TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Janne Jürimaa 213256IADB

Reaalajas teadete jälgimise lahendus teenuse kasutajatele

Bakalaureusetöö

Juhendaja: Meelis Antoi
Magistrikraad

Tallinn 2025

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: Janne Jürimaa

01.01.2025

Annotatsioon

Teadete edastamise teenustel on kaasaegses kommunikatsioonis oluline roll. Vaadeldavas organisatsioonis puudub teavitusteenuse kasutajatel kiire ja usaldusäärne ülevaade selle kohta, kas teade väljastati rakendusest edukalt või selle väljastamine ebaõnnestus.

Käesoleva lõputöö eesmärk on analüüsida erinevaid lahendusvõimalusi, mis võimaldaksid teavitusteenuse kasutajatel saada reaalajas teavet teadete väljastamise kohta. Samuti arendada lahendus, mis vastab seatud funktsionaalsetele ja mittefunktsionaalsetele nõuetele ning sobitub olemasolevate teenuste ja mikroteenuste arhitektuuriga.

Lõputöö keskendub analüüsile, milles kirjeldatakse ja võrreldakse tehnoloogiaid, mis sobivad lahenduse loomiseks. Eesmärk on leida lahendus, mida oleks tulevikus lihtne edasi arendada ja hallata ning mis sobiks kõige paremini olemasoleva süsteemiga. Analüüsi põhjal arendatud lahendus juurutatakse ja testitakse arenduskeskkonnas.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 27 leheküljel, 7 peatükki, 4 joonist, 1 tabelit.

Abstract

Real-Time Message Monitoring Solution for Service Users

Notification services play a crucial role in modern communication. In the observed organization, users of the notification service lack a quick and reliable overview of whether a notification was successfully delivered from the application or if the delivery failed.

The aim of this thesis is to analyze various solution options that would enable users of the notification service to receive real-time information about the delivery status of notifications. Additionally, to develop a solution that meets the set functional and non-functional requirements and integrates well with existing services and microservices architecture.

The thesis focuses on an analysis that describes and compares technologies suitable for creating the solution. The goal is to find a solution that can be easily developed and managed in the future and that fits best with the existing system. The solution developed based on the analysis will be deployed and tested in the development environment.

The thesis is in Estonian and contains 27 pages of text, 7 chapters, 4 figures, 1 table.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakendusliides, mis võimaldab erinevatel programmidel omavahel infot vahetada
AWS	*Amazon Web Services*, Amazoni veebiteenused
AWS CDK	*Amazon Web Services Cloud Development Kit*, raamistik serverivabade rakenduste loomiseks kasutades tuntud programmeerimiskeeli
AWS SAM	*Amazon Web Services Serverless Application Model*, raamistik serverivabade rakenduste loomiseks kasutades infrastruktuuri kui koodi
CI/CD	*Continuous Integration/Continuous Delivery*, pidev lõimimine ja pidev tarnimine – tarkvaraarenduse praktikad, mis automatiseerivad koodi lõimimise, testimise ja juurutamise erinevatesse keskkondadesse
DLQ	*Dead Letter Queue*, ebaõnnestunud sõnumite järjekord
FIFO	*First-In-First-Out*, andmete organiseerimise viis, mille puhul andmed väljastatakse samas järjekorras, nagu need lisati
IaC	*Infrastructure as Code*, infrastruktuur kui kood
ID	*Identification*, identifikaator
IDE	*Integrated Development Environment*, lõimitud arenduskeskkond
KiB	kibibait, 1024 baiti
JAR	*Java Archive*, Java arhiivifail
JSON	*JavaScript Object Notation,* andmevahetusvorming
POM	*Project Object Model*, projekti objektimudel, põhifail Mavenis, mis sisaldab teavet projekti kohta ja konfiguratsioonidetaile
SDK	*Software Development Kit,* tarkvaraarenduse komplekt
SMS	*Short Message Service*, lühisõnumiteenus, lühisõnum
SQS	*Simple Queue Service*, lihtne järjekorrateenus
URL	*Uniform Resource Locator*, ühtne ressursilokaator
ÜRO	Ühinenud Rahvaste Organisatsioon
XML	*Extensible Markup Language*, laiendatav märgistuskeel

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Kaasaegses kommunikatsioonis on teadete edastamise teenustel oluline roll, kuna need võimaldavad ettevõtetel kiiresti ja tõhusalt oma klientideni jõuda. Näiteks mobiiltelefonile saadetavad lühisõnumid ehk SMS-teated võimaldavad ettevõtetel teavitada oma kliente olulistest sündmustest, nagu saadetiste staatused, broneeritud teenuste meeldetuletused ja sooduspakkumised. Sõnumeid on võimalik ajastada ja saata klientidele korraga. Antud teenuste kõrge kvaliteet, sealhulgas põhjalik ja kiire teave saadetud teadete kohta on hädavajalik teenuse kasutajate rahulolu ja teavitusteenuse usaldusväärsuse tagamiseks [1].

Vaadeldavas organisatsioonis puudub teavitusteenuse kasutajatel täpne ja usaldusväärne ülevaade selle kohta, millised teated on teenusest väljapoole edukalt saadetud ja millised mitte.

Lõputöö eesmärk on lisada olemasolevasse süsteemi lahendus, mis pakuks teavitusteenuse kasutajatele võimalust jälgida nende poolt saadetud teadete kohta käivat usaldusväärset ja täpset infot reaalajas ning juurutada ja testida loodud lahendust arenduskeskkonnas.

Lõputöös käsitletakse SMS-teavituste olulisust ettevõtete suhtlusel oma klientidega. Kirjeldatakse süsteemiosa, kuhu loodav lahendus tuleb lõimida, määratakse funktsionaalsed ja mittefunktsionaalsed nõuded ning antakse ülevaade kasutajalugudest. Seejärel analüüsitakse erinevaid võimalusi ja tehnoloogiaid lahenduse loomiseks arvestades nõudeid, olemasolevat süsteemi ja klientide ootusi ning põhjendatakse valikuid. Töö tulemusena arendatakse lahendus, mis võimaldab reaalajas jälgida välja saadetud teadete kohta käivat põhjalikku infot ning juurutatakse ja testitakse seda arenduskeskkonnas.

2 Ülesande püstitus

Antud peatükis käsitletakse ettevõttes esinevat probleemi ja analüüsitakse selle olulisust. Järgnevalt määratletakse lõputöö eesmärgid ja ulatus ning lõpuks kirjeldatakse kasutatavat metoodikat.

2.1 Probleemi kirjeldus ja aktuaalsus

SMS-teavitused võimaldavad ettevõtetel tõhusalt suhelda oma klientidega ja omavad olulist rolli ettevõtete kasvus. ÜRO hinnangul kasvab maailma rahvaarv aastaks 2030 8,5 miljardini, mis toob kaasa ka mobiiltelefonide kasutajate arvu suurenemise [2]. Erinevate uuringute tulemustest selgub, et 98% mobiiltelefonide kasutajatest loevad neile saadetud SMS-teate läbi 90 sekundi jooksul pärast teate saamist ja vastavad sellele ligikaudu 2 korda suurema tõenäosusega kui telefonikõnele, e-kirjale või sotsiaalmeedia turundussõnumile, millest võib järeldada, et SMS-teavitused muutuvad üha olulisemaks vahendiks klientidega ühenduse võtmisel [3].

Uuritav ettevõte pakub teavitusteenust, mida kasutavad paljud teised ettevõtted oma klientidele SMS-teavituste saatmiseks. Kahjuks puudub teenuse kasutajatel võimalus saada usaldusväärset detailset infot klientidele saadetud lühisõnumite kohta. Neile on oluline teada, kas lühisõnumid saadeti teavitusteenusest edukalt edasi ja juhul, kui ei saadetud, siis milliseid ei saadetud.

Kuna see teave mõjutab oluliselt teavitusteenust kasutavate ettevõtete majanduslikku olukorda, on oluline saada see info võimalikult kiiresti pärast SMS-teavituste saatmist. Samuti on oluline, et selle teabe kättesaamine oleks garanteeritud. See võimaldaks teenust kasutavatel ettevõtetel reageerida viivitamatult ja vältida võimalikke kahjusid või probleeme, mis võivad tekkida teavituste mitte väljastamise tõttu.

2.2 Eesmärk

Käesoleva lõputöö eesmärgid:

- Analüüsida erinevaid lahendusvõimalusi, mis võimaldaksid ettevõtte teavitusteenuse kasutajatel saada põhjalikku, kiiret ja usaldusväärset teavet teenuse kaudu saadetud lühisõnumite väljastamise kohta.
- Arendada lahendus vastavalt läbiviidud analüüsile ja testida seda arenduskeskkonnas.

2.3 Metoodika

Käesolevas lõputöös uuritakse esmalt olemasolevat süsteemi, kuhu lahendus tuleb lõimida, seatakse funktsionaalsed ja mittefunktsionaalsed nõuded probleemi lahendamiseks ning kirjeldatakse kasutajalood. Seejärel analüüsitakse erinevaid lahendusi, mis võimaldavad suure hulga teadete kiiret ja usaldusväärset edastamist. Samuti analüüsitakse erinevaid tehnoloogiaid, mida lahenduse loomisel kasutada sobiks. Analüüsist lähtuvalt valitakse lahendusviis ja tehnoloogiad, mis vastavad kõige paremini nõuetele ja kliendi ootustele ning sobituvad olemasoleva süsteemiga. Valiku põhjendamiseks tuuakse esile valitud lahenduse eelised teiste võimaluste ees.

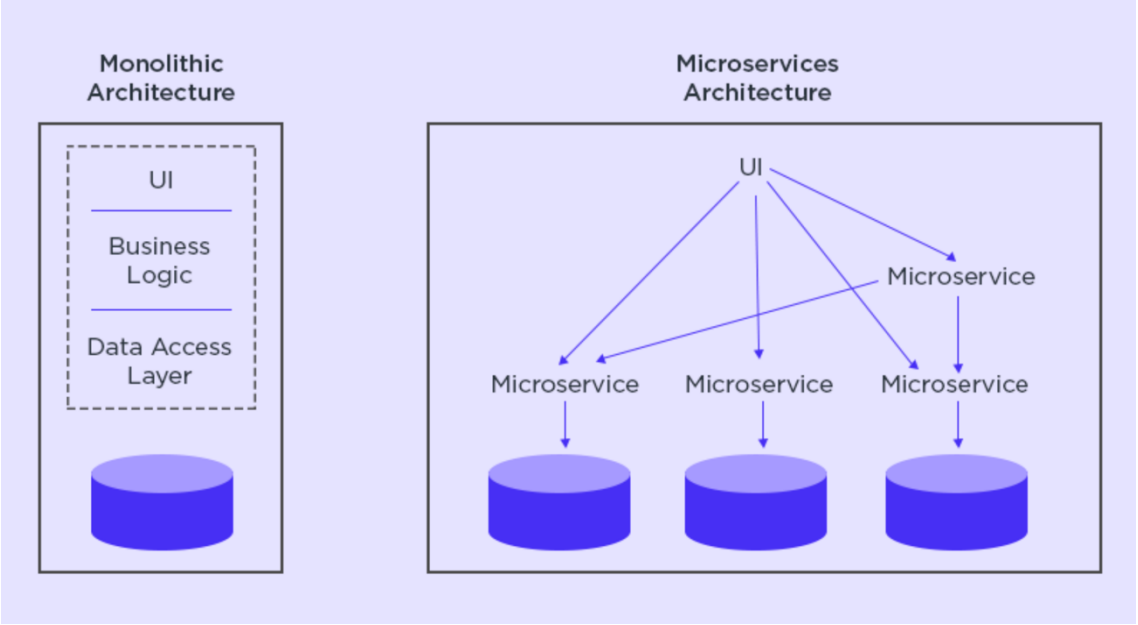
Järgmiseks etapis teostatakse valitud lahendus ja viiakse läbi testimine. Lõpuks analüüsitakse saadud tulemust ja kirjeldatakse edasisi arendusvõimalusi.

3 Olemasolev süsteem ja nõuded

Antud peatükis kirjeldatakse olemasolevat süsteemi, kuhu lahendus tuleb lõimida. Seejärel seatakse funktsionaalsed ja mittefunktsionaalsed nõuded loodavale lahendusele ning antakse ülevaade peamistest kasutajalugudest.

3.1 Olemasoleva süsteemi kirjeldus

Olemasolev süsteem on üles ehitatud mikroteenuste arhitektuurile. See on arhitektuuri stiil, mille puhul suur rakendus on jagatud väiksemateks, iseseisvalt töötavateks teenusteks, mis saavad omavahel suhelda. Erinevalt monoliitsest arhitektuurist, kus tarkvaraprogramm on üles ehitatud ühtse üksusena, täidab iga mikroteenus kindlat eesmärki ja on iseseisvalt juurutatav automaatsete juurutusmehhanismide abil. Joonisel 1 on illustreeritud ülesehituse erinevused. Monoliitse arhitektuuri puhul on kasutajaliides, äriloogika ja andmetele juurdepääsu kiht üks ühine rakendus, mis on seotud ühe andmebaasiga. Mikroteenuste arhitektuuri korral on kasutajaliides, teenused ja andmebaasid eraldatud, kuid omavahel ühendatud info vahetamiseks [4].



Joonis 1. Monoliitse ja mikroteenuste arhitektuuri ülesehitus [4].

Mikroteenuste arhitektuuri eeliseks monoliitse arhitektuuri ees on minimaalne haldus, võimalus arendada teenuseid erinevates programmeerimiskeeltes ja kasutada

mitmesuguseid andmesalvestustehnoloogiaid. Samas suurenevad arenduse keerukus, kulutused infrastruktuurile, organisatsiooniline koormus ning koodi silumine muutub keerulisemaks [4], [5].

Üks olemasolevatest ja töötavatest mikroteenustest on Teavitusteenus, mille API ehk rakendusliidese lõpp-punkti kasutades saavad teenuse kasutajad saata SMS-teavitusi oma klientidele [6].

Teine teenus on Infoteenus, mis sisaldab API lõpp-punkti, millele laekuvat infot on Teavitusteenuse kasutajatel võimalik tellida. Selles teenuses on info kuulajaks Apache Kafka (edaspidi Kafka) tarbija, mis haldab suhtlust Kafkaga ja töötleb sõnumeid vastavalt vajadusele. Kafka on voogedastusplatvorm, mis on loodud suurte andmemahtude töötlemiseks reaalajas [7].

Kuigi ka Infoteenus on klientidele kasutajaliidese kaudu kättesaadav, puudub hetkel usaldusväärne info saadetud teavituste kohta. Väärib märkimist, et olemasolev Kafka tarbija on rakendatud selliselt, et uue Kafka teema loomisel on hõlpsasti võimalik seadistada see tarbima sõnumeid, mis saabuvad sellesse uude Kafka teemasse. Need sõnumid edastatakse seejärel Infoteenuses olevasse API lõpp-punkti, mis omakorda muudab info Teavitusteenuse kasutajatele kättesaadavaks.

Kuna mikroteenuste arhitektuuri puhul on teenused üksteisest eraldatud ja probleemi lahendamiseks ei ole tarvis ülejäänud teenustega tutvuda, siis ei pea autor vajalikuks kirjeldada rohkemaid teenuseid uuritavas ettevõttes.

3.2 Funktsionaalsed nõuded

Funktsionaalsed nõuded kirjeldavad, milliseid ülesandeid süsteem peab täitma, et vastata kasutajate vajadustele ja ootustele. Need hõlmavad ka süsteemi toimimist tervikuna ja süsteemi eri osade omavahelist lõimimist [8].

Kuna on olemas teenus, mille kaudu saadetakse teavitusi, on mõistlik kasutada seda teenust, et koguda infot saadetud teavituste kohta. Samuti on otstarbekas pakkuda seda infot teenuse kasutajatele Infoteenuse kaudu, millel on juba olemas API lõpp-punkt teavituste kohta käiva info tellimiseks. Selleks, et saavutada maksimaalne efektiivsus,

tuleks kasutada olemasolevat Kafka tarbijat ja luua uus Kafka teema uue teavituste kohta käiva info jaoks.

Kogutav info peab olema piisavalt detailne, sisaldades vähemalt kliendi aadressi, kellele lühisõnum saadeti, lühisõnumi väljastamise aega, staatust ja sisu ning saatja identifikaatorit.

Mikroteenuste arhitektuuri põhimõtetele vastavalt ei ole sobilik lisada teavituse info ümberstruktureerimist ja rakendada Kafka tootjat olemasolevates teenustes. Seetõttu tuleks luua uus mikroteenus, mis vahendab teavituste kohta käivat infot.

Lähtuvalt eelnevast on loodavale lahendusele esitatud järgmised funktsionaalsed nõuded:

- Lahendus on lõimitud olemasoleva Teavitusteenuse ja Infoteenusega nii, et info teavituste kohta liigub Teavitusteenusest Infoteenusesse.
- Infoteenuses rakendatud Kafka tarbija töötleb teateid uuest Kafka teemast, mis on loodud spetsiaalselt uue teate tüübi jaoks.
- Saadetud teade sisaldab järgnevat lühisõnumi kohta käivat infot: saaja aadress, lühisõnumi väljastamise staatus, sisu, väljastamise aeg ja saatja identifikaator.
- Teate struktuuri muutmine toimub eraldi mikroteenuses.
- Kafka tootja on rakendatud eraldi mikroteenuses.

3.3 Mittefunktsionaalsed nõuded

Mittefunktsionaalsed nõuded keskenduvad süsteemi omadustele ja kvaliteedile, nagu jõudlus, skaleeritavus ehk omadus tulla toime suurenenud töökoormusega ning töökindlus, määratledes, kuidas süsteem peaks töötama või käituma [8].

Kuna teenuse tarbijaid on palju, peab süsteem olema võimeline saatma suures mahus teavet. Selleks, et teenuse kasutajad saaksid maksimaalset kasu, peab info kohale jõudma võimalikult kiiresti.

Loodavale lahendusele on järgmised mittefunktsionaalsed nõuded:

- Süsteem peab suutma saata mitu tuhat teadet sekundis.
- Teave peab liikuma Teavitusteenuse API lõpp-punktist Infoteenuse API lõpp-punkti võimalikult väikse viivitusega, eelistatult 5 sekundi jooksul.
- Teabe kohalejõudmine peab olema garanteeritud.

Peamised kasutajalood on määratletud Lisas 2. Kasutajalood on lühikesed kirjeldused tarkvara funktsionaalsusest, mis on kirjutatud kasutaja vaatenurgast ja mille eesmärk on selgitada, kuidas tarkvara funktsioon pakub kasutajale väärtust [9].

4 Lahendusvõimaluste analüüs

Selles peatükis kirjeldatakse ja analüüsitakse erinevaid meetodeid ja tehnoloogiaid reaalajas info edastamiseks ja infot vahendava mikroteenuse loomiseks. Lähtuvalt analüüsist tehakse sobivaimad valikud ning põhjendatakse neid.

4.1 Reaalajas teadete edastamine

Reaalajas andmed on teave, mis liigub lõppkasutajale või rakendusele kohe, kui see on saadaval, üldjuhul ilma märkimisväärse vahepealse töötlemise ja salvestamiseta. Reaalajas andmeid on kahte tüüpi – sündmuste andmed ja voogandmed. Esimesed kajastavad konkreetseid juhtumeid kindlal ajahetkel ja sisaldavad ajatemplit, näiteks krediitkaarditehing. Teiste puhul on tegu pidevalt uueneva andmevooga, mis ei pruugi olla ajatempliga, näiteks takso sõiduteekond [10].

Sündmuste andmete edastamiseks toimumise järjekorras kasutatakse sõnumijärjekorrateenuseid (inglise keeles *message queue service*) ja voogandmete edastamiseks voogedastusteenuseid (inglise keeles *streaming*). Erinevus seisneb selles, et sõnumijärjekorrateenused on loodud sõnumite usaldusväärse edastamise ja säilitamise eesmärgil, kuid voogedastusteenuste peamine eesmärk on kiire andmete töötlemine ja analüüs [11]. Kuna esitatud probleemi lahendamisel on oluline andmete kindel kohalejõudmine, siis on tarvis valida sobivaim teenus andmete edastamiseks sõnumijärjekorrateenuste hulgast.

4.2 Sõnumijärjekorrateenused

Enim kasutatavate sõnumijärjekorrateenuste hulka kuuluvad: Kafka, Amazon Kinesis (edaspidi Kinesis), RabbitMQ, Amazon Simple Queue Service (edaspidi SQS) ja Google Cloud Pub/Sub. Sobiva teenuse valimisel tuleb arvestada mitmete erinevate teguritega, mille alusel otsus langetada:

- Skaleeritavus – süsteem peab olema laiendatav või taandatav vastavalt vajadusele [12], [13].

- Sõnumite säilitamine – sõnumid peavad säilima sõnumite järjekorras teatud aja jooksul, et kindlustada nende edastamine teenuse tarbijale. Näiteks juhul, kui tarbijapoolne teenus on ajutiselt katkestatud ja sõnumeid ei õnnestu koheselt edastada, siis on võimalik nende edastamine tarbijapoolse teenuse taastamise järel.

- Madal latentsus – sõnumite kohaletoimetamine peab olema võimalikult kiire.

- Kõrge läbilaskevõime – teenus peab suure sõnumite hulga korral olema suuteline edastama sõnumid õigeaegselt.

- Sõnumite järjekord – sõnumid peavad jõudma kohale ajalises järjekorras, et info oleks teenuse tarbijatele kõige kasulikum.

- Ebaõnnestunud sõnumite järjekord ehk DLQ (inglise keeles *dead letter queue*) – sõnumijärjekorrateenus, mis salvestab ajutiselt sõnumid, mille edastamine ebaõnnestus, näiteks vigase sõnumi või vastuvõtja teenuses läbiviidud muudatuste tõttu. Ebaõnnestunud sõnumeid peab olema võimalik eraldada ja neid analüüsida, sest tänu sellele paraneb sõnumijärjekorrateenuse jõudlus ja vigade tuvastamine [12],[14].

Tabelis 1 on võrreldud eelnevalt loetletud sõnumijärjekorrateenuseid erinevate tegurite alusel [12],[15]–[18]. Kuigi erinevaid sõnumijärjekorrateenuseid on veel, siis on autor lihtsuse huvides juba teinud valiku sobivamate kasuks ja jätnud ülejäänud tabelist välja. Võrdluse tulemusel selgub, et RabbitMQ teenus pakub võrreldes teistega kehvemat läbilaskevõimet ja samuti pisut keerulisemat rakendamist ja jääb seega valikutest kõrvale [12],[18]. Google Cloud Pub/Sub pakub kõrget läbilaskevõimet ja seda on lihtne rakendada, kuid see ei suuda tagada sõnumite järjekorda ja seetõttu ei sobi parimaks lahenduseks [12],[16] . Kafka, Kinesis ja SQS pakuvad kõik madalat latentsust, kõrget läbilaskevõimet ja tagavad sõnumite järjekorra, kuid Kinesisel puudub otsene tugi DLQ jaoks ja Kafka puhul nõuab DLQ rakendamine arvestatavat arendustööd [12],[17],[18]. SQS-il on DLQ rakendamine toetatud ja selle lisamine on lihtne ja ei nõua erilist ajakulu. Kafka rakendamine ja haldamine on keeruline, Kinesise puhul on keerukus keskmine, samas kui SQS-i omandamine ja rakendamine on suhteliselt lihtne [12],[16],[18]. Seega on eelnevatest

variantidest parimaks valikuks SQS-i kasutamine, et Teavitusteenusest lühisõnumite kohta käiv info edasi saata.

Kuna Infoteenus on ettevalmistatud võtma informatsiooni vastu Kafka tarbija kaudu ja see osa ei kuulu muutmisele, tuleb uuest loodavast mikroteenusest info edastada Infoteenusesse kasutades Kafkat. See on samuti kriteeriumite poolest sobiv valik, kuid nõuab rohkem aega ja suuremaid kulutusi arendustööle [12].

Tabel 1. Sõnumijärjekorrateenuste võrdlus.

Tegur	**Kafka**	**Amazon Kinesis**	**SQS**	**RabbitMQ**	**Google Cloud Pub/Sub**
Skaleeritavus	Väga hästi skaleeritav	Väga hästi skaleeritav	Automaatne skaleerumine AWS[1]-i abil	Skaleeritav	Automaatne skaleerumine
Sõnumite säilitamine	Piirangud puuduvad, seadistatav	1 kuni 7 päeva, seadistatav	1 minut kuni 14 päeva, seadistatav	Piirangud puuduvad, seadistatav	10 minutit kuni 7 päeva, seadistatav
Latentsus	Madal	Madal	Madal	Seadistusest sõltuv	Madal
Läbilaskevõime	Kõrge	Kõrge	Kõrge	Keskmisest kõrgeni	Kõrge
Sõnumite järjekord	Tagatud, kui vastavalt seadistada	Tagatud, kui vastavalt seadistada	Tagatud	Tagatud	Ei ole tagatud
Ebaõnnestunud sõnumite järjekord	Vaja eraldi rakendada	Puudub	Toetatud	Toetatud	Toetatud
Õppimise ja rakendamise keerukus	Keeruline	Keskmine	Lihtne	Keskmine	Lihtne

[1] *Amazon Web Services*, Amazoni veebiteenused

4.2.1 Järjekordade loomine

SQS-i järjekordi võib luua kasutades käsurea käske, AWS-i konsooli või *Infrastructure as Code* ehk infrastruktuur kui kood (edaspidi IaC) tööriistu. IaC on meetod IT-infrastruktuuride haldamiseks käsitsi protsesside asemel koodi kaudu [41]. See võimaldab luua, seadistada ja hallata pilveressursse. IaC peamine eelis AWS-i konsooli kasutamise ees on infrastruktuuri haldamise lihtsustamine ja käsitsi tehtavatest protsessidest tulenevate vigade vältimine. Ühtlasi on IaC abil võimalik muudatusi kiiremini sisse viia kui konsooli kaudu [42].

SQS-i ressursside loomiseks ja haldamiseks saab kasutada erinevaid IaC tööriistu. AWS-i poolt pakutavad on AWS Serverless Application Model (edaspidi AWS SAM), AWS CloudFormation (edaspidi CloudFormation) ja AWS Cloud Development Kit (edaspidi AWS CDK). AWS CDK on populaarne valik, sest see on raamistik, mis võimaldab seadistada ressursse tuntud programmeerimiskeeltes nagu Python, Java, TypeScript, Go, JavaScript ja C#. CloudFormationi puhul saab valida YAML või JSON süntaksi kasutamise vahel. AWS SAM on CloudFormationi laiendus, mis on loodud spetsiaalselt serverivabade (inglise keeles *serverless*) rakenduste jaoks. See koosneb kahest osast: AWS SAM mallidest, mis juurutatakse otse CloudFormationisse ja AWS SAM käsurealiidesest, mis võimaldab kiiresti luua, arendada ja juurutada serverivabu rakendusi [42],[43].

AWS-i poolt pakutavatele lahendustele lisaks on laialdaselt kasutusel Terraform. Terraform on avatud lähtekoodiga projekt, mis pakub erakordset paindlikkust ja toetab kõiki peamisi pilveplatvorme: AWS-i, Google Cloudi ja Azure'i [44]. Kuigi sama koodibaasi ei saa otse ühelt pakkujalt teisele üle kanda, pakub Terraform järjepidevat süntaksit ja juurutustehnikaid erinevates pilvedes [45]. Antud ettevõtte puhul on see järjepidevus suureks eeliseks, mistõttu otsustati antud lahenduse puhul kasutada IaC tööriistana SQS-i järjekordade loomiseks ja haldamiseks Terraformi.

4.3 Infot vahendav mikroteenus

Kuna teavituse kohta käiva info muutmiseks ja edastamiseks on tarvis luua eraldi mikroteenus, siis järgnevalt analüüsitakse eelnevalt valitud sõnumijärjekorrateenustega sobivaid variante uue mikroteenuse loomiseks.

4.3.1 Programmeerimiskeel ja raamistik

Infot vahendavas mikroteenuses on vaja rakendada Kafka tootjat ja SQS-i tarbijat. Seega tuleb valida programmeerimiskeel, mis sobib nende mõlema jaoks. Erinevate artiklite põhjal on parim valik Kafkaga töötamiseks Java keel [19]–[21]. Teised populaarsemad valikud, mis sobivad hästi ka mikroteenuste loomiseks, on Python, Go, Node.js ja C# [4],[22],[23].

Java on laialdaselt kasutatav programmeerimiskeel ja üks parimatest valikutest, kuna see lõimub sujuvalt Kafka API-de ja teekidega, mis on algselt samuti kirjutatud Java keeles [23]. Kafka tootja rakendamiseks teavet otsides leidub kõige rohkem õpetusi just Java keeles. See näitab, et Java jaoks on olemas laialdane tugi ja palju dokumentatsiooni, mis on kasulik rakenduse arendamisel ja tõrkeotsingul. Sama ilmneb, kui otsida teavet SQS-i tarbija rakendamise kohta. Enamus õpetusi on tehtud Javat ja Spring Booti kasutades. Spring on Java raamistik, mis muudab Java rakenduste loomise ja käivitamise lihtsamaks. See vähendab konfiguratsiooni ja seadistamise keerukust, võimaldades arendajatel keskenduda rohkem rakenduse põhifunktsionaalsuste arendamisele. Spring Boot on kombinatsioon Spring raamistikust ja sisseehitatud serveritest ning pakub kiiremat viisi nii lihtsate kui ka veebipõhiste rakenduste seadistamiseks, konfigureerimiseks ja käitamiseks ning on laialdaselt kasutusel mikroteenuste loomisel [24].

Mikroteenuste arendamiseks on kasutusel veel teisigi Java raamistikke, näiteks Quarkus, Micronaut, Helidon, Chronicle Services, Vert.x, Kalix.io ja Dropwizard. Spring Boot koos Spring Cloudiga on neist parim valik, sest see on väljakujunenud ja testitud raamistik, millel on suur kogukond ja ulatuslik hulk olemasolevaid teeke, mida kasutada. Lisaks on saadaval palju ajakohast ja põhjalikku dokumentatsiooni ja õpetusi, mis muudavad arendustöö lihtsamaks ja kiiremaks. Spring Cloud pakub palju kasutusvalmis funktsioone, mis on vajalikud mikroteenuste arhitektuuris, näiteks koormuse tasakaalustamine. Spring Cloud AWS on osa Springi raamistikust ja on spetsiaalselt loodud selleks, et lihtsustada AWS-i ühildamist Springi rakendustega. Peale Dropwizardi on kõik mainitud raamistikud uuemad kui Spring Boot ja nende kohta on saadaval vähem õpetusi ja väiksem kogukonnatugi. Dropwizard ei ole samuti parim valik, kuna see on loodud peamiselt veebiteenuste arendamise lihtsustamiseks ja ka selle kohta on õpetusi vähem kui Spring Booti kohta, mis võib muuta arenduse ja tõrkeotsingu keerulisemaks [25],[26].

Kuna töö autoril on eelnevatest programmeerimiskeelte valikutest ühtlasi kõige rohkem kogemust Java keeles arendamisel ja samuti Spring Boot raamistiku kasutamisel, siis sobis see valik hästi uue mikroteenuse arendamiseks.

4.3.2 Raamistikusisesed valikud

Rakenduse arendamisel on oluline valida sobiv ehitustööriist (inglise keeles *build tool*), mis aitab lähtekoodi kompileerida, sõltuvusi hallata, teste käivitada, koodi pakendada ja juurutada. Arendaja saaks kõike seda ka ilma ehitustööriista kasutama teha, kuid tänu ehitustööriistale on võimalik neid protsesse automatiseerida ja tänu sellele arendustööd kiirendada [27].

Spring Boot raamistikku kasutades on peamised ehitustööriistad Maven ja Gradle. Maven põhineb Project Object Model (edaspidi POM) failil, mis on XML-formaadis ja sisaldab projekti ning kõigi seotud konfiguratsioonide üksikasju. See aitab leida õiged JAR-failid, pakub keskset hoidlat sõltuvuste jaoks ning lihtsustab projekti ehitamist ja juurutamist. Gradle kasutab Groovy'l või Kotlinil põhinevat domeenispetsiifilist keelt ehitusskriptide määratlemiseks ja pakub suurt paindlikkust ning kohandatavust. Maven on lihtsamini mõistetav ja kiiremini omandatav, kuid Gradle on arenduse ajal kiirem, sest kompileerimise käigus kompileeritakse ainult muudetud failid, mitte kõik failid nagu Maveni puhul. Samas on Mavenil laialdasem teekide olemasolu, sest see on kauem kasutusel olnud ja samuti suurem kasutajaskond. Antud ettevõttes on Maven ka eelistatum valik ja seetõttu otsustati antud projekti puhul kasutada ehitustööriistana Mavenit [28].

Spring Booti puhul on tavapärane kasutada atribuutide määratlemiseks välist konfiguratsiooni, sest see võimaldab kasutada sama rakenduse koodi erinevates keskkondades. Vaikimisi on need pärast projekti loomist Spring Boot Initializr'i abil määratud failis *application.properties*, mis kasutab võti ja väärtus vormingut. Iga rida on üks konfiguratsioon, mis tähendab, et hierarhiliste andmete väljendamiseks tuleb kasutada korduvalt samu eesliiteid. Teise variandina võib kasutada YAML-il põhinevat konfiguratsiooni. YAML on mugav vorming hierarhiliste konfiguratsiooniandmete määratlemiseks. Joonisel 2 on näha konfiguratsioon kasutades application.properties faili [29].

```
application.servers[0].ip=127.0.0.1
application.servers[0].path=/path1
application.servers[1].ip=127.0.0.2
application.servers[1].path=/path2
application.servers[2].ip=127.0.0.3
application.servers[2].path=/path3
```

Joonis 2. Konfiguratsiooni näidis application.properties failis.

Joonisel 3 on sama konfiguratsioon kasutades YAML-faili. YAML-i konfiguratsioon on paremini loetav, sest ei sisalda korduvaid eesliiteid [29].

```
application:
    servers:
    -   ip: '127.0.0.1'
        path: '/path1'
    -   ip: '127.0.0.2'
        path: '/path2'
    -   ip: '127.0.0.3'
        path: '/path3'
```

Joonis 3. Konfiguratsiooni näidis application.yml failis.

Lihtsama loetavuse huvides valiti atribuutide määratlemiseks loodavas Spring Boot rakenduses YAML-faili kasutamine.

4.3.3 Kafka skeemiformaat

Serialiseerimine on protsess, mille käigus andmestruktuurid või objektide olekud teisendatakse binaar- või tekstivormingusse, et neid saaks salvestada failidesse, andmebaasidesse või edastada võrgu kaudu. Kafka kontekstis tähendab serialiseerimine sõnumite teisendamist baitideks enne nende saatmist Kafka teemasse. Deserialiseerimine on vastupidine protsess, kus baitidena saadetud sõnumid muudetakse tagasi andmestruktuurideks või objektideks, mida rakendus saab kasutada [30].

Kafka edastab andmeid baitidena ilma valideerimiseta ning Kafka tootja ja tarbija ei suhtle otseselt, vaid Kafka teema kaudu. Selleks, et tarbijad teaksid, milliseid andmeid tootjad saadavad, kasutatakse Kafkaga koos skeemiregistrit (inglise keeles *Schema Registry*). See on tsentraliseeritud teenus, mis haldab ja salvestab Kafka kaudu vahetatavate andmete skeeme. Enne andmete saatmist Kafkale kontrollitakse, kas kasutatav skeem on skeemiregistris saadaval. Kui antud skeemi ei leita, siis salvestatakse see skeemiregistrisse. Serialiseerimise ajal kasutavad Kafka tootjad skeemiregistrit, et konkreetse skeemi alusel kodeerida andmeobjektid binaarvormingusse ja saata Kafkale

koos lisatud skeemi identifikaatoriga (edaspidi ID-ga). Sõnumite tarbimisel hangivad Kafka tarbijad sõnumi päisest esmalt skeemi ID, millele vastab skeemiregistris kordumatu skeem, mille järgi dekodeeritakse binaarne sisu tagasi andmeobjektideks. Skeemiregister võimaldab skeemide muutumist ajas, tagades ühilduvuse erinevate skeemiversioonide vahel. Toetatavad skeemiformaadid on Apache Avro (edaspidi Avro), JSON ja Protobuf [31].

JSON on kerge tekstipõhine andmevahetuse vorming, mida inimestel on lihtne lugeda ja kirjutada ning masinatel töödelda. Avro on binaarne serialiseerimisvorming, mis kasutab andmetüüpide ja protokollide määratlemiseks JSON vormingut ning serialiseerib andmed kompaktsesse binaarvormingusse. Avro võimaldab skeemide ühilduvust nii edasi- kui tagasipidi, mis lihtsustab serialiseeritud andmete muutuste haldamist arendajate jaoks. JSON ei toeta skeemi evolutsiooni vaikimisi ja sõnumisuurused on üldiselt suuremad kui Avro ja Protobufi puhul. Protobuf on Google'i loodud tõhus ja kiire meetod struktureeritud andmete serialiseerimiseks, kuid see nõuab spetsiifiliste *.proto* failide kompileerimist keelespetsiifiliseks koodiks, mis võib aeglustada arendustsüklit [30]. Protobufi serialiseeritud andmed on ainult binaarsed, mistõttu on Avroga võrreldes neid raskem kontrollida ja siluda [32]. Seetõttu otsustati antud projekti puhul kasutada Avro formaati skeemide jaoks.

4.3.4 Konteineritehnoloogia

Konteineriseerimine tähendab rakenduse ja selle kõigi sõltuvuste pakendamist ühte transporditavasse üksusesse. See muudab rakenduse testimise ja juurutamise kiiremaks ning tõhusamaks, sest tänu sellele on tagatud, et rakendus töötab ühtemoodi erinevates keskkondades. Konteinerid on nagu virtuaalmasinad, need on isoleeritud ja neil on piiratud juurdepääs süsteemiressurssidele. Kuid erinevalt virtuaalmasinatest ei vaja konteinerid virtuaalset riistvara ega operatsioonisüsteemi, tänu millele on konteineriseerimine lihtsam ja tõhusam meetod kui virtuaalmasinate kasutamine [33].

Konteineriseeritud mikroteenuseid on lihtne juurutada erinevatel platvormidel ja operatsioonisüsteemides, tänu millele kiireneb arendusprotsess. Mikroteenuste arhitektuuri puhul on oluline, et iga mikroteenust oleks võimalik juurutada ja skaleerida teistest mikroteenustest sõltumatult. Ühes mikroteenuses esinev viga ei tohiks mõjutada teisi mikroteenuseid. Tuntuim tehnoloogia konteinerite loomiseks on Docker [34],[35].

Veebilehekülje *6sense.com* andmetel jäävad teised konteineriseerimise tehnoloogiad Dockerist, mida kasutab ligi 83% turuosast, kasutatavuse poolest märkimisväärselt maha. Kusjuures teisena, ligi 11% turuosast, on arvestatud Google Kubernetes Engine'it, mis on platvorm konteinerite haldamiseks ja mida saab samuti kasutada koos Dockeriga. Kolmandana, 1 % turuosast, moodustab Linux Containers (edaspidi LXC), mis on eelkäijaks Dockerile [36]. Dockeri eelistena võib välja tuua aktiivse arendajate kogukonna olemasolu ja avaliku hoidla, kust saab leida valmis konteineri kujutisi, mis võimaldavad kiiret ja lihtsat juurutamist. [37]. Konteineri kujutis on väike eraldiseisev tarkvarapakett, mis sisaldab kõike, mis on rakenduse käivitamiseks vajalik. Käivitamise ajal moodustatakse konteineri kujutisest konteiner [38]. LXC-st väiksema turuosalusega tehnoloogiaid ei kaalutud ja antud projekti puhul otsustati kasutada konteinerite moodustamiseks Dockerit.

Dockerit kasutades on konteinerite loomiseks mitu viisi:

- Dockerfile'i ja Dockeri käskude abil – luua projektis dokument nimetusega Dockerfile, mis sisaldab juhiseid konteineri kujutise loomiseks. Kujutise loomise saab käivitada käsuga *docker build* ja konteineri saab luua käsuga *docker run*, mis käivitab rakenduse Dockeris [39].

- Docker Compose'i tööriista abil – eeldab samuti Dockerfile'i loomist, kuid käivitamiseks kasutatakse eraldi YAML-formaadis dokumenti nimega *docker-compose.yml*, mis sobib paremini mitme konteineri haldamiseks. Selle abil saab ühe käsuga käivitada või peatada mitmest teenusest koosnevat komplekti. Vastavad käsud on *docker-compose up --build* ja *docker-compose down* [39].

- BuildPacksi abil – need on tööriistad, mis automatiseerivad konteinerite loomise protsessi. Kuigi nende kasutamine lihtsustab konteinerite loomist, piirab see ühtlasi paindlikkust ja kohandamisvõimalusi [40].

Kuna Docker Compose'i kasutamine võimaldab hallata mitut konteinerit koos ja pakub täpsemat kontrolli konteinerite loomise ja käivitamise üle, mis on eriti oluline rakenduste puhul, mis nõuavad spetsiifilisi konfiguratsioone, siis otsustati kasutada seda lahendusviisi konteinerite loomiseks Dockeriga [39],[40].

4.4 Arenduskeskkond

Java keeles ja Spring Boot rakenduse arendamiseks on erinevaid lõimitud arenduskeskkondi ehk IDE-sid. Need on tarkvararakendused, mis pakuvad terviklikke arendustööriistu ja keskkonda koodi kirjutamiseks, silumiseks ja juurutamiseks. Populaarseimad valikud on IntelliJ IDEA, Eclipse ja NetBeans [46],[47].

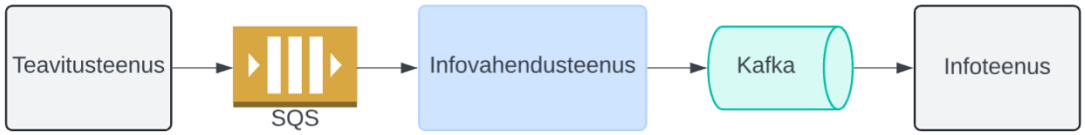
IntelliJ IDEA on väidetavalt alati olnud eespool Eclipse'ist ja NetBeansist koodi intelligentsuse, koodi täiendamise, vigade kontrollimise, ümberstruktureerimise ja nutikate soovituste osas. Eclipse ja NetBeans ei suuda jõudluse poolest võistelda IntelliJ IDEA poolt pakutava kohese reageerimise ja analüüsiga suurte koodibaaside puhul (Lisa 3). IntelliJ IDEA paistab silma oma intuitiivse ja esteetiliselt meeldiva disainiga, mis soodustab tõhusust ja sujuvat töövoogu. Lisaks sisaldab see mitmeid sisseehitatud tööriistu, näiteks versioonikontrolli, andmebaasitööriistu ja terminali. Eclipse'i ja NetBeansi puhul tuleb arendajal need pistikprogrammid ise leida ja installeerida, mis vähendab kasutajamugavust [47]. Kuigi IntelliJ IDEA Ultimate versioon on tasuline, siis otsustati kõiki eelnevaid eeliseid arvesse võttes kasutada projekti arendamiseks just IntelliJ IDEA-d ja selle Ultimate versiooni, mis pakub tuge Spring raamistikule. Tasuta versioon IntelliJ IDEA Community Edition seda ei paku [48].

5 Lahenduse arendus

Antud peatükis kirjeldatakse loodud lahenduse arhitektuuri ja arendust. Arenduseks kasutati lahendusviise ja tehnoloogiaid, mis lähtuvalt analüüsist peatükis 4 osutusid parimateks valikuteks, et lahendada peatükis 2.1 kirjeldatud probleem.

5.1 Arhitektuur

Loodi uus Infovahendusteenus, mis lõimiti olemasolevate mikroteenustega: Teavitusteenuse ja Infoteenusega. Info saatmiseks Teavitusteenusest Infoteenusesse kasutatakse SQS-i järjekorrateenust, mis saadab info Infovahendusteenusesse ja sealt edastatakse see Kafka kaudu Infoteenusesse. Olemasolevasse süsteemi lõimitud lahenduse arhitektuuri illustreerib Joonis 4.



Joonis 4. Lahenduse arhitektuur.

5.2 Infovahendusteenus

Esmalt loodi uus Spring Boot rakendus kasutades IntelliJ IDEA sisseehitatud funktsionaalsust Spring Boot rakenduse loomiseks. Uute projektide loomise nimekirjast valiti Spring Initializr, programmeerimiskeeleks Java versiooniga 17 ja ehitustööriistaks Maven (Lisa 4). Projekti jaoks loodi Git hoidla. Git on versioonihaldussüsteem, mis jälgib failidest tehtud muudatusi. Projekti koodi hoiustatakse GitHubis, mis on pilvepõhine platvorm koodi salvestamiseks, jagamiseks ja teistega ühiselt koodi kirjutamiseks [49].

Loodud moodul oli projekti juur-mooduliks ja selle sisse loodi eraldi moodul serveri jaoks (edaspidi “serverimoodul“). Serverimoodulis asuvasse *pom.xml* faili lisati järgnevad teegid:

- *lombok* – Java teek, mis kasutab annotatsioone koodi genereerimiseks kompileerimise ajal. See aitab vähendada standardkoodi (inglise keeles *boilerplate*), lihtsustab arendusprotsessi ja parandab koodi loetavust [50].

- *spring-boot-starter-web* – algkomponent, mis võimaldab luua veebirakendusi [50].

- *spring-boot-devtools* – tööriistade komplekt, mis kiirendab arendust, pakkudes näiteks automaatset taaskäivitamist koodimuudatuste korral [50].

- *spring-boot-starter-validation* – lisab valideerimisraamistiku Java ubade (inglise keeles *Bean*) valideerimiseks [50]. Java uba on Java klass, mis hoiab andmeid ja funktsioone järgides teatud reegleid: avalik vaikimisi konstruktor, privaatväljad avalike juurdepääsu- ja muutmismeetoditega (inglise keeles *getters and setters*) ning serialiseeritavus. Java ubade kasutamine muudab arenduse lihtsamaks ja suurendab koodi taaskasutust [51].

- *spring-boot-starter-actuator* – lisab funktsioonid, mis kontrollivad rakenduse olekut ja jälgivad selle toimimist [50].

- *spring-kafka* – lisab tööriistad, mis lihtsustavad Kafka lõimimist Springi rakendusega [52].

- *spring-cloud-starter-aws-messaging* – lisab AWS-i toe Spring projektile, mis võimaldab lihtsustatud lõimimist ja sõnumite saatmist/vastuvõtmist SQS-i kaudu [53].

Testimise jaoks vajalikud teegid on eelnevast loetelust välja jäetud, sest enamus neist lisati hiljem.

Vanem-moodulist kustutati ära *src* kaust, sealhulgas klass, millel oli annotatsioon *@SpringBootApplication*, kuna rakenduse käivitumiskohaks on servermoodulis asuv klass, millel on sama annotatsioon. *@SpringBootApplication* annotatsioon sisaldab endas kolme järgnevat annotatsiooni:

- *@EnableAutoConfiguration* – lubab Spring Booti automaatse seadistamise mehhanismi.

- *@ComponentScan* – lubab komponentide otsimise paketis, kus rakendus asub.

- *@SpringBootConfiguration* – võimaldab kontekstis täiendavate ubade registreerimist või täiendavate konfiguratsiooniklasside importimist [54].

See klass sisaldab *main* meetodit, mis käivitab rakenduse.

Serverimooduli sisse lisati Dockeri abil konteinerite loomiseks vajalikud failid: Dockerfile ja *entrypoint.sh*. Viimases failis seadistati rakenduse käivitamiseks vajalikud keskkonnamuutujad ja turvaseaded Kafkaga ühendamiseks. Juurmoodulisse lisati fail *docker-compose.yml*, kus määratleti erinevad teenused ja nende konfiguratsioonid, sealhulgas seadistused Kafka ja LocalStacki kasutamiseks.

LocalStack on raamistik, mis simuleerib AWS-i teenuseid lokaalselt ning lihtsustab testimist ja arendamist ilma tegelikke AWS-i ressursse kasutamata. Tänu sellele välditakse kulutusi, mis on seotud reaalsete AWS-i teenuste kasutamisega arenduse ja testimise käigus. Lisaks tagab LocalStacki kasutamine selle, et kõigi meeskonnaliikmete arvutites on ühtne arendus- ja testimiskeskkond [55].

5.2.1 Avro skeem ja Java klasside automatiseerimine

Seejärel loodi Avro skeem Kafka jaoks. Avro skeemifail võib sisaldada ainult üht skeemidefinitsiooni. Kasutatakse primitiivseid ja komplekseid andmetüüpe ning skeemifail peab sisaldama välju *type, name* ja *fields*. Neist viimane väli defineeritakse objektide kogumina, millest igaüks sisaldab välju *name* ja *type.* Erinevate primitiivsete andmetüüpide kasutust Avro skeemis illustreerib Lisas 5 olev näide [56].

Kuna käsitsi Avro skeemidest Java klasside loomine on aeganõudev ja vigadele vastuvõtlik protsess, siis järgnevalt automatiseeriti Kafkale edastatavate sõnumite jaoks vajalike Java klasside moodustamine Avro skeemi järgi. Selleks loodi eraldi alam-moodul (edaspidi avromoodul), mille *pom.xml* faili lisati sõltuvused *avro* ja *avro-compiler*. Seadistati *avro-maven-plugin* eesmärgiga *schema*, mis tähendab, et projekti ehituse käigus moodustatakse automaatselt Java klassid vastavalt Avro skeemifailides olevale infole. Selleks, et serverimoodulis neid Java klasse kasutada saaks, lisati serverimooduli *pom.xml* faili sõltuvuseks avromoodul [56].

5.2.2 Kommunikatsioon Kafka kaudu

Kafka kaudu sõnumite edastamiseks loodi Kafka teema ja lisati *application.yml* faili vajalikud seadistused Kafka jaoks. Peamised omadused, mida Kafka puhul määrata on:

- Kafka serveri aadressid (*application.yml* faili parameeter *bootstrap-servers*), mida nii tootjad kui ka tarbijad kasutavad sõnumite saatmiseks ja vastuvõtmiseks [57].

- Võtme ja väärtuse serialiseerijad/deserialiseerijad (parameetrid *key-serializer* ja *value-serializer*), sest Kafka nõuab, et sõnumid oleksid serialiseeritud. Tootjad serialiseerivad võtmed ja väärtused ning tarbijad deserialiseerivad need pärast vastuvõtmist [57]. Serialiseerimine on oluline sõnumite suuruse vähendamisel, muutes andmete edastamise efektiivsemaks. Serialiseeritud andmed võtavad ka vähem salvestusruumi ning andmete serialiseerimise ja deserialiseerimise kasutamine võimaldab ühilduvust erinevates keeltes ja platvormidel rakendatud tootjate ja tarbijate vahel [31].

- Grupi ID, mida kasutatakse Kafka tarbijate identifitseerimiseks grupi osana, kuid antud juhul seda ei määratud, sest Kafka tootja puhul ei ole see vajalik [57].

Seejärel loodi eraldi konfiguratsiooniklass, mis kasutab Kafka parameetrite väärtusi, mis on kirjas *application.yml* failis. Kui parameetreid on palju ja need asuvad *application.yml* failis ühe eesliite all, siis on parim viis nende lugemiseks kasutada *@ConfigurationProperties* annotatsiooni. Efektiivseks andmete hoiustamiseks valiti Java rekord (inglise keeles *record*). Rekord on Java klass, mis toimib läbipaistva andmekandjana ja kuna see sisaldab muutumatuid andmeid, siis sobib see ideaalselt konfiguratsiooniandmete hoidmiseks. Konfiguratsiooniklassis saab rekordis olevaid andmeid kasutada annotatsiooni *@EnableConfigurationProperties* abil [58].

Kafkale sõnumite saatmiseks saab kasutada Spring Bootis olemasolevat *KafkaTemplate*'i. Konfiguratsiooniklassis loodi 2 Springi uba:

- *ProducerFactory*, mis määrab strateegia Kafka tootja eksemplaride loomiseks [58]. Kafka parameetrite väärtuste saamiseks kasutati konfiguratsiooniandmete hoidmiseks loodud Java rekordit.

- *KafkaTemplate*, mille sisendiks on eelnevalt kirjeldatud *ProducerFactory*.

Kafka tootja kasutab *KafkaTemplate'*i ja selle *send* meetodit, et Kafkale sõnumeid edastada.

Infoteenusesse lisati konfiguratsioonifailid eelnevalt loodud Kafka teema ja sellega seotud skeemifaili haldamiseks, võimaldades teenuses juba olemasoleval Kafka tarbijal alustada uue teema kuulamist.

5.3 Infoedastus SQS-i kaudu

SQS on täielikult hallatav sõnumijärjekorrateenus mikroteenuste, hajussüsteemide ja serverivabade rakendustega lõimimiseks. See toimib kui sõnumivahendaja, mis tähendab, et üks komponent saab saata sõnumeid järjekorda ja teised komponendid saavad neid sõnumeid sellest järjekorrast samal ajal lugeda. Selline arhitektuur võimaldab asünkroonset suhtlust, sest tootjad lisavad sõnumeid järjekorda ja tarbijad töötlevad neid sobivas tempos [59],[60].

5.3.1 SQS-i järjekorrad

SQS toetab kahte tüüpi järjekordi:

- Standardjärjekord – vaikimisi järjekorraliik, mis toetab piiramatut arvu päringuid sekundis iga sõnumitoimingu kohta. Tagatud on sõnumite edastamine vähemalt üks kord, kuid võib juhtuda, et sama sõnum saadetakse mitu korda või vales järjekorras [60].

- FIFO (inglise keeles *First-In-First-Out*) järjekord – järjekorraliik, millel puhul sõnumid väljastatakse selles järjekorras, nagu need lisati ehk esimesena lisatud sõnum väljastatakse esimesena ja igat sõnumit saadetakse täpselt ühe korra [60].

Kuna sõnumite saatmise järjekord oli projekti puhul oluline, siis loodi FIFO järjekord sõnumite jaoks. Plaanide järgi kasutati uue järjekorra (edaspidi *sms-main-queue*) loomiseks Terraformi. Teavitusteenusesse lisati uus dokument faililaiendiga *.tf*, mis viitab Terraformi failile. SQS-i järjekorda saab Terraformi abil luua kasutades komponenti nimega *resource* (eesti keeles *allikas*) *"aws_sqs_queue"*. Ükski parameeter järjekorra loomisel ei ole kohustuslik, mis tähendab, et väärtuse puudumisel asendatakse see vaikimisi väärtusega. Lisas 7 on toodud näide SQS-i järjekorra loomiseks kasutades Terraformi [61]. Samas failis seadistati ka DLQ (edaspidi *sms-dlq*), et püüda sõnumeid, mida ei saa pärast korduvaid katseid edukalt töödelda. Need teisaldatakse *sms-main-queue*'st *sms-dlq*'sse [59].

Pärast muudatuste paigaldamist saab loodud järjekordi ja nende parameetreid vaadata, kui logida sisse AWS-i konsooli[1]. Samas keskkonnas saab testida sõnumi saatmist ja saabumist loodud järjekorda. Kui on tarvis teha muudatusi loodud järjekorras, siis ei ole soovitatav teha neid otse AWS-i konsoolis. Selle asemel tuleks teha muudatused loodud Terraformi failis, sest Terraform haldab ja jälgib infrastruktuuri. Otsene muudatuste tegemine AWS-i konsoolis võib põhjustada vastuolusid Terraformiga loodud ressursside soovitud oleku ja ressursside tegeliku oleku vahel [61].

5.3.2 SQS-i tootja ja tarbija

Teavitusteenuses loodi saadetava objekti jaoks sobivate parameetritega klassid, lisati vajalikud konfiguratsioonid ning loodi sõnumi saatmise teenus. SQS-i tootja saadab sõnumeid kasutades klassi *SqsClient*, mis on osa AWS SDK-st. AWS SDK on tarkvaraarenduse komplekt, mis lihtsustab AWS-i teenuste kasutamist, muutes automaatseks sellised toimingud nagu teenusepäringute krüptograafiline allkirjastamine, päringute kordamine ja veavastuste käsitlemine [62].

Infovahendusteenuses loodi SQS-i järjekorda saabuvate sõnumite lugemiseks SQS-i tarbija, mis võtab sõnumi vastu, töötleb selle struktuuri ja edastab samas teenuses oleva Kafka tootja kaudu Kafkale. Spring Bootis saab SQS-i tarbija luua annotatsiooniga *@SqsListener*, mis võtab argumendiks SQS-i järjekorra nime. Annotatsiooniga märgitud meetod konfigureeritakse Springi poolt automaatselt nii, et kui sõnum saadetakse määratud järjekorda, käivitub see meetod ja sõnumi sisu edastatakse meetodi parameetrina. Näide *@SqsListener*'i kasutamisest SQS-i tarbija loomiseks on Lisas 8 [59].

SQS-iga ühendamiseks lisati *application.yml* faili AWS-i piirkond, AWS-i võtmed, SQS-i lõpp-punkti URL[2], mille väärtus lokaalselt testides on *http://localhost:4566*, ning SQS-i järjekorra nimetus – *sms-main-queue*, mille loomist kirjeldati eelnevas punktis [63].

[1] https://aws.amazon.com/console/

[2] *Uniform Resource Locator*, ühtne ressursilokaator

Testimise eesmärgil lisati pom.xml faili sõltuvused *localstack* ja *junit-jupiter* (grupist *org.testcontainers*) ning *awaitility* (grupist *org.awaitility*). Viimane on kasulik asünkroonse sõnumite tarbimise kinnitamiseks [64].

6 Tulemused

Lõputöö tulemusteks on lahenduse analüüs ja selle põhjal arendatud lahendus reaalajas teadete saatmiseks Teavitusteenusest Infoteenusesse. Loodud lahendus on juurutatud ja testitud arenduskeskkonnas. Valminud lahenduses on arenduskeskkonnas teostatud kõik funktsionaalsed nõuded, mis on loetletud peatükis 3.2.

6.1 Nõuetele vastavus

Vastavalt nõuetele lõimiti lahendus olemasolevate teenustega. Teavitusteenuses rakendati SQS-i tootja ja Infoteenusesse lisati Kafka teema kuulamiseks vajalikud konfiguratsioonifailid ja fail skeemi haldamiseks. Lahenduse lõimimisel olemasolevate teenustega lisati väärtust juba toimivatele teenustele. Lisaks säilitati teenuse kasutajate jaoks kasutusmugavus, kuna uut teavet on võimalik tellida kasutajatele juba tuttavat API lõpp-punkti kasutades. Samuti hoiti kokku arendusele tehtavaid kulutusi, sest polnud tarvis luua juurde mitmeid uusi lisateenuseid, vaid piisas ühe uue teenuse loomisest, mis edastab infot Teavitusteenusest Infoteenusesse.

Infoteenusesse lisatud konfiguratsioonid võimaldavad olemasoleval Kafka tarbijal teavituste edastamiseks loodud Kafka teemat kuulata. Lisatud skeemifail võimaldab uue teate tüüpi sõnumeid Kafka tarbijal deserialiseerida. Tänu lisatud konfiguratsioonidele ei olnud tarvis luua uut Kafka tarbijat, mille arendusele oleks kulunud märkimisväärselt rohkem aega, ning paranes koodi taaskasutus.

Kafka tootja, mis saadab infot Kafka teemale, on rakendatud Infovahendusteenuses, mis on loodud spetsiaalselt antud lahenduse raames. Uue teenuse loomine võimaldas kasutada Spring Boot võimalusi ja lihtsustas Kafka tootja ja SQS-i tarbija arendamist tänu Spring Booti eelkonfigureeritud seadistustele ja sisseehitatud tööriistadele.

Infoteenusesse edastatav teade sisaldab nõutele vastavat informatsiooni sõnumi kohta, mis Teavitusteenust kasutades saadetakse. Teate struktureerimine toimub Infovahendusteenuses, kus SQS-i tarbija poolt saadetud infost eraldatakse vajalik ja lisatakse Kafka teemale edastamiseks nõutud andmed. Tänu sellele on andmetöötlemise loogika eraldatud teistest teenustest, mis vähendab süsteemi keerukust ja muudab süsteemi lihtsamini hooldatavaks. Ilma Infovahendusteenuse loomiseta oleks arendus

olnud aeglasem ja riskantsem, sest muudatuste tegemine ja uute funktsioonide loomine olemasolevates teenustes mõjutaks suuremat osa süsteemist. Lisaks piiraks selline arendus skaleeritavust, sest ühe teenuse koormuse suurenemine mõjutaks kogu süsteemi jõudlust [5].

6.2 Lahenduse lõpptestimine

Arenduskeskkonnas juurutatud lahendust testiti manuaalselt otsast lõpuni (inglise keeles *end-to-end*). Selleks kasutati Postmani, mis on tarkvararakendus API lõpp-punktide testimiseks, dokumenteerimiseks ja jagamiseks [65]. Postmani abil saadeti Teavitusteenuse API lõpp-punktist sõnum, mille saabumist Infoteenusesse kontrolliti kasutades terminalis avatud akent, mis oli seadistatud Infoteenuse API lõpp-punkti saabuvaid teateid kuulama. Sõnumeid saadeti ühele adressaadile korraga ja viidi läbi ligikaudu 50 saatmiskatset. Vastus saabus Infoteenuse API lõpp-punkti igal katsel ja saabumise aeg jäi autori objektiivsel hinnangul igal katsel paari sekundi piiresse, saabudes enamasti koheselt pärast Postmani kaudu päringu saatmist. Kuna peatükis 3.3 seatud mittefunktsionaalsete nõuete kohaselt peaks teave jõudma Teavitusteenusest Infoteenusesse 5 sekundi jooksul, ei peetud vajalikuks täpset saabumise aega mõõta, sest see ei ületanud ühelgi katsel paari sekundit.

Infoteenusesse kohale jõudnud info vastas peatükis 3.2 esitatud funktsionaalsele nõudele, sisaldades kõiki nõutud välju ja neile vastavaid väärtusi. Testiti nii edukalt kui ka mitteedukalt Teavitusteenuse kaudu väljastatud sõnumi kohta käiva info saabumist Infoteenusesse. Mõlemal juhul jõudis korrektne info Infoteenuse API lõpp-punkti.

Testimise tulemus näitas lisaks ka mittefunktsionaalse nõude täitmist, mille kohaselt peab teabe kohalejõudmine olema garanteeritud, sest kõigil testitud juhtudel jõudis teave Infoteenusesse.

Koormustestimist arenduskeskkonnas juurutatud lahendusega läbi ei viidud, sest testimise hetkel puudusid vastavad ressursid. Tulevikus on kindlasti soovitatav läbi viia koormustestimine, et tagada süsteemi töökindlus. Edukas koormustestimine näitaks, et kõik lahendusele seatud mittefunktsionaalsed nõuded on samuti täidetud.

7 Kokkuvõte

Käesoleva lõputöö eesmärkideks olid analüüsida erinevaid lahendusvõimalusi, mis võimaldaksid vaadeldava ettevõtte teavitusteenuse kasutajatel saada usaldusväärset ja kiiret teavet teenuse kaudu saadetud lühisõnumite väljastamise kohta. Lisaks arendada lahendus vastavalt analüüsile, juurutada ja testida seda arenduskeskkonnas.

Töös kirjeldatakse olemasolevat süsteemi, kuhu lahendus tuleb lõimida ja seatakse nõuded loodavale lahendusele. Seejärel analüüsitakse erinevaid lahendusviise ja tehnoloogiaid, mida probleemi lahendamiseks kasutada sobiks, ning põhjendatakse lahenduse jaoks valitud variante.

Analüüsi tulemusena loodi kahe olemasoleva teenuse vahele eraldi mikroteenus Spring Booti rakendusena, mis tegeleb info struktuuri muutmise ja vahendamisega. Teabe edastamiseks teavitusi väljastavast teenusest infot vahendavasse teenusesse kasutatakse AWS SQS-i ja sealt edastatakse see Kafka kaudu infot väljastavasse teenusesse.

Loodud lahendus juurutati ja testiti arenduskeskkonnas. Testimise tulemused näitasid, et lahendus vastab kõigile seatud nõuetele, välja arvatud nõue, mille täitmise kontrollimiseks on tarvis läbi viia koormustestimine. Ressursside puudumise tõttu jäi see antud töö raames sooritamata, kuid tulevikus tuleks see kindlasti läbi viia. Samuti tuleks lahendus juurutada ja testida testimiskeskkonnas, mis on võimalikult sarnane tootmiskeskkonnale. Eduka testimise järel saab lahenduse viia tootmiskeskkonda ja pakkuda klientidele usaldusväärset infot teavituste kohta.

Tulevikus on vajaduste muutumisel lihtne muuta olemasoleva teavituse struktuuri ja hallata SQS järjekordi. Seda tänu lahenduses rakendatud automatiseeritud klasside moodustamisele Avro skeemidest ja Terraformi kasutamisele järjekordade haldamisel.

Kasutatud kirjandus

[1] A. Stringfellow, „What Are SMS Notifications, and Why Are They Important?,“ MagicBell, 31. mai 2024. [Võrgumaterjal]. Loetud aadressil: https://www.magicbell.com/blog/what-are-sms-notifications-and-why-are-they-important/. [Kasutatud 13. september 2024].

[2] United Nations, „How certain are the United Nations global population projections?,“ detsember 2019, No. 2019/6.

[3] V. Magaline, „60 SMS Marketing Statistics That Will Change Your Mind About Text Message Marketing,” [Võrgumaterjal]. Loetud addressil: https://customers.ai/blog/sms-marketing-statistics/#17/. [Kasutatud 13. september 2024].

[4] V. Nadar, „5 Best Technologies To Build Microservices Architecture,“ Clarion Technologies, [Võrgumaterjal]. Loetud aadressil: https://www.clariontech.com/blog/5-best-technologies-to-build-microservices-architecture/. [Kasutatud 18. september 2024].

[5] C. Harris, „Microservices vs. monolithic architecture,” Atlassian, [Võrgumaterjal]. Loetud aadressil: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith/. [Kasutatud 19. september 2024].

[6] M. Goodwinn, „What is an API (application programming interface)?,“ IBM, 9. aprill 2024. [Võrgumaterjal]. Loetud aadressil: https://www.ibm.com/topics/api/. [Kasutatud 16. september 2024].

[7] A. Dearmer, „Amazon Kinesis vs. Kafka:A Detailed Comparison of Data Stream Services,” 24. detsember 2020. [Võrgumaterjal]. Loetud aadressil: https://www.integrate.io/blog/amazon-kinesis-vs-kafka/. [Kasutatud 23. september 2024].

[8] Grow Solutions, „Functional and Non-Functional Requirements: The Ultimate Checklist with Examples,” [Võrgumaterjal]. Loetud aadressil: https://medium.com/@growsolutions/functional-and-non-functional-requirements-the-ultimate-checklist-with-examples-cde16a/. [Kasutatud 16. september 2024].

[9] M. Rehkopf, „User stories with examples and a template,” [Võrgumaterjal]. Loetud aadressil: https://www.atlassian.com/agile/project-management/user-stories/. [Kasutatud 23. september 2024].

[10] W. To, „Real-Time Data: What it is, Why it Matters, and More,” 12. detsember 2023. [Võrgumaterjal]. Loetud aadressil: https://imply.io/blog/real-time-data-what-it-is-why-it-matters-and-more/. [Kasutatud 23. september 2024].

[11] K. Bhaduri, „Message Queue vs Streaming,” 19. veebruar 2023. [Võrgumaterjal]. Loetud aadressil: https://blog.iron.io/message-queue-vs-streaming/. [Kasutatud 24. september 2024].

[12] N. L, „Choosing The Right Message Queue Technology For Your Notification System,” 3. oktoober 2023. [Võrgumaterjal]. Loetud aadressil: https://dev.to/nikl/choosing-the-right-message-queue-technology-for-your-notification-system-2mji/. [Kasutatud 24. september 2024].

[13] A. Hayes, „Scalability: What a Scalable Company Is and Examples,” 23. juuli 2024. [Võrgumaterjal]. Loetud aadressil: https://www.investopedia.com/terms/s/scalability.as/. [Kasutatud 24. september 2024].

[14] AWS, „What is a Dead-Letter Queue (DLQ)?,” [Võrgumaterjal]. Loetud aadressil: https://aws.amazon.com/what-is/dead-letter-queue/. [Kasutatud 24. september 2024].

[15] A. Carr, „Comparing Apache Kafka, Amazon Kinesis, Microsoft Event Hubs and Google Pub/Sub,” 17. aprill 2018. [Võrgumaterjal]. Loetud aadressil: https://blog.scottlogic.com/2018/04/17/comparing-big-data-messaging.html/. [Kasutatud 25. september 2024].

[16] System Design School, „Understanding Pub/Sub vs Kafka," [Võrgumaterjal]. Loetud aadressil: https://systemdesignschool.io/blog/pub-sub-vs-kafka/. [Kasutatud 25. september 2024].
[17] Team Aspecto, „Kafka vs RabbitMQ vs AWS SNS/SQS: Which Broker to Choose?," 18. mai 2021. [Võrgumaterjal]. Loetud aadressil: https://www.aspecto.io/blog/kafka-vs-rabbitmq-vs-aws-sns-sqs-which-broker-to-choose/. [Kasutatud 25. september 2024].
[18] Habr, „Message broker selection cheat sheet: Kafka vs RabbitMQ vs Amazon SQS, " 10. veebruar 2023. [Võrgumaterjal]. Loetud aadressil: https://habr.com/en/articles/716182/. [Kasutatud 25. september 2024].
[19] V. Crudu & MoldStud Research Team, „What programming languages do Kafka developers need to know?," 19. august 2024. [Võrgumaterjal]. Loetud aadressil: https://moldstud.com/articles/p-what-programming-languages-do-kafka-developers-need-to-know/. [Kasutatud 26. september 2024].
[20] K. Pandey, „Best Practices for Integrating Apache Kafka with Java: A Comprehensive Guide," 24. juuni 2024. [Võrgumaterjal]. Loetud aadressil: https://www.linkedin.com/pulse/best-practices-integrating-apache-kafka-java-guide-kailash-pandey-qi8gc/. [Kasutatud 26. september 2024].
[21] Confluent, „Programming Languages for Apache Kafka: Essential Resources for Developers," 17. jaanuar 2024. [Võrgumaterjal]. Loetud aadressil: https://www.confluent.io/blog/programming-languages-for-apache-kafka-essential-resources-for-developers/. [Kasutatud 26. september 2024].
[22] S. Shrivastava, „5 Most Popular Programming Languages For Apache Kafka Cluster," 2. november 2021. [Võrgumaterjal]. Loetud aadressil: https://www.ksolves.com/blog/big-data/apache-kafka/the-top-5-programming-languages-for-the-apache-kafka-cluster/. [Kasutatud 26. september 2024].
[23] B. Reselman, „A developer's guide to using Kafka with Java, Part 1," 5. aprill 2022. [Võrgumaterjal]. Loetud aadressil: https://developers.redhat.com/articles/2022/04/05/developers-guide-using-kafka-java-part-1#/. [Kasutatud 26. september 2024].
[24] GeeksforGeeks, „Spring Boot Tutorial," 8. august 2024. [Võrgumaterjal]. Loetud aadressil: https://www.geeksforgeeks.org/spring-boot/. [Kasutatud 29. september 2024].
[25] R. Austin, „Top 7 Java Microservices Frameworks," 29. mai 2024. [Võrgumaterjal]. Loetud aadressil: https://foojay.io/today/top-7-java-microservices-frameworks/. [Kasutatud 30. september 2024].
[26] Medium, „Top 5 Java Microservices Frameworks to Learn in 2024," 7. jaanuar 2022. [Võrgumaterjal]. Loetud aadressil: https://medium.com/javarevisited/top-5-frameworks-java-developers-can-learn-for-microservices-development-in-2022-848da66d6651/. [Kasutatud 30. september 2024].
[27] T. Gregory, „Maven vs. Gradle in-depth comparison," 17. detsember 2021. [Võrgumaterjal]. Loetud aadressil: https://tomgregory.com/gradle/maven-vs-gradle-comparison/. [Kasutatud 30. september 2024].
[28] B V. Shree, „Maven vs Gradle.," 24. aprill 2023. [Võrgumaterjal]. Loetud aadressil: https://medium.com/@varshaashree6/maven-vs-gradle-416c506f90cd/. [Kasutatud 30. september 2024].
[29] Baeldung, „Using application.yml vs application.properties in Spring Boot," 11. mai 2024. [Võrgumaterjal]. Loetud aadressil: https://www.baeldung.com/spring-boot-yaml-vs-properties/. [Kasutatud 30. september 2024].
[30] A. C. Codex, „Kafka Message Serialization Techniques: Avro vs. JSON vs. Protobuf," 20. juuli 2024. [Võrgumaterjal]. Loetud aadressil: https://reintech.io/blog/kafka-message-serialization-avro-json-protobuf/. [Kasutatud 1. oktoober 2024].
[31] Medium, „Mastering Serialization and Deserialization in Kafka Streams: Building Efficient Data Pipelines," 29. märts 2024. [Võrgumaterjal]. Loetud aadressil: https://dip-mazumder.medium.com/mastering-serialization-and-deserialization-in-kafka-streams-building-efficient-data-pipelines-ff04154e374d/. [Kasutatud 1. oktoober 2024].

[32] D. Silber, „Comparing Avro vs Protobuf for Data Serialization,” 5. jaanuar 2024. [Võrgumaterjal]. Loetud aadressil: https://medium.com/streamdal/comparing-avro-vs-protobuf-for-data-serialization-2b97d419e525/. [Kasutatud 1. oktoober 2024].
[33] R. Powell, „Benefits of containerization,” 3. mai 2024. [Võrgumaterjal]. Loetud aadressil: https://circleci.com/blog/benefits-of-containerization/. [Kasutatud 2. oktoober 2024].
[34] H. Jeremy, „What Are Containerized Microservices?,” 12. juuli 2023. [Võrgumaterjal]. Loetud aadressil: https://blog.dreamfactory.com/what-are-containerized-microservices#1/. [Kasutatud 2. oktoober 2024].
[35] AviNetworks, „Microservices and Containers,” [Võrgumaterjal]. Loetud aadressil: https://avinetworks.com/what-are-microservices-and-containers/. [Kasutatud 2. oktoober 2024].
[36] 6sense, „Market Share of Docker,” [Võrgumaterjal]. Loetud aadressil: https://6sense.com/tech/containerization/docker-market-share/. [Kasutatud 2. oktoober 2024].
[37] P. G. Miguel, „Guide To The 20 Best Containerization Software Of 2024,” 1. oktoober 2024. [Võrgumaterjal]. Loetud aadressil: https://thectoclub.com/tools/best-containerization-software/. [Kasutatud 2. oktoober 2024].
[38] Docker, „Use containers to Build, Share and Run your applications,” [Võrgumaterjal]. Loetud aadressil: https://www.docker.com/resources/what-container/. [Kasutatud 3. oktoober 2024].
[39] Baeldung, „Dockerizing a Spring Boot Application,” 8. jaanuar 2024. [Võrgumaterjal]. Loetud aadressil: https://www.baeldung.com/dockerizing-spring-boot-application/. [Kasutatud 3. oktoober 2024].
[40] B. Chastanier, „Why You Should Use Docker Over Buildpacks,” 22. aprill 2023. [Võrgumaterjal]. Loetud aadressil: https://www.qovery.com/blog/why-you-should-use-docker-over-buildpacks/. [Kasutatud 3. oktoober 2024].
[41] Successive Technologies LLC, „Advantage of Infrastructure as Code With Terraform in Cloud & DevOps,” [Võrgumaterjal]. Loetud aadressil: https://successive.cloud/advantage-infrastructure-as-code-terraform-cloud-devops/. [Kasutatud 4. oktoober 2024].
[42] P. Vogel, „Implementing AWS Well-Architected best practices for Amazon SQS – Part 1,” 27. juuni 2023. [Võrgumaterjal]. Loetud aadressil: https://aws.amazon.com/blogs/compute/implementing-aws-well-architected-best-practices-for-amazon-sqs-part-1/. [Kasutatud 4. oktoober 2024].
[43] AWS, „AWS Serverless Application Model,” [Võrgumaterjal]. Loetud aadressil: https://aws.amazon.com/serverless/sam/. [Kasutatud 4. oktoober 2024].
[44] F. Pialoux, „Best Infrastructure as Code Tools (IaC): The Top 15 for 2024,” [Võrgumaterjal]. Loetud aadressil: best/. [Kasutatud 4. oktoober 2024].
[45] S. Gabrail, „Terraform vs AWS CloudFormation: An In-Depth Comparison,” 5. veebruar 2024. [Võrgumaterjal]. Loetud aadressil: https://www.env0.com/blog/terraform-vs-aws-cloudformation-an-in-depth-comparison/. [Kasutatud 4. oktoober 2024].
[46] AppsRhino, „The Best Tools for Successful Java Spring Boot Development,” 31. august 2024. [Võrgumaterjal]. Loetud aadressil: https://www.appsrhino.com/blogs/the-best-tools-for-successful-java-spring-boot-development/. [Kasutatud 4. oktoober 2024].
[47] MK Usmaan, „IntelliJ vs Eclipse vs NetBeans in 2024: Which Java IDE is Best?,” 14. august 2024. [Võrgumaterjal]. Loetud aadressil: https://techlasi.com/savvy/intellij-vs-eclipse-vs-netbeans/. [Kasutatud 4. oktoober 2024].
[48] JetBrains, „IntelliJ IDEA Ultimate vs IntelliJ IDEA Community Edition,” [Võrgumaterjal]. Loetud aadressil: https://www.jetbrains.com/products/compare/?product=idea&product=idea-ce/. [Kasutatud 4. oktoober 2024].
[49] GitHub Docs, „About GitHub and Git,” [Võrgumaterjal]. Loetud aadressil: https://docs.github.com/en/get-started/start-your-journey/about-github-and-git/. [Kasutatud 7. oktoober 2024].

[50] P. Naveen „Important Spring Dependencies," 3. juuni 2024. [Võrgumaterjal]. Loetud aadressil: https://www.linkedin.com/pulse/important-spring-dependencies-naveen-pn--rj7fc/. [Kasutatud 8. oktoober 2024].
[51] M.G. Memon, „Understanding Java Beans: A Comprehensive Guide for Beginners," 4. juuli 2023. [Võrgumaterjal]. Loetud aadressil: https://medium.com/@mgm06bm/understanding-java-beans-a-comprehensive-guide-for-beginners-684163011c82/. [Kasutatud 8. oktoober 2024].
[52] Baeldung, „Intro to Apache Kafka with Spring," 8. jaanuar 2024. [Võrgumaterjal]. Loetud aadressil: https://www.baeldung.com/spring-kafka/. [Kasutatud 8. oktoober 2024].
[53] Spring, „Spring Cloud AWS," [Võrgumaterjal]. Loetud aadressil: https://docs.spring.io/spring-cloud-aws/docs/2.2.3.RELEASE/reference/html/. [Kasutatud 8. oktoober 2024].
[54] Spring, „Using the @SpringBootApplication Annotation," [Võrgumaterjal]. Loetud aadressil: https://docs.spring.io/spring-boot/reference/using/using-the-springbootapplication-annotation.html/. [Kasutatud 8. oktoober 2024].
[55] V. D. Sciullo, „LocalStack: Simulating AWS Services for Local Development," 26. märts 2024. [Võrgumaterjal]. Loetud aadressil: https://medium.com/@disciullovincenzo/localstack-simulating-aws-services-for-local-development-589bad0700d1/. [Kasutatud 10. oktoober 2024].
[56] A. Maré, „Data Serialisation and Deserialisation using Apache Avro and Java," 10. märts 2024. [Võrgumaterjal]. Loetud aadressil: https://www.beyondengineering.io/data-serialisation-and-deserialisation-using-apache-avro-and-java-8d8e4db11283/. [Kasutatud 11. oktoober 2024].
[57] GeeksforGeeks, „Spring Boot - Integration with Kafka," 6. september 2024. [Võrgumaterjal]. Loetud aadressil: https://www.geeksforgeeks.org/spring-boot-integration-with-kafka/. [Kasutatud 14. oktoober 2024].
[58] Baeldung, „Guide to @ConfigurationProperties in Spring Boot," 8. jaanuar 2024. [Võrgumaterjal]. Loetud aadressil: https://www.baeldung.com/configuration-properties-in-spring-boot/. [Kasutatud 14. oktoober 2024].
[59] L. Gupte, „AWS SQS with Spring Cloud AWS and Spring Boot 3," 17. september 2023. [Võrgumaterjal]. Loetud aadressil: https://howtodoinjava.com/spring-cloud/aws-sqs-with-spring-cloud-aws/. [Kasutatud 15. oktoober 2024].
[60] M. Çakmak, „AWS Standard SQS Queue with Spring Boot," 14. september 2024. [Võrgumaterjal]. Loetud aadressil: https://medium.com/@mertcakmak2/aws-standard-sqs-queue-with-spring-boot-974c163e0616/. [Kasutatud 15. oktoober 2024].
[61] S. Sharmila, „Creating an SQS Queue Using Terraform: A Step-by-Step Guide," 5. juuli 2023. [Võrgumaterjal]. Loetud aadressil: https://sharmilas.medium.com/creating-an-sqs-queue-using-terraform-a-step-by-step-guide-54f1005dc616/. [Kasutatud 15. oktoober 2024].
[62] Amazon Web Services, „SqsClient, AWS SDK for Java - 2.10.61 API," [Võrgumaterjal]. Loetud aadressil: https://www.javadoc.io/doc/software.amazon.awssdk/sqs/2.10.61/software/amazon/awssdk/services/sqs/SqsClient.html. [Kasutatud 17. oktoober 2024].
[63] A. Dekavadiya, „Spring Boot + SQS + LocalStack," 13. oktoober 2021. [Võrgumaterjal]. Loetud aadressil: https://medium.com/@anujpatel2809/spring-boot-with-sqs-localstack-46b3b8c79e80/. [Kasutatud 17. oktoober 2024].
[64] T. Fernandes, „Cloud AWS 3.0 – SQS Integration," 4. juuli 2024. [Võrgumaterjal]. Loetud aadressil: https://www.baeldung.com/java-spring-cloud-aws-v3-intro/. [Kasutatud 17. oktoober 2024].
[65] P. P. Kore, M. J. Lohar, M. T. Surve, S. Jadhav, „API Testing Using Postman Tool," 10. detsember 2022. [Võrgumaterjal]. Loetud aadressil: https://www.ijraset.com/research-paper/api-testing-using-postman-tool/. [Kasutatud 29. oktoober 2024].

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

Mina, Janne Jürimaa

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Reaalajas teadete jälgimise lahendus teenuse kasutajatele“, mille juhendaja on Meelis Antoi
   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.

01.01.2025

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

Lisa 2 – Kasutajalood

Teenuse kasutaja kasutajalood:

- Teenuse kasutajana soovin tellida teavituste kohta käivat infot juba tuttavast infoteenuse API lõpp-punktist, et saada vajalikku teavet kiiresti ja mugavalt.
- Teenuse kasutajana soovin, et teavituste info jõuaks minuni mõne sekundi jooksul pärast Teavitusteenuse kaudu lühisõnumite saatmist, et saaksin vajadusel kiiresti reageerida.
- Teenuse kasutajana soovin jälgida saadetud teavituste staatust, et teada saada, kas teavitused on edukalt saadetud.
- Teenuse kasutajana soovin näha saadetud teavituste infot, sealhulgas lühisõnumi saaja aadressi, sisu ja saatmisaega, et saada täielikku ülevaadet.

Arendaja kasutajalood:

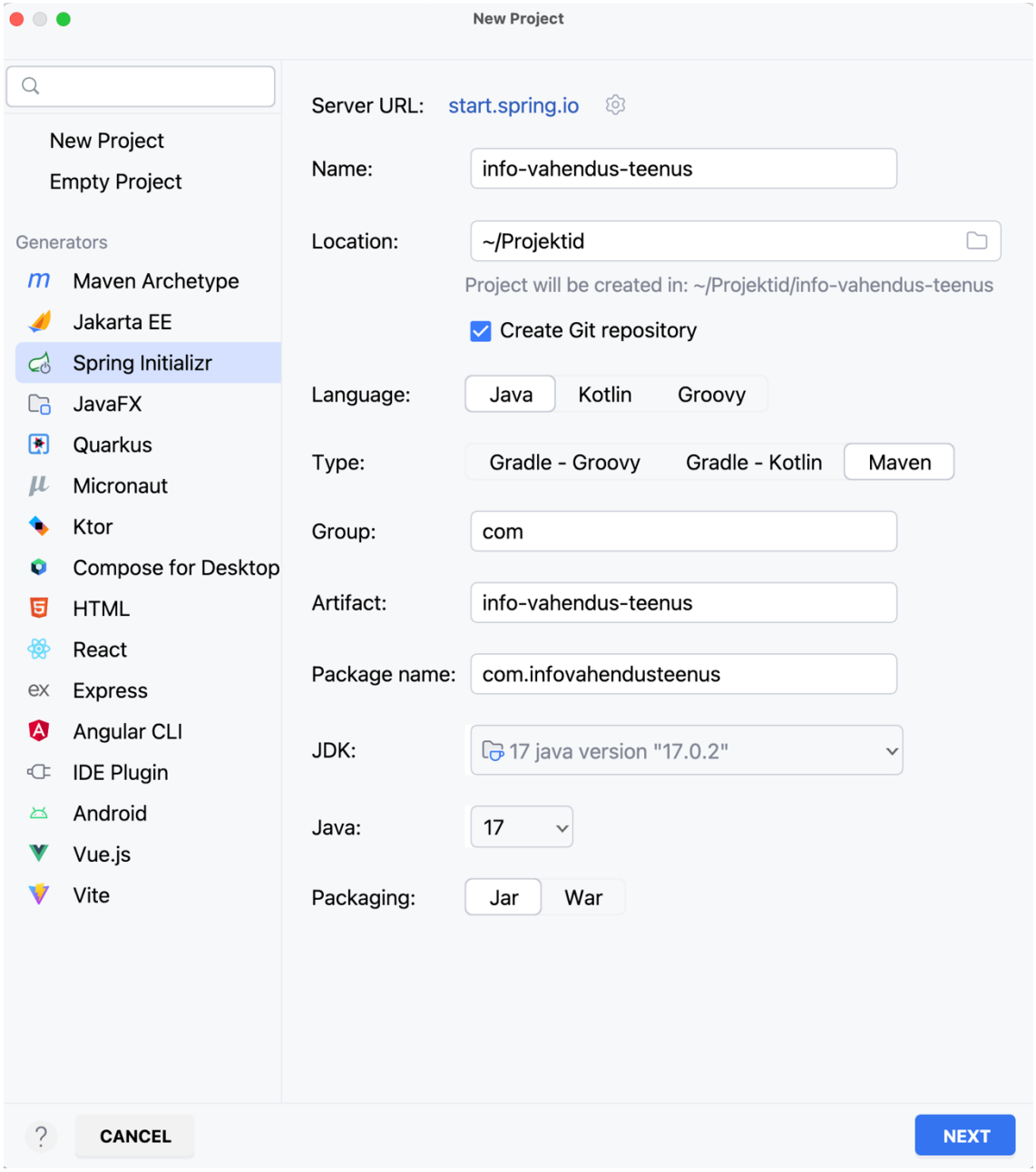
- Arendajana soovin tagada Teavitusteenuse ja Infoteenuse vahelise info liikumise ilma kadudeta, et teenuse kasutajad saaksid usaldusväärset infot.
- Arendajana soovin tagada süsteemi suutlikkuse saata mitu tuhat teadet sekundis, et kasutajad saaksid info kätte kiiresti ja tõrgeteta.
- Arendajana soovin näha logimist iga saadetud teavituse kohta, et tuvastada probleemid kiiresti.
- Arendajana soovin tagada süsteemi skaleeritavuse, et süsteem töötaks efektiivselt kasutajate arvu ja nõudmiste suurenemisel.

Lisa 3 – Arenduskeskkondade kiiruse võrdlus

IntelliJ IDEA, Eclipse'i ja NetBeansi kiiruse võrdlus 200 000 rea koodi indekseerimisel ja kontrollimisel sekundites [47].

IDE	Time to Index 200k Lines of Code	Time to Run Code Inspection on 200k Lines of Code
IntelliJ IDEA	42 seconds	58 seconds
Eclipse	87 seconds	98 seconds
NetBeans	62 seconds	121 seconds

Lisa 4 – IntelliJ IDEA kaudu projekti loomine



Lisa 5 – Avro skeemi näide

Näidisobjekti “SensorData” kohta käiv skeem, mis illustreerib erinevate primitiivsete andmetüüpide kasutust Avro skeemis [56].

```
{
  "type": "record",
  "name": "SensorData",
  "doc": "Schema representing various types of data collected from sensors.",
  "fields": [
    {
      "name": "sensorId",
      "type": "string",
      "doc": "A unique identifier for the sensor."
    },
    {
      "name": "timestamp",
      "type": "long",
      "doc": "Timestamp when the data was collected, in milliseconds since
the epoch."
    },
    {
      "name": "isActive",
      "type": "boolean",
      "doc": "Indicates if the sensor is active."
    },
    {
      "name": "temperature",
      "type": "float",
      "doc": "Temperature reading from the sensor in Celsius."
    },
    {
      "name": "humidity",
      "type": "double",
      "doc": "Humidity reading from the sensor as a percentage."
    },
    {
      "name": "firmwareVersion",
      "type": "bytes",
      "doc": "Binary data representing the sensor's firmware version."
    },
    {
      "name": "statusMessage",
      "type": "string",
      "doc": "A textual status message about the sensor."
    },
    {
      "name": "errorCount",
      "type": "int",
      "doc": "The number of errors reported by the sensor since last reset."
    }
  ]
}
```

Lisa 6 – Automatiseeritud klasside loomise konfiguratsiooni näide

Faas *generate-sources* tähendab, et tegevus toimub lähtekoodi moodustamise faasis. Eesmärk *schema* tähendab, et kompileeritakse määratud skeemifailid ja moodustatakse neist vastavad Java klassid. Konfiguratsioon *sourceDirectory* on kataloog, kust Avro skeem loetakse, *outputDirectory* on väljundkataloog, kuhu Java klassid luuakse ja *includes* määrab, et kõik Avro failid *.avsc* faililaiendiga kaasatakse töötlemiseks. *${project.basedir}* viitab projekti juur-kataloogile, milles antud pom.xml dokument paikneb [31].

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.avro</groupId>
            <artifactId>avro-maven-plugin</artifactId>
            <version>${avro.version}</version>
            <executions>
                <execution>
                    <phase>generate-sources</phase>
                    <goals>
                        <goal>schema</goal>
                    </goals>
                    <configuration>

<sourceDirectory>${project.basedir}/src/main/resources/</sourceDirectory>

<outputDirectory>${project.basedir}/src/main/java/</outputDirectory>
                        <includes>
                            <include>*.avsc</include>
                        </includes>
                    </configuration>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

Lisa 7 – Terraformi abil SQS-i järjekorra seadistamise näide

```
resource "aws_sqs_queue" "sh_queue" {
  name                        = "sh-example-queue"
  delay_seconds               = 10
  visibility_timeout_seconds = 30
  max_message_size            = 2048
  message_retention_seconds   = 86400
  receive_wait_time_seconds   = 2
  sqs_managed_sse_enabled = true
}
```

Parameetrite tähendused, vaikeväärtused ja väärtusevahemikud:

- *name* – järjekorra nimi, vaikimisi määratakse unikaalne nimi SQS-i järjekorrale.
- *delay_seconds* – iga sõnum, mis saadetakse järjekorda jääb tarbijatele seatud sekunditeks nähtamatuks. Vaikeväärtus on 0 sekundit, võimalik väärtusvahemik 0 kuni 900 sekundit (15 minutit).
- *visibility_timeout_seconds* – määrab aja, mille jooksul sõnum jääb nähtamatuks teistele tarbijatele pärast seda, kui see on juba ühele tarbijale saadetud. Vaikeväärtus 30 sekundit, võimalik väärtusvahemik 0 kuni 43200 sekundit (12 tundi).
- *max_message_size* – maksimaalne lubatud sõnumi suurus. Kui sõnum on suurem, siis see lükatakse tagasi. Vaikeväärtus 262144 baiti (256 KiB), võimalik väärtusvahemik 1024 baiti (1 KiB) kuni 256 KiB.
- *message_retention_seconds* – aeg, mille jooksul sõnumeid järjekorras säilitatakse. Pärast määratud aja möödumist eemaldatakse automaatselt kõik sõnumid, mida pole töödeldud või kustutatud. Vaikeväärtus 345600 (4 päeva), võimalik väärtusvahemik 60 (1 minut) kuni 1209600 sekundit (14 päeva).
- *receive_wait_time_seconds* – ajaperiood, mille jooksul päring on ootel. Kui selle aja jooksul sõnum järjekorda ei jõua, siis tagastatakse tühi vastus. Vaikeväärtus 0 sekundit, võimalik väärtusvahemik 0 kuni 20 sekundit.

- *sqs_managed_sse_enabled* – argument, mis lubab või keelab SQS-i sõnumite krüpteerimise. Vaikeväärtus *true* (eesti keeles *tõene* ehk antud kontekstis – krüpteerimine lubatud), võimalikud väärtused *true* või *false* (eesti keeles *väär* ehk antud kontekstis – krüpteerimine keelatud) [61].

Lisa 8 – SQS-i tarbija näide

Meetod *listen* kuulab järjekorda nimega "*HowToDoInJava".* See käivitatakse, kui sõnum saabub määratud järjekorda. Meetodi parameetriks on sõnumi objekti tüüpi *Message*<?>, mis on SQS-i järjekorrast saadud sõnum. Meetod logib sõnumi vastuvõtmise aja, kasutades *LOGGER.info* meetodit ja *OffsetDateTime.now()* funktsiooni, et saada hetke aeg [59].

```
@SqsListener("HowToDoInJava" )
public void listen(Message<?> message) {

  LOGGER.info("Message received on listen method at {}",
OffsetDateTime.now());
}
```