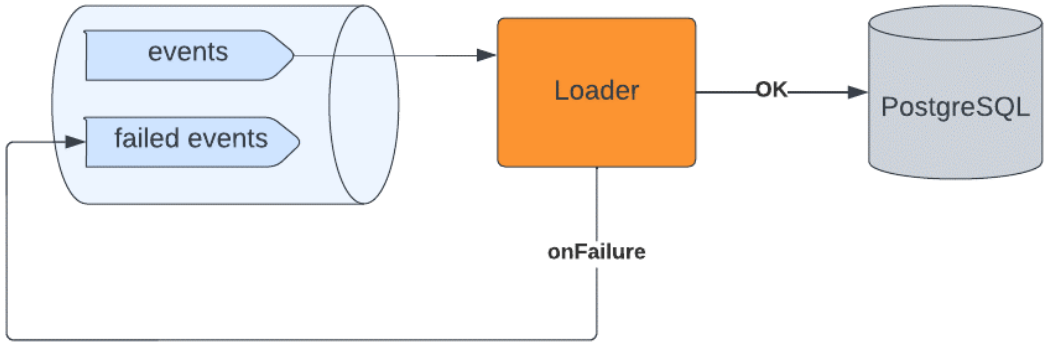
Joonis 37 Sündmuste aheldamise koodi näide.

5.2.6.2 Ebaõnnestunud sündmuste töötlemise strateegia

Mõnikord võib juhtuda, et sündmuste salvestamine ei õnnestu. Selle põhjused võivad olla erinevad nagu näiteks deserialiseerimise viga või andmebaasiga ühenduse kadumine. Selleks, et need sündmused ei läheks kaduma, saadetakse kõik neid sündmuseid sisaldavad sündmuste pakid uuesti eraldi Kafka teemasse, mis on mõeldud just ebaõnnestunud sündmuste andmete salvestamise jaoks. Siia hulka ei kuulu sündmused, mille puhul tekkis sündmuste pakkide vastuvõtmisel valideerimise viga, sest selliste sündmustega sündmuste pakke ei võta süsteem üldse vastu.

Hetkel ei ole realiseeritud ühist algoritmi ebaõnnestunud sündmuste töötlemiseks ning neid peab käsitsi üle vaatama.

Joonis 38 esitab ebaõnnestunud sündmuste töötlemise strateegiat.



Joonis 38 Ebaõnnestunud sündmuste töötlemise strateegia.

5.2.7 Sündmuste terviklikkuse kontroll

Jobs mikroteenus vastutab sündmuste terviklikkuse kontrollimise eest. Sündmuste terviklikkuse kontroll on lisafunktsionaalsus ning selle teostamine sõltub kliendi soovist.

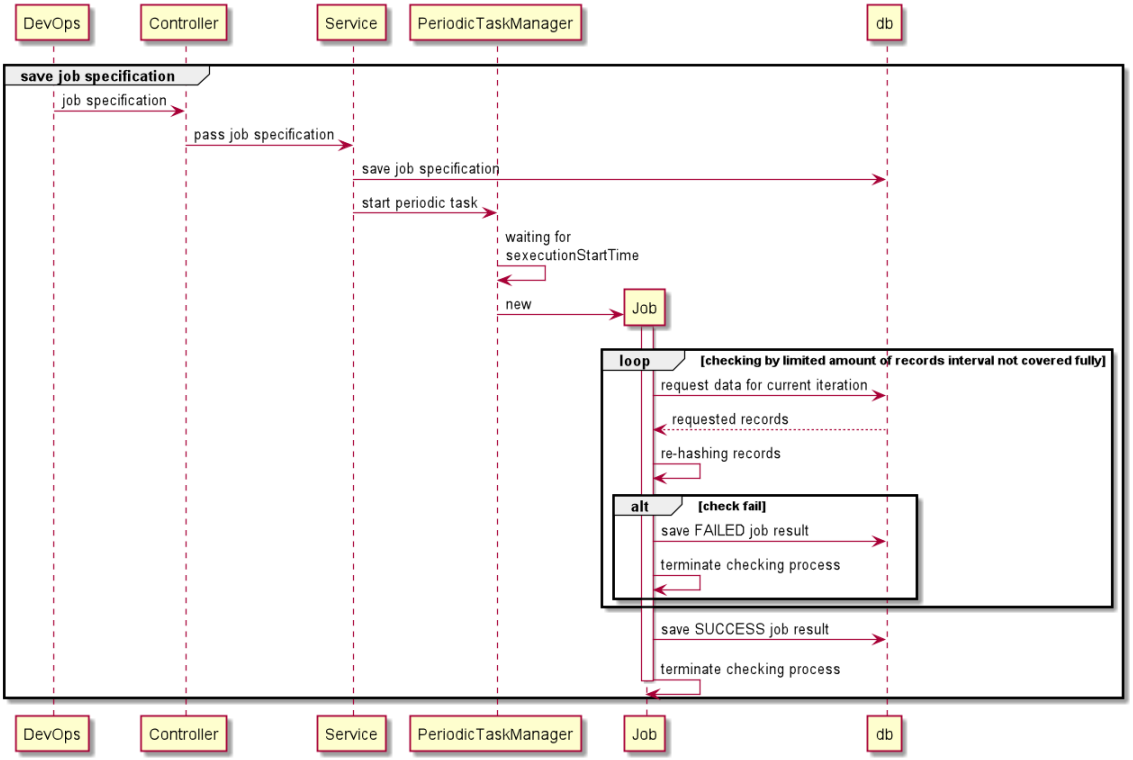
Kontrolli algatamiseks peab andmebaasi administraator või DevOps saatma REST liidese kaudu töö spetsifikatsiooni identifikaatori. Töö spetsifikatsioonis peab olema määratud, millises andmebaasis, skeemis ja tabelis peab sündmuseid kontrollima. Samuti tuleb määrata, millises ajavahemikus salvestatud sündmuseid peab kontrollima ning millise

intervalliga tuleb kontrollimise tööd teha. Täpsemalt kirjeldatakse töö spetsifikatsiooni parameetreid ja nende tähendust jaotises 4.5.5.

Tööks nimetatakse ühte kontrolliprotsessi. Igas töö protsessis kontrollitakse vajalikku andmevahemikku. Kuna ühes andmevahemikkus võib olla väga palju sündmuseid, siis teostatakse kontrolli osade kaupa. See tähendab, et andmebaasist ei küsita korraga kõiki andmevahemikku kuluvaid sündmuseid, vaid ainult piiratud arvu sündmuseid (maksimaalne arv on koodi tasemel konfigureeritav). Kontrolli ajal arvutatakse üle sündmuse räsi ning võrreldakse seda olemasolevaga. Kui need väärtused pole võrdsed, st uuesti arvutatud räsi erineb olemasolevast räsist, siis see tähendab, et sündmuse terviklikus on riknenud. Kui kogu andmevahemik on kontrollitud, siis salvestatakse andmebaasi tabelisse uus töö koos tulemusega. Erinevate töö spetsifikatsioonidega seotud töid teostatakse paralleelselt ning sõltumatult.

Kui leitakse, et andmete terviklikkus on rikutud, siis töö tulemuses määratakse, milline oli esimene rikutud sündmus kontrollitud ajavahemikus. Hetkel rikutud sündmustega midagi edasi ei tehta. Täpsemalt on igal töö tulemus kas COMPLETED (edukalt lõpetatud) või FAILED (ebaõnnestunud). Kui kontrolli tulemusel terviklikkuse viga ei leita, siis on tulemus COMPLETED ja siis salvestatakse ka esimese ja viimase kontrollitud sündmuse identifikaatorid. Kui kontroll peatus (ehk leiti vigane sündmus), siis töö tulemus on FAILED ja viimase kontrollitud sündmuse identifikaator on see sama vigase sündmuse identifikaator, mille kontrollimisel töö peatus (vt jaotis 4.5.6).

Joonis 39 näitab terviklikkuse kontrolli protsessi, mis algab töö spetsifikatsiooni saatmisega ning lõpeb kui kõik sündmused on kontrollitud.



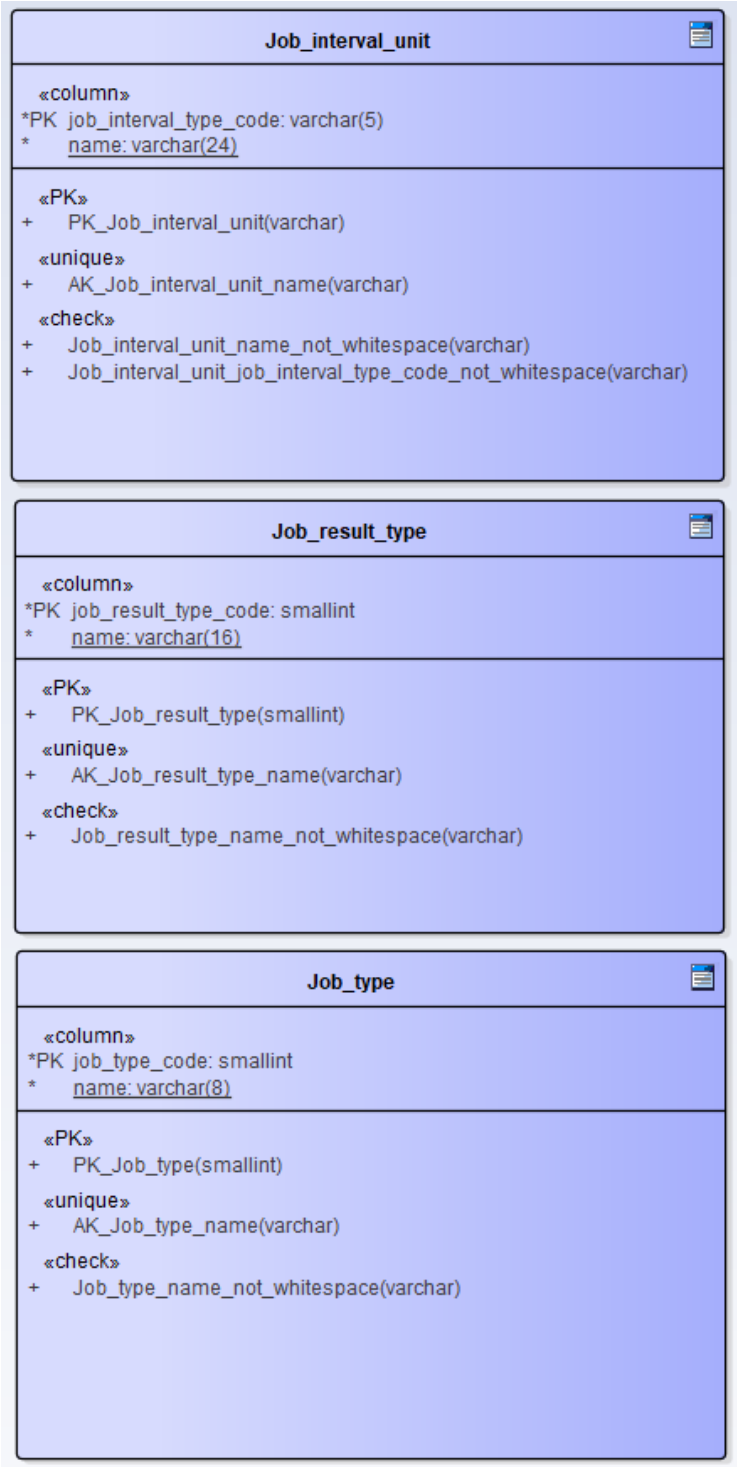
Joonis 39 Sündmuste terviklikkuse kontrolli jadadiagramm.

5.3 Andmebaas

Selles jaotises esitatakse funktsionaalsete allsüsteemide vajatavate registrite füüsilise disaini andmemudelid andmebaasisüsteemi PostgreSQL jaoks.

5.3.1 Klassifikaatorite register

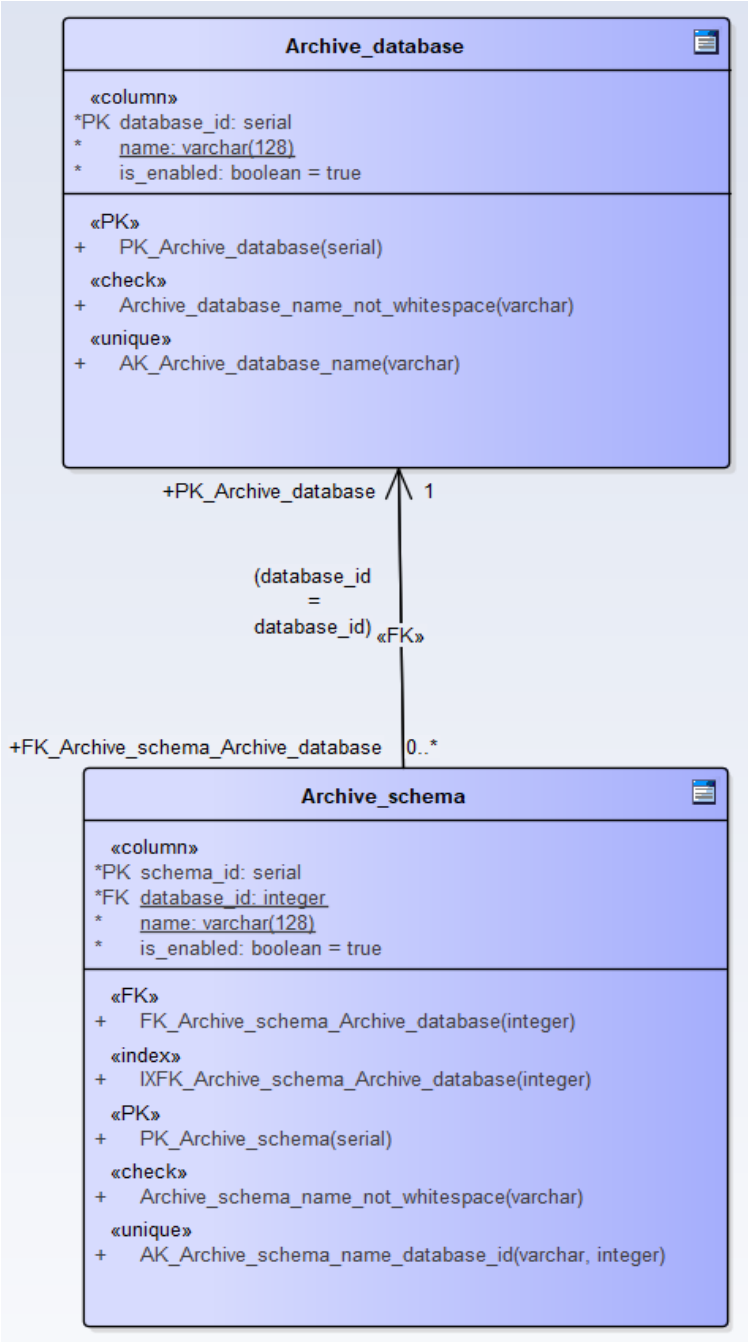
Joonis 40 esitab klassifikaatorite registri füüsilise disaini andmebaasi diagrammi.



Joonis 40 Klassifikaatorite registri füüsilise disaini andmebaasi diagramm.

5.3.2 Andmebaaside register

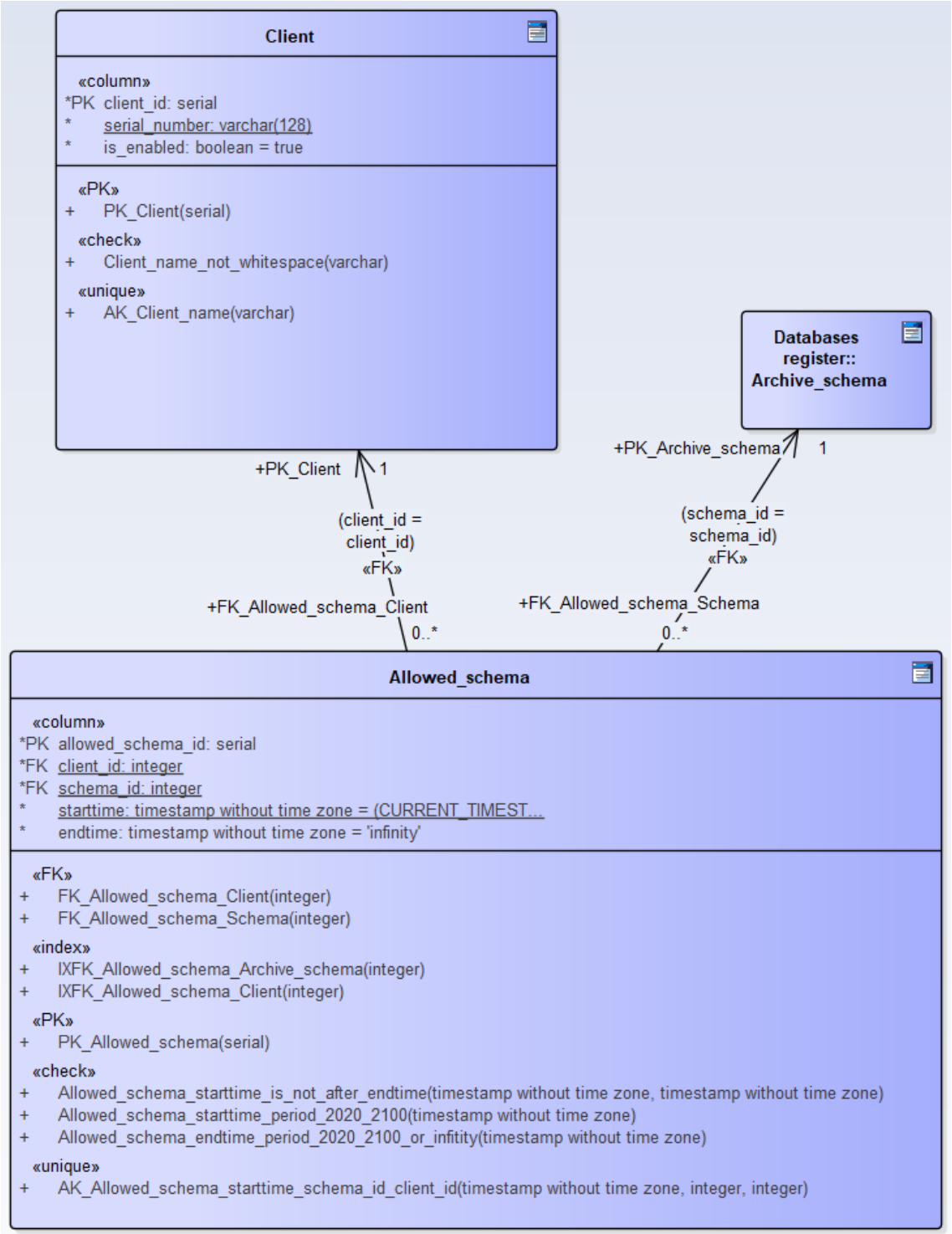
Joonis 41 esitab andmebaaside registri füüsilise disaini andmebaasi diagrammi.



Joonis 41 Andmebaaside registri füüsilise disaini andmebaasi diagramm.

5.3.3 Klientide register

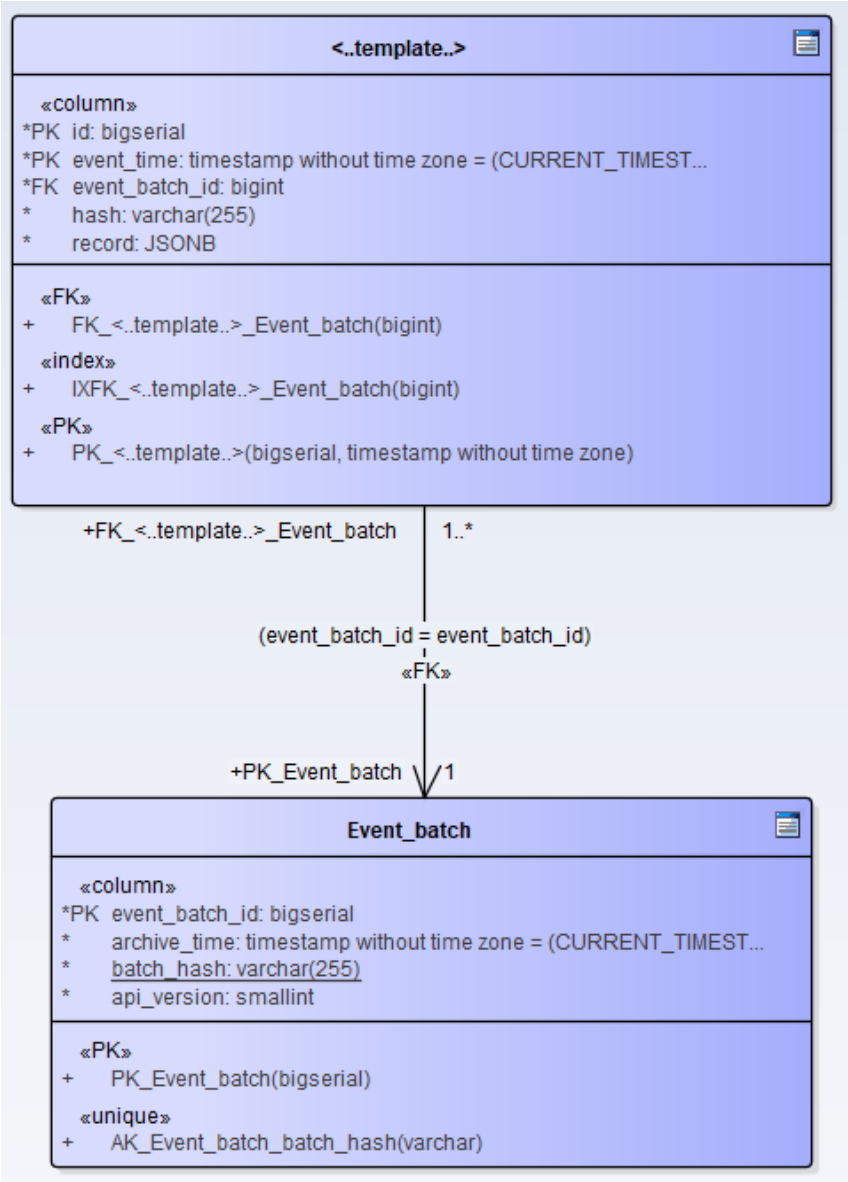
Joonis 42 esitab klientide registri füüsilise disaini andmebaasi diagrammi.



Joonis 42 Klientide registri füüsilise disaini andmebaasi diagramm.

5.3.4 Sündmuste pakkide register

Joonis 43 esitab sündmuste pakkide registri füüsilise disaini andmebaasi diagrammi.



Joonis 43 Sündmuste pakkide registri füüsilise disaini andmebaasi diagramm.

5.3.4.1 Sektsioonideks jagamine ja kustutamine

Sündmuste tabelite sektsioonideks jagamine ja andmete kustutamine realiseeritakse PostgreSQL *pg_partman* laienduse abil. *Pg_partman* annab võimaluse luua ja hallata ajal põhinevaid tabelisektsioone [29].

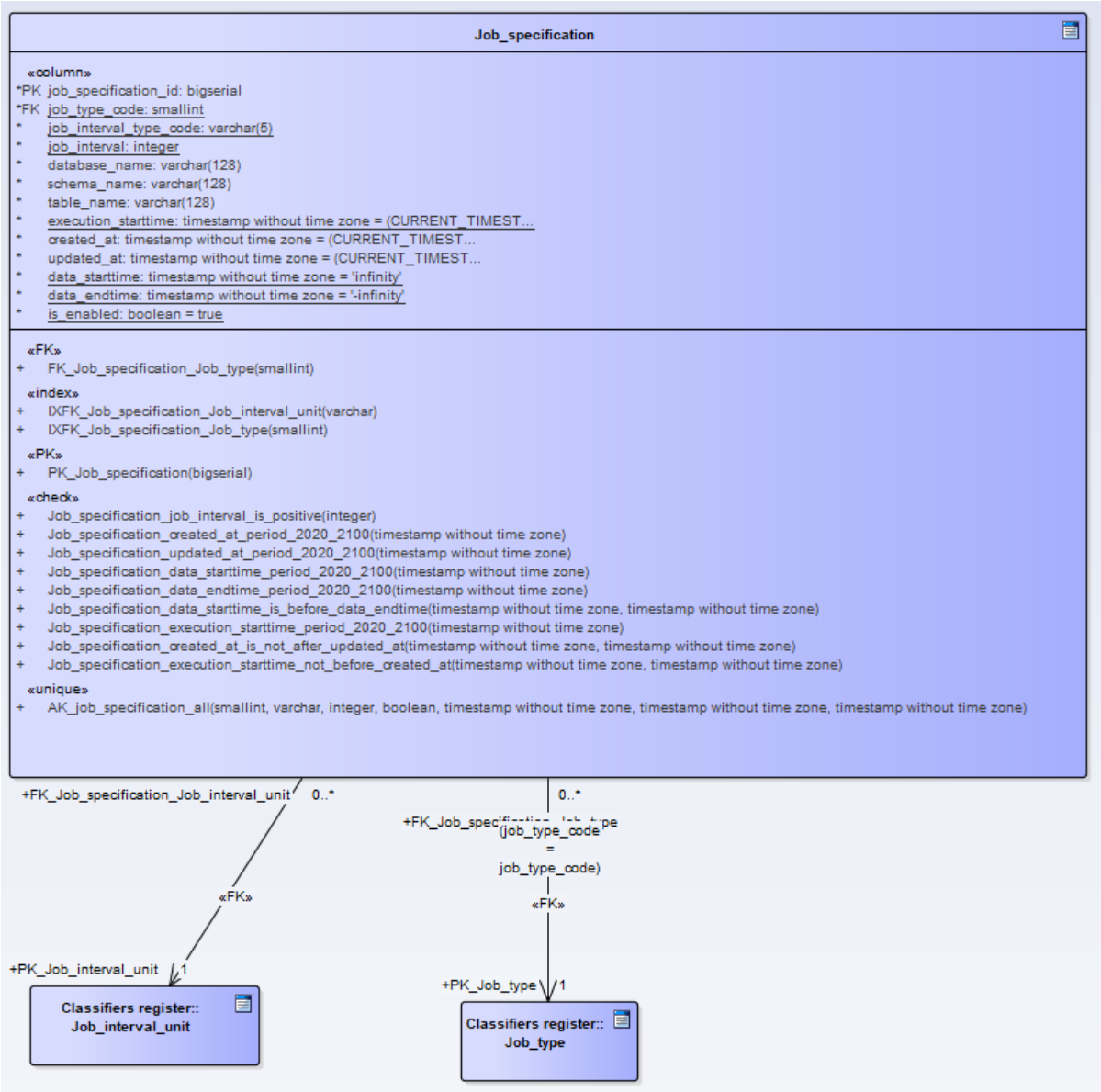
Joonis 44 esitab, milline peab olema tabeli loomise skript. Tabeli loomisel määratakse, milliste veergude järgi toimub sektsioonideks jagamine. Samuti on võimalik määrata sektsioonideks jagamise algusaeg ning säilitamisaeg.

```
CREATE TABLE IF NOT EXISTS ${schema-name}.example_event
(
    id                  BIGSERIAL,
    event_time          timestamp without time zone not null DEFAULT (CURRENT_TIMESTAMP(0) AT TIME ZONE 'utc'),
    event_batch_id      BIGINT                      not null,
    hash                varchar(255)                not null,
    record              jsonb                       not null,
    primary key (id, event_time)
) partition by range(event_time);

CREATE TABLE partman.${schema-name}_example_event (LIKE ${schema-name}.example_event);
ALTER TABLE partman.${schema-name}_example_event ADD PRIMARY KEY (id, event_time);

select partman.create_parent(
    p_parent_table=>'${schema-name}.example_event',
    p_control=>'event_time',
    p_type=>'native',p_interval =>'monthly',
    p_constraint_cols=>null,
    p_premake=>3,
    p_automatic_maintenance=>'on',
    p_start_partition=>'2022-05-01',
    p_inherit_fk=>'true',
    p_epoch=>'none',
    p_upsert=>'',
    p_publications=>null,
    p_trigger_return_null=>'true',
    p_template_table=>'partman.${schema-name}_example_event',
    p_jobmon=>'true',
    p_date_trunc_interval=>null);

update partman.part_config set retention='6 years',
                               retention_keep_table='f',
                               infinite_time_partitions='t',
                               jobmon='t',
                               inherit_privileges='t'
where parent_table='${schema-name}.example_event';
```

Joonis 44 Tabeli loomise skripti näide.

5.3.5 Töö spetsifikatsioonide register

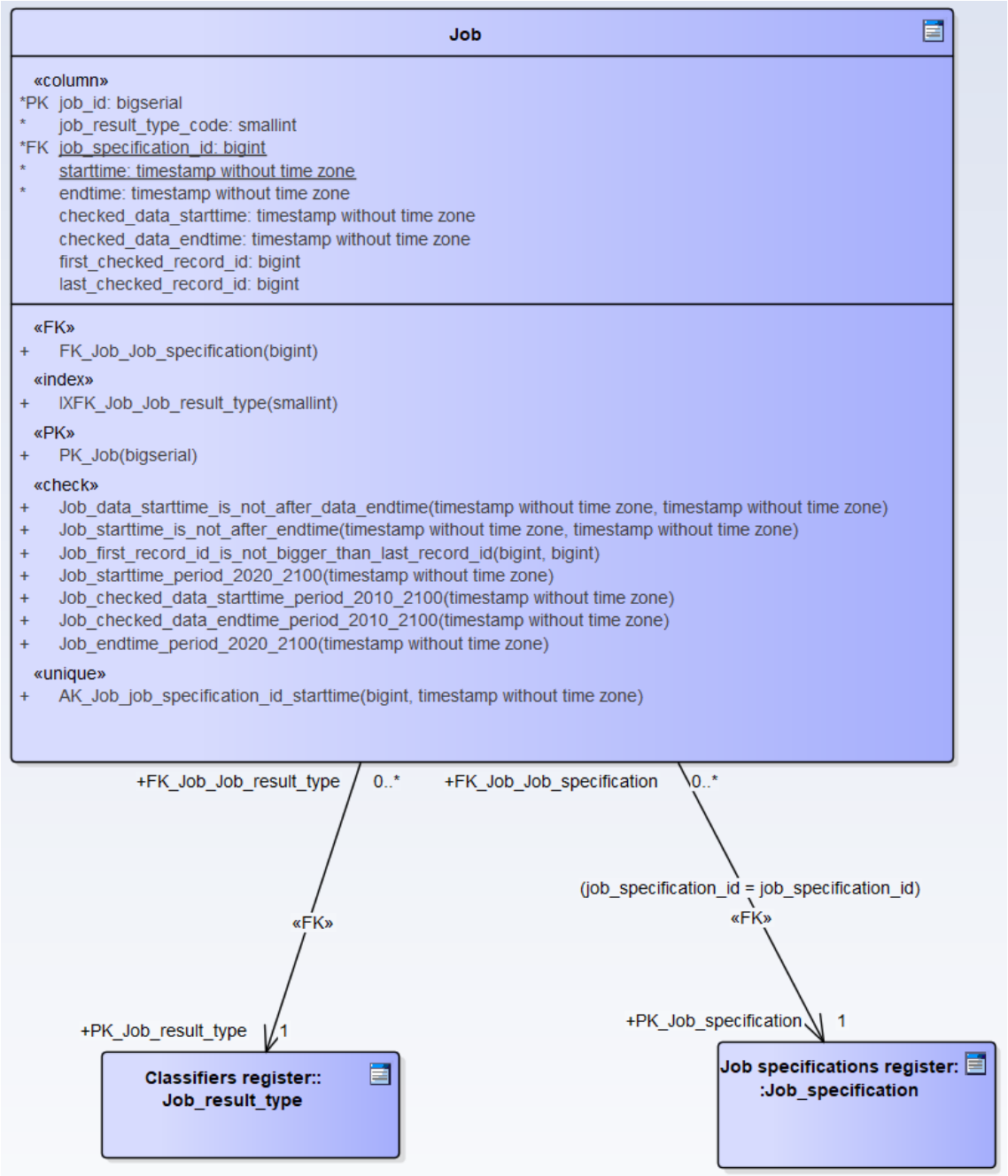
Joonis 45 esitab töö spetsifikatsioonide registri füüsilise disaini andmebaasi diagrammi.



Joonis 45 Töö spetsifikatsioonide registri füüsilise disaini andmebaasi diagramm.

5.3.6 Tööde register

Joonis 46 esitab tööde registri füüsilise disaini andmebaasi diagrammi.



Joonis 46 Tööde registri füüsilise disaini andmebaasi diagramm.

5.4 Turvalisuse tagamisest

Andmete muutumatuse kontrollist kirjutati jaotises 5.2.5. Klientide õigused andmeid saata on registreeritud klientide registris (vt jaotis 4.5.3). HTTPS protokolli kasutamisel sidepartneri identiteedi kontrollimiseks kasutatakse X509 sertifikaate (vt jaotis 3.7).

Andmebaaside hoolduse ja ülalhoiuga tegelevad andmebaasi administraatorid. Nad valmistavad ette andmebaasi, kuhu lisavad *pg_partman* laienduse. Siis annavad nad valmis andmebaasi arendajale ning jälgivad ise kõike mis andmebaasiga toimub. Andmete varundamine ja vajadusel taastamine on andmebaasi administraatorite ülesanne. Arendajad saavad valmis andmebaasis skeeme ja tabeleid luua ning õiguseid jagada.

Andmebaasi turvalisuse tagamiseks luuakse andmebaasi kasutajaid ja antakse neile õigused. Iga andmebaasiga suhtleva mikroteenuse jaoks on tehtud oma piiratud õigustega andmebaasi kasutaja. Näiteks *Loader* ja *Jobs* mikroteenused saavad ainult lugeda andmeid tabelitest ja lisada andmeid tabelitesse ning nendel puuduvad õigused andmete uuendamiseks ja kustutamiseks. Nendele teenustele vastavad kasutajad saavad kasutada erinevate riikide andmeid, st erinevaid andmebaase. Uue skeemi loomisel lisatakse kohe skeemile ja kõikidele skeemis olevate tabelitele ka pääsupiirangud. Õiguseid skeemide ja tabelite suhtes annavad arendajad ja teevad seda skeemide/tabelite loomise skriptiga.

5.5 Parandused kasutajate tagasiside alusel

Kui arhiivi esimene versioon oli valmis ja testitud, siis võeti see kasutusele. Arhiivi aktiivne kasutamine näitas, et lahenduses esinesid mõned puudused. Globaalseid parandusi kas arhitektuuris või funktsionaalsuses polnud vaja teha. Peamiselt olid kõik ebamugavused seotud arhiivi seadistamisega – klientide ja skeemide haldamisega. Kõigepealt parandati klientide õiguste haldamist. Sündmuseid saadetakse REST kasutajaliidese kaudu URLile –

https://<hostname>:<service.port>/api/v1/{database}/{schema}/{table}

Arhiivi algses versioonis hoiti kliendi kõigi õiguste andmeid klientide tabelis ühe stringina eraldi väljas. Kui õigustes oli nt ainult „api/v1/database1“, siis see tähendas, et klient võis saata andmeid kõikidesse selle andmebaasi skeemidesse. Selline lahendus oli mugav kui kliendil oli vähe õigusi ning nende uuendamine toimus harva. Kuid suure õiguste hulga puhul on see evamugav. Paranduseks eraldati õigused eraldi tabelisse ning iga õigust hoitakse tabelis eraldi reana. Täpsem kirjeldus on esitatud jaotises 4.5.3. Samuti lisati süsteemi dünaamiline skeemide ja tabelite loomine, kuna käsitsi loomine võttis aega. Selleks lisati skeemide ja tabelite loomise SQL skriptid, mida süsteem käivitas, kui saabus nõue uue skeemi loomiseks. Samuti lisati neid skripte käivitav moodul.

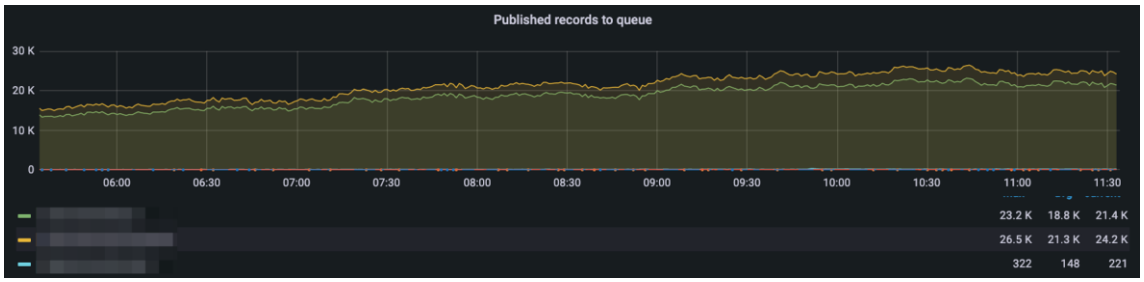
6 Testimine

Iga mikroteenuse jaoks on kirjutatud ühiktestid, mis katavad suurema osa koodist ning kogu funktsionaalsuse. Kokku kirjutati 68 ühiktesti.

- *Publisher* – 16
- *Manager* – 30
- *Loader* – 4
- *Job* – 18

Peale ühiktestide tegemist oli võimalus ka proovida valminud rakendust reaalses kasutuses. Selleks pandi rakendus valmis ühe kliendi poolt kasutamiseks. Järgnevad joonised esitavad testimise tulemusi.

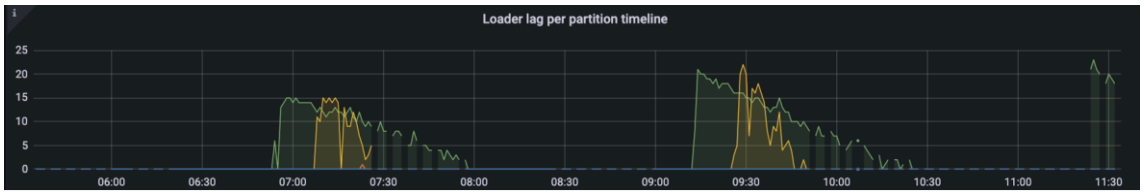
Joonis 47 esitab vastuvõetud ja Kafkasse saadetud sündmuste arvu. Diagrammil on esitatud kolm erinevad tüüpi sündmust (mille nimed on konfidentsiaalsuse huvides varjatud). Ajavahemikus 5:30–11:30 võttis arhiiv vastu ja pani Kafkasse keskmiselt umbes 40 000 sündmust minutis.



Joonis 47 Vastuvõetud sündmuste arv.

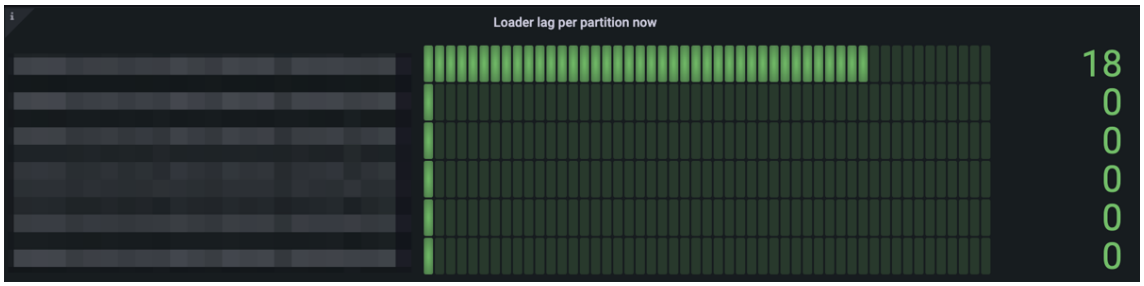
Joonis 48 näitab, miks Kafka kasutuselevõtt oli antud lahenduse jaoks hea otsus. Nagu on graafikult näha, siis *Loader* komponent ei suuda alati koheselt töödelda ja salvestada kõiki vastuvõetud sündmuseid ning sellest tekib sündmuste töötlemisel mahajäämus.

Kuid tänu Kafkale ei lähe ükski sündmus kaduma. Neid hoitakse Kafkas kuni *Loader* on jälle võimeline sündmuseid salvestama. Mahajäämuse põhjused võivad olla erinevad, nt andmebaasiga ühenduse kadumine või võrguühenduse kadumine. Graafikult on näha, et umbes kell 7:30 hakkasid kogunema sündmused, mida *Loader* ei jõudnud andmebaasi salvestada. Kuid juba kellaks 8:00ks oli see probleem lahendatud ja kõik sündmused olid salvestatud. Sama olukord kordus umbes kell 9:15 ja 11:20.



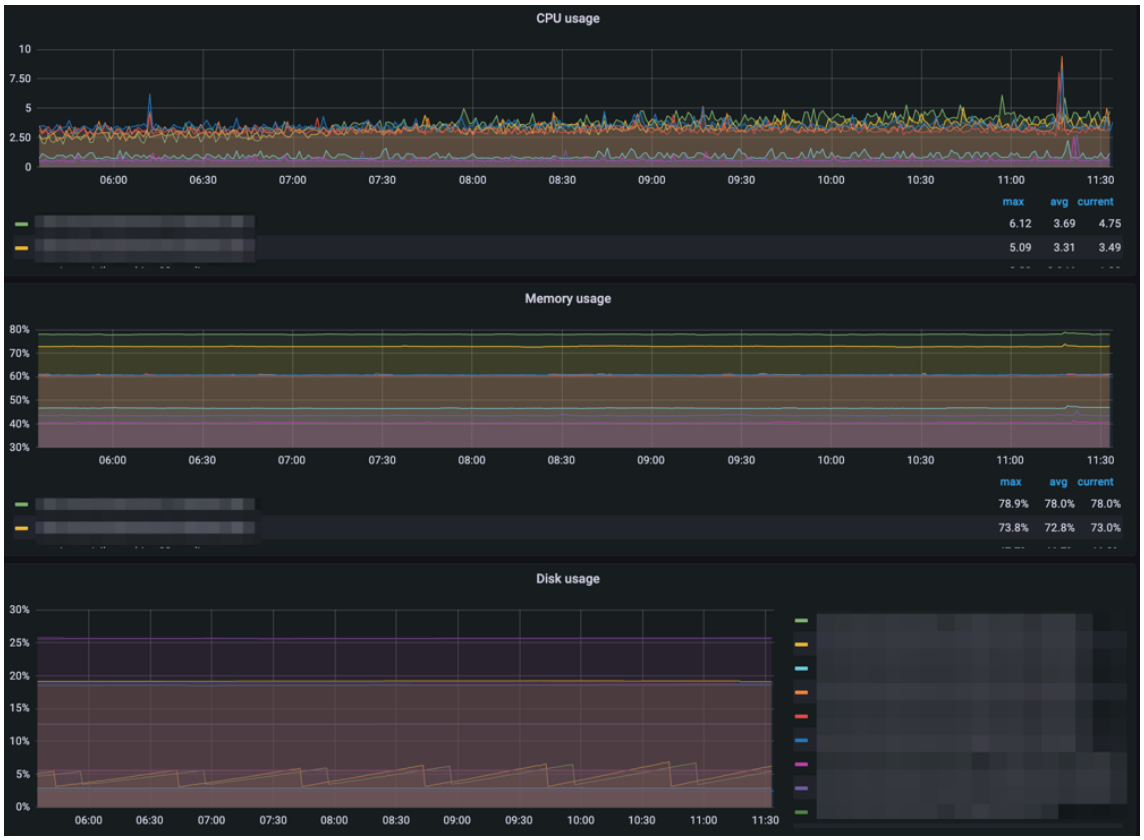
Joonis 48 Loader komponendi mahajäämuse illustratsioon.

Pildi tegemise ajal on mahajäämus ühe sündmuse tüübi (ehk sektsiooni) puhul 18 sündmust, mis on esitatud **Joonis 49**.



Joonis 49 Kafka järjekorras olevate sündmuste arv sektsioonis.

Joonis 50 esitab masina keskprotsessori, mälu ja ketta kasutamist antud ajavahemikus.



Joonis 50 CPU, mälu ja ketta kasutamine.

7 Analüüs ja järeldused

Töö tulemusena on loodud tarkvara, mis võimaldab võtta vastu JSONL formaadis andmeid ning salvestada ja hoida neid andmebaasis.

7.1 Töö tulemuste põhjendus

Antud jaotises põhjendatakse loodud tarkvara arhitektuuri ning kasutatud vahendite valikut.

7.1.1 Tükeldus allsüsteemideks

Analüüsi käigus jagati süsteem alamosadeks e allsüsteemideks. Allsüsteemide leidmisel lähtuti andmetest. Leiti süsteemi põhiobjektid e põhiolemitüübid. Igale põhiobjektile vastab süsteemi arhitektuuris eraldi funktsionaalne allsüsteem ja andmekeskne allsüsteem e register. Põhiobjektide kohta on süsteemis vaja säilitada andmeid ja igal põhobjektil on süsteemi seisukohast huvipakkuv elutsükkel e seisundid, milles selle objekti eksemplarid saavad viibida ja sündmused, mis põhjustavad seisundite üleminekuid. Iga funktsionaalne allsüsteem koondab konkreetse põhiobjektiga seotud andmete haldamise funktsionaalsust. Iga register koondab põhiobjektile vastavaid andmeid. Selline tükeldus tagab, et igal allsüsteemil on oma teema ja sellesse koondatud funktsionaalsus või andmed on kõik „ühe asja kohta“. Eesmärgiks oli eraldada funktsionaalsus võimalikult iseseisvateks osadeks ning süsteemi andmekeskne tükeldamine võimaldab seda teha [30]. Hiljem loodi iga funktsionaalse allsüsteemi jaoks vastav mikroteenus.

7.1.2 Kasutatud vahendid

Sobiva tehnoloogia valik on ülesande õnnestunud lahendamise jaoks kriitilise tähtsusega. Valitud tehnoloogiast sõltub projekti edukus. Antud jaotises kirjeldatakse kasutatud vahendeid ning põhjendatakse vahendite valikut.

7.1.2.1 Micronaut

Micronaut on JVM-põhine raamistik, mis on loodud Grails raamistikku arendajate poolt. Micronaut sobib hästi mikroteenuste rakenduste arendamiseks. Micronaut on alternatiiv Spring raamistikule, mis on 2022. aasta alguse seisuga väga populaarne. Micronaut pakub võrreldes Spring’iga kiiremat käivitusaega ning väiksemat mälukasutust.

Kuna Micronaut avaldati alles 2018. aastal, siis selle dokumentatsioon ei ole nii põhjalik nagu Spring raamistikul. Samuti leidub vähem õpetusi ja kasutusjuhendeid ning ka sobivaid näiteid on raskem leida. Vaatamata sellele on antud raamistikul oluliselt rohkem panustajaid võrreldes selliste raamistikega nagu näiteks POCO C++ [31] või Vert.x [32], mida näitab suur hulk Github *commite* ning suur hulk uute versioonide väljalaskeid [33]. See näitab, et arendajad kasutavad ja toetavad antud raamistikku.

Ka Spring raamistik toetab mikroteenuste arhitektuuri. Micronaut käivitub kiiremini kui Spring ja võtab sellest vähem muutmälu [34]. Kuna arhiivi rakendust plaaniti realiseerida kasutades mikroteenuste arhitektuuri ning Micronaut raamistik toetab seda, siis valiti süsteemi realiseerimiseks just see raamistik.

7.1.2.2 Kafka

Andmete vahetamise mikroteenuste vahel võib realiseerida kasutades erinevaid võimalusi, mida on kirjeldatud jaotises 3.4. Arhiivi rakenduses suhtlevad mikroteenused sõnumivahetuse kaudu, mis on realiseeritud Apache Kafka abil. Apache Kafka on avatud lähtekoodiga tarkvara. Latentsus tähendab aega sündmuste saatmisest salvestamiseni. Kafka eesmärgiks on pakkuda suurte andmemahtude korral võimalikult väikest latentsussaega ning suurt läbilaskevõimet. Kafka alternatiivideks on näiteks RabbitMQ [35] või Pulsar [36]. Kafka pakub suuremat läbilaskevõimet, väiksemat latentsust, suuremat töökindlust ja vastupidavust kui näiteks RabbitMQ [37].

Arvestades autori kogemuste ning leitud uuringutega valiti süsteemis kasutamiseks Apache Kafka.

7.1.2.3 PostgreSQL

Andmebaasisüsteemi valimisel peab 2022. aasta seisuga otsustama, kas kasutada SQL-andmebaasisüsteemi või NoSQL (Not Only SQL) andmebaasisüsteemi. SQL-andmebaasis on põhiliseks ehitusplokiks tabelid. NoSQL süsteemide puhul võib rääkida erinevatest andmemudelitest (võti-väärtus paarid, dokumendipõhine, veeruperekonnad, graafipõhine), mille variatsioone ja kombinatsioone andmebaasisüsteemid pakuvad.

PostgreSQL on avatud lähtekoodiga SQL-andmebaasisüsteem. Alates PostgreSQL 9.2 on süsteemis kasutusele võetud JSON andmetüüp, mis võimaldab PostgreSQL baasil

realiseerida hübriidse andmebaasi, kus osa andmeid saavad olla esitatud hierarhiliselt dokumentidena (JSON tüüpi väärtustena). JSON tüüpi veerus salvestatakse JSON dokument sellisena nagu see salvestamiseks esitati. Alates PostgreSQL 9.4 versioonist on süsteemis kasutusele võetud uus andmetüüp JSONB, mille korral JSON formaadis andmed sõelutakse ja salvestatakse kahendformaadis, mis veidi aeglustab andmete salvestamist kuid parandab olulisel määral salvestatud JSON väärtustega toimuvate operatsioonide jõudlust.

See muudab PostgreSQL-i sarnasemaks dokumendipõhiste NoSQL süsteemidega. JSONB andmete salvestamine on PostgreSQL andmebaasi puhul kiirem kui dokumendi andmete salvestamine NoSQL andmebaasi või MySQL andmebaasi puhul [38]. Samas võtab indeksite loomine PostgreSQL-s palju aega ning kui andmetele peab looma palju indekseid, siis lisamise operatsioon on aeglasem kui NoSQL süsteemide puhul. Kuna arhiivirakenduse andmetele ei ole plaanitud palju indekseid luua, siis tundub PostgreSQL antud lahenduse jaoks õige valik [38].

Antud töö jaoks valiti kasutamiseks PostgreSQL andmebaasisüsteem, sest selle kasutamine oli üheks nõudeks. Vaatamata ettekirjutusele on see autori arvates hea ja täiesti õigustatud valik. Lahenduse realiseerimisel ei tekkinud seoses andmebaasisüsteemiga mingeid raskuseid ja takistusi ning andmebaasi realiseerimine ja kasutamine oli mugav ning arusaadav.

7.1.3 Tehniline arhitektuur ja tehniline lahendus

Arhiivi rakendus on realiseeritud kasutades mikroteenuste arhitektuuri. Mikroteenuste arhitektuuril on palju eeliseid, mida kirjeldatakse jaotises 3.3. Kuna süsteem peab saama hakkama suure andmemahuga (minutis saabuvad mitmed tuhanded sündmused), siis on tähtis, et süsteemile saaks horisontaalse skaleerimise abil võimsust lisada. Mikroteenused toetavad horisontaalset skaleerimist ning seega sobib selline arhitektuuriline lahendus antud süsteemi jaoks. Lahenduse testimisel ning ka kasutamisel skaleeriti süsteemi lisades nii *Publisher* kui ka *Loader* mikroteenuste jaoks uusi sõlmi.

Samuti on tänu mikroteenuste arhitektuurile võimalik süsteemi lisada uusi mikroteenuseid, mille abil saab süsteemi laiendada ning lisada uut funktsionaalsust. Uute mikroteenuste lisamine ei nõua olemasolevas koodis muudatuste tegemist, sest kõik mikroteenused on iseseisvad komponendid.

Mikroteenuste vahel andmete vahetamiseks kasutatakse sõnumivahetust. Kuna andmete salvestamine andmebaasi võib olla aeglasem kui andmete vastuvõtmine, siis sobib sõnumivahetusel põhinev lahendus väga hästi. Sõnumivahetuse abil on võimalik edastada andmeid asünkroonselt ning andmeid vastuvõttev mikroteenus ei pea ootama kuni andmeid salvestatakse. Kui andmebaasi ühendusega on probleeme, siis andmete vastuvõtmine saab ikkagi jätkuda ning vastuvõetud andmeid hoitakse sõnumijärjekorras kuni ühendus andmebaasiga on taastatud.

7.1.4 Süsteemi hooldamise lihtsus

Üheks süsteemile seatud eesmärgiks oli selle hooldamise lihtsus. Kõige raskem ja aeganõudvam on arhiivi ettevalmistamine ja seadistamine (Kafka teemade tegemine, andmebaasi ettevalmistamine). Kui arhiiv läheb käima, siis peale klientide ja skeemide lisamist toimib arhiiv iseseisvalt ning ei nõua mingeid taaskäivitusi (*restart*). Uusi kliente ja skeeme on võimalik lisada REST liidese kaudu ning see ei vaja samuti süsteemi taaskäivitamist. Kõik skeemid ja tabelid luuakse samuti süsteemi poolt.

7.2 Võrdlus olemasolevate arhiivilahendustega

Jaotises 3.8.1 kirjeldati olemasolevaid arhiivilahendusi. Kõik need lahendused on realiseeritud kasutades erinevaid põhimõtteid ja tehnoloogiaid. Kuna digitaalsete andmete maht suureneb väga kiiresti, siis kõikide analüüsitud lahenduste üheks ja tähtsamaks nõudeks oli hakkama saamine suurte andmemahtudega. Samasugune eesmärk on ka antud töös kirjeldatud arhiivirakendusel, mis peab suutma võtta vastu tuhandeid sündmuste pakke minutis.

Kuna leitud lahenduste ja ettevõte X jaoks realiseeritud arhiivilahenduse kasutusalad on erinevad, siis on ka põhiliseks erinevuseks arhiveeritavate andmete formaat. Loodud arhiivirakendus võtab vastu JSONL formaadis andmeid, mis oli süsteemi üheks nõudeks. Leitud lahendused võtavad vastu erinevate tüüpi faile nagu meditsiinilised pildid, Word dokumendid ja PDF failid.

Arhiivirakenduse veel üheks nõudeks oli vastuvõetud sündmuste aheldamine ning andmebaasis säilitatavate sündmuste terviklikkuse kontroll. Andmete aheldamine aitab tuvastada andmetes tehtud muudatusi. Ükski leitud lahendustest sellist funktsionaalsust ei pakkunud.

Kõikide lahenduste sarnasuseks on see, et need on skaleeritavad. See tähendab, et need on realiseeritud nii, et koormuse suurenemisega on võimalik nendele lahendustele lisada juurde uusi sõlmi. Leitud lahendustest on see realiseeritud näiteks hajussüsteemide, mikroteenuste arhitektuuri või pilvetöötluse abil. Käesolevas töös esitletud arhiivirakenduses on see realiseeritud mikroteenuste arhitektuuri abil.

Leitud lahenduses, mis on mõeldud teaduslike digitaalsete andmete töötlemiseks [25], on kasutatud andmete hoidmiseks PostgreSQL andmebaasi nii nagu ka käesolevas arhiivirakenduses. Ülejäänud lahendused kasutavad andmete hoidmiseks NAS Synology lahendust ning Azure SQL andmebaasi.

7.3 Milliste teiste arhiivide loomiseks saab/ei saa seda lahendust kasutada?

Antud töös kirjeldatud süsteem loodi etteantud nõuete järgi ettevõtte X siseseks kasutamiseks. Loodud süsteem sobib selliste andmete hoidmiseks, mis on ainult JSONL formaadis.

7.4 Mida sellest arendusest õppida?

Töös esitatud süsteem on autori poolt valmis tehtud. Siis kui see süsteem sai valmis, võeti see kasutusele reaalsete klientide teenindamiseks. Tegelike andmetega ja suure koormuse all töötamine tõi esile mõned disainiga seotud probleemid.

Esimeseks probleemiks on ettenägematult suur andmemaht, mille tõttu oli vaja suurendada mälumahtu. Suure hulga JSON objektide mälus hoidmine võtab liiga palju muutmälu. Andmete maht sõltub teenuspakkujate ja sündmuste tüüpide arvust. Umbkaudu oli iga teenuspakkuja kohta kolm kuni kümme sündmuste tüüpi ning saabus 25 000 sündmuste pakki ühe sündmuse tüübi kohta minutis. Igas sündmuse pakkis oli maksimaalselt 1000 sündmust. Probleemi võimalikuks lahenduseks võiks olla andmebaasi struktuuri muutmine. Hetkel hoitakse vastuvõetud sündmuseid tabelis JSON dokumentidena JSONB tüüpi veergudes. Mälu säästmiseks peaks JSON väljade väärtuseid hoidma eraldi veergudes. Samas teeks selline muudatus süsteemi andmete salvestamise loogika keerulisemaks.

Teiseks probleemiks on sündmuste aheldamine ja skaleerimine. Probleem seisneb selles, et vastuvõetud andmeid aheldatakse sündmuste ning sündmuste tüüpide kaupa. Sellise lähenemise korral on tähtis jälgida vastuvõetud sündmuste pakkide järjekorda ning salvestada sündmuseid samas järjekorras nagu neid vastu võeti. Seepärast peavad kõik ühte tüüpi sündmused kindlasti olema ühes Kafka teemas. Selline loogika seab piiranguid skaleerimisele. See tähendab, et *Loader* komponendi skaleerimisel on maksimaalne sõlmede arv võrdne sündmuste tüüpide arvuga. Teiste sõnadega, iga sündmuste tüüpi peab teenindama ainult üks masin. Hetkel on selline olukord piisav, sest süsteem sai koormusega hakkama, kuid see on kindlasti probleem, mida peab võimalikult kiiresti lahendama.

Käesolevas töös pakutud andmete muutumatuse kontrolli lahendus kaitseb selle eest, kui mingis sündmuses muudetakse midagi või see sündmus kustutatakse. See võib juhtuda nii kogemata või meelega ja seda võib teha näiteks DevOps, andmebaasi administraator või arendaja. Käesolevas töös pakutud lahendus ei kaitse selle eest, et ründaja (sh pahatahtlik andmebaasi administraator, arendaja või DevOps) saab juurdepääsu andmebaasile, muudab andmeid ühes plokis ning siis arvutab iga järgneva ploki räsi uuesti ja salvestab andmebaasis. Praegu on lahenduseks kaitsta andmebaasi välispiiri, et keegi sellele loata ligi ei pääseks ning valida hoolega töötajaid, et nad ei kujutaks siseohtu.

7.5 Edasine arendus

Süsteemi seadistamine ja tööks ettevalmistamine nõuab päris palju tegevusi nagu klientide salvestamine ja salvestatud klientidele õiguste jagamine ning kasutamiseks lubatud andmebaaside ja skeemide lisamine. Hetkel peavad kõik pöördumised süsteemi pole toimuma REST päringute saatmise kaudu. Kasutajate mugavamaks teenindamiseks peab süsteemi lisama kasutajaliidese, mille kaudu oleks võimalik läbi viia kõiki eespool nimetatud tegevusi.

Iga süsteemi lisatud skeemi (teenusepakkuja) jaoks peab looma Kafka teema. Loodud teema nimi peab vastama mustrile ning teema peab olema ka õigesti konfigureeritud. See protsess võtab aega. Protsessi lihtsustamiseks võiks olla olemas eraldi teenus või süsteem koos kasutajaliidesega. Ettevõttes, mille jaoks arhiivilahendus realiseeriti, on selline süsteem olemas. Antud lahenduse iseseisvaks kasutamiseks oleks kasulik realiseerida see funktsionaalsus eraldi mikroteenusena.

Sündmuste andmed on vahepeal Kafkas. Seega peab uurima Kafka turvalisuse tagamise praktikaid ning realiseerima need antud rakenduses.

Samuti on ettevõttes realiseeritud ka süsteem, mis võimaldab salvestatud andmeid vaadata. Teiste sõnadega on neil olemas graafiline kasutajaliides, mille abil kasutaja saab vaadata salvestatud andmeid ja andmete terviklikkuse kontrolli tulemusi. Lahenduse iseseisvaks kasutamiseks võiks selline funktsionaalsus olla ka antud süsteemile lisatud.

Hetkel ei ole realiseeritud algoritmi ebaõnnestunud sündmuste töötlemiseks ning neid peab käsitsi üle vaatama. Süsteemis tuleks kavandada ja realiseerida automatiseeritud lahendus ebaõnnestunud sündmuste uuesti salvestamiseks.

Samuti on praeguses realisatsioonis tehtud nii, et kui sündmuste terviklikkuse kontrolli jooksul leitakse, et mingi sündmus on rikutud, siis selle kohta salvestatakse andmebaasi töö tulemus. Edaspidi tuleks realiseerida mingi lahendus kas sündmuse parandamiseks või vähemalt leitud rikkest teavitamiseks.

Lisaks uue funktsionaalsuse lisamisele peab lahendama ka need probleemid, mis toodi välja eelmises jaotises.

8 Kokkuvõte

Käesoleva töö eesmärgiks oli disainida ja realiseerida digitaalsete andmete arhiivilahendus ettevõtte X jaoks, mis vastaks kõikidele ettevõtte poolt esitatud nõuetele.

Töö oli realiseeritud kasutades disaini tegevusuuringut, mille eesmärgiks oli arendada tehnilist tehist ning peale esimese arhiivilahenduse versiooni valmimist võtta see kasutusele ja teha vastavalt kasutamise tulemustele jooksvalt parandusi.

Kõigepealt uuriti ja analüüsiti juba olemasolevaid arhiivilahendusi. Otsing näitas, et kuigi digitaalsete andmete arhiveerimiseks on olemas väga palju erinevaid lahendusi, siis puudub selline lahendus, mis vastaks täpselt kõikidele ettevõtte X poolt esitatud nõuetele. Töö esimeses pooles viidi läbi süsteemianalüüs. Kõigepealt jagati süsteem vastavalt süsteemi ülesannetele andmetest e põhiobjektides lähtuvalt võimalikult iseseisvateks allsüsteemideks e alamosadeks. Seejärel kirjutati lahti kõik süsteemi funktsionaalsed ja mittefunktsionaalsed nõuded. Funktsionaalsed nõuded kirjeldati funktsionaalsete allsüsteemide kaupa ning nende esitamiseks kasutati kasutusjuhtude mudelit. Nõuded andmebaasile esitati registrite kaupa kontseptuaalse andmemudelina. See mudel koosneb olemi-suhte diagrammidest ning sõnalistest kirjeldustest diagrammidel esitatud olemitüüpide ja atribuutide kohta.

Töö tulemusena loodi arhiivi rakendus, mis realiseeriti kasutades mikroteenuste arhitektuuri. Rakendus realiseeriti kasutades Micronaut raamistikku. Mikroteenuste suhtlemiseks kasutatakse asünkroonset teadete vahetamist, mis on realiseeritud Apache Kafka abil. Arhiveeritud sündmuseid koos metaandmetega salvestatakse ja hoitakse PostgreSQL andmebaasis.

Valminud rakenduse valideerimiseks kirjutati ühiktestid. Samuti valideeriti rakendust lokaalsel masinal ning hiljem testkeskkonnas autori ja testijate poolt. Peale valideerimist võeti rakendus reaalses keskkonnas kasutusele. Rakenduse käitumist jälgiti ning selle põhjal tehti rakenduses parandusi.

Antud töö lõpus analüüsiti töö tulemusi, põhjendati vahendite valikut ning rakenduse arhitektuuri ning võrreldi loodud rakendus olemasolevate lahendustega. Kuna lahendusest leiti nõrkuseid, siis analüüsiti neid nõrkusi ja pakuti nende nõrkuste

parandamiseks võimalikke lahendusi. Samuti tehti ettepanekud edasisteks arendustöödeks antud süsteemiga seoses.

Autor leiab, et töö alguses püstitatud eesmärk on saavutatud. Töö käigus valmis arhiivi rakendus, mida ettevõte X on juba hakanud kasutama.

9 Kasutatud kirjandus

[1] „Digital Preservation Handbook,“ [Võrgumaterjal]. Available: https://www.dpconline.org/handbook/glossary. [Kasutatud 25 04 2022].

[2] M. K. Sein, O. Henfridsson, S. Purao, M. Rossi ja R. Lindgren, „Action Design Research,“ *MIS Quart,* nr 35, pp. 37-56, 2011.

[3] N. Singh ja Z. Dawood, Building Microservices with Micronaut®: A quick-start guide to building high-performance reactive microservices for Java developers, Packt Publishing, 2021.

[4] M. van Steen ja A. S. Tanenbaum, Distributed Systems 3rd edition, CreateSpace Independent Publishing Platform, 2017.

[5] N. Medhat, „What are distributed systems?,“ [Võrgumaterjal]. Available: https://medium.com/nerd-for-tech/what-are-distributed-systems-75b93208aae0. [Kasutatud 06 05 2022].

[6] „Horizontal Scaling vs. Vertical Scaling: How to Choose Which is Right for You“.

[7] T. SOOME , „HAJUSSÜSTEEMID,“ [Võrgumaterjal]. Available: https://kodu.ut.ee/~mroos/hs/hs2.pdf. [Kasutatud 06 05 2022].

[8] M. Fowler, „Microservices,“ [Võrgumaterjal]. Available: https://martinfowler.com/articles/microservices.html. [Kasutatud 29 04 2022].

[9] A. Megargel, V. Shankararaman ja D. K. Walker, „Migrating from Monoliths to Cloud-Based Microservices: A Banking Industry Example,“ %1 *Software Engineering in the Era of Cloud Computing*, Research Collection School Of Information Systems, 2020, pp. 85-108.

[10] H. Mannaert, J. Verelst ja K. Ven, „Towards evolvable software architectures based on systemstheoretic stability,“ *SOFTWARE – PRACTICE AND EXPERIENCE,* nr 42, pp. 89-116, 2012.

[11] M. Kleppmann, Designing Data-Intensive Applications, O'Reilly Media, Inc., 2017.

[12] M. Fowler, „Microservice Trade-Offs,“ [Võrgumaterjal]. Available: https://martinfowler.com/articles/microservice-trade-offs.html. [Kasutatud 03 05 2022].

[13] G. Hohpe ja B. Woolf , Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, Addison-Wesley Professional, 2003.

[14] „Introduction to gRPC,“ [Võrgumaterjal]. Available: https://www.grpc.io/docs/what-is-grpc/introduction/. [Kasutatud 03 05 2022].

[15] „JSON Lines,“ [Võrgumaterjal]. Available: https://jsonlines.org/. [Kasutatud 08 05 2022].

[16] M. Hölbl, M. Kompara, A. Kamišalić ja L. N. Zlatolas, „A Systematic Review of the Use of Blockchain in Healthcare,“ *Symmetry,* nr 10, 2018.

[17] J. A. Berkowsky ja T. Hayajneh, „Security issues with certificate authorities,“ *IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON),* pp. 449-455, 2017.

[18] „Que sont les archives?,“ INTERNATIONAL COUNCIL ON ARCHIVES, [Võrgumaterjal]. Available: https://www.ica.org/fr/que-sont-les-archives. [Kasutatud 25 04 2022].

[19] A. Mkadmi, Archives in the Digital Age: Preservation and the Right to be Forgotten, Wiley-ISTE, 2021.

[20] „Google Scholar,“ [Võrgumaterjal]. Available: https://scholar.google.com/. [Kasutatud 08 05 2022].

[21] K. Shvachko, H. Kuang, S. Radia ja R. Chansler, „IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST),“ *ACM Computing Surveys*, pp. 1-10, 2010.

[22] A. Ergüzen ja M. Ünver , „Developing a File System Structure to Solve Healthy Big Data Storage and Archiving Problems Using a Distributed File System,“ *Applied Sciences,* nr 8, 2018.

[23] „Synology,“ [Võrgumaterjal]. Available: https://www.synology.com. [Kasutatud 08 05 2022].

[24] M. Prannoy , „Design and Implementation of an Archive Microservice solution for the Multi-AgentResearch and Simulation Distributed System,“ 2018.

[25] J. Jomier, S. R. Aylward, C. Marion, J. Lee ja M. Styner, „A digital archiving system and distributed server-side processing of large datasets,“ 2009.

[26] „DICOM#,“ [Võrgumaterjal]. Available: https://sourceforge.net/projects/dicom-cs/. [Kasutatud 08 05 2022].

[27] M. Chisholm, Managing Reference Data in Enterprise Databases: Binding Corporate Data to the Wider World, Morgan Kaufmann, 2000.

[28] „Liquibase,“ [Võrgumaterjal]. Available: https://www.liquibase.org/. [Kasutatud 03 05 2022].

[29] „PG Partition Manager,“ [Võrgumaterjal]. Available: https://github.com/pgpartman/pg_partman. [Kasutatud 25 04 2022].

[30] E. Eessaar, „On Applying Normalized Systems Theory to the Business Architectures of Information Systems,“ kd. 2, nr 3, pp. 132-149, 2014.

[31] „POCO C++ libraries,“ [Võrgumaterjal]. Available: https://pocoproject.org/. [Kasutatud 08 05 2022].

[32] „Eclipse Vert.x,“ [Võrgumaterjal]. Available: https://vertx.io/. [Kasutatud 08 05 2022].

[33] B. Mendes, N. Correia ja S. du Plessis, *A Comparative Study of Microservices Frameworks in IoT Deployments,* International Young Engineers Forum (YEF-ECE), 2021, pp. 89-91.

[34] „Micronaut,“ [Võrgumaterjal]. Available: https://micronaut.io/. [Kasutatud 10 05 2022].

[35] „RabbitMQ,“ [Võrgumaterjal]. Available: https://www.rabbitmq.com/. [Kasutatud 08 05 2022].

[36] „Apache Pulsar,“ [Võrgumaterjal]. Available: https://pulsar.apache.org/. [Kasutatud 08 05 2022].

[37] A. Bondarenko ja K. Zaytsev, „STUDYING SYSTEMS OF OPEN SOURCE MESSAGING,“ *Journal of Theoretical and Applied Information Technology,* nr 19, 2019.

[38] W. L. Schulz, B. G. Nelson, D. K. Felker, T. J. Durant ja R. Torres, „Evaluation of relational and NoSQL database architectures to manage genomic annotations,“ *Journal of Biomedical Informatics,* nr 64, pp. 288-295, 2016.

[39] A. R. Hevner, S. T. March, J. Park and S. Ram, "Design science in information systems research," *MIS Quarterly,* vol. 28, pp. 75-105, 2004.

[40] „Eclipse Vert.x,“ [Võrgumaterjal]. Available: https://vertx.io/. [Kasutatud 08 05 2022].

[41] „POCO C++ Libraries,“ [Võrgumaterjal]. Available: https://pocoproject.org/. [Kasutatud 08 05 2022].

[42] „JSON Lines,“ [Võrgumaterjal]. Available: https://jsonlines.org. [Kasutatud 08 05 2022].

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

Mina, Svetlana Ušakova

Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose " Digitaalse arhiivilahenduse kavandamine ja realiseerimine", mille juhendaja on Erki Eessaar

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.

03.05.2022

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.