TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Mikk Merimaa 185190IAIB

MIKROTEENUSTE LIIDESTAMINE X-TEEGA OKAS2 NÄITEL

Bakalaureusetöö

Juhendaja: Tarvo Treier
MSc

Kaasjuhendaja: Anton Anikin
MSc

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: Mikk Merimaa

15.05.2025

Annotatsioon

Käesoleva bakalaureusetöö eesmärk on analüüsida ja välja töötada sobivaim lahendus, kuidas liidestada mikroteenuste arhitektuuril üles ehitatud infosüsteem X-tee keskkonnaga. Töö tulemusena töötati välja ja võeti kasutusele praktiline lahendus, mis võimaldab mikroteenustel põhineval infosüsteemil turvaliselt ja efektiivselt X-teega suhelda.

Probleem esines Objekti Kontrollimise ja Andmekogu Süsteem 2.0 (OKAS2) loomisel ning lõputöö raames on läbivalt kasutatud OKAS2-t läbivalt praktilise näitena.

Lõputöö raames analüüsiti erinevaid võimalusi OKAS2 infosüsteemi ja X-tee omavaheliseks integreerimiseks, lähtudes mõlema süsteemi tehnilistest nõuetest. Analüüsi ja riskihinnangu tulemusel valiti sobivaimaks lahenduseks adapterkomponendi loomine ja RabbitMQ kasutamine sõnumimaaklerina.

Valitud lahendus juurutati OKAS2 infosüsteemis ning töö käigus valmis praktiline juhend, mis kirjeldab mikroteenustel põhineva infosüsteemi liidestamist X-teega, kasutades keskset adapterkomponenti ja sõnumipõhist andmevahetust. Lisaks on lõputöös esitatud näide ühe X-tee teenuse tarbimise voo realiseerimisest OKAS2 näitel.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 51 leheküljel, 8 peatükki, 25 joonist, 3 tabelit.

Abstract

Integrating Microservices with X-tee on the Example of OKAS2

The aim of this bachelor's thesis is to analyse and develop the most suitable solution for integrating an information system built on microservices architecture with the X-tee environment. As a result of this work, a practical solution was developed and implemented, enabling a microservice-based information system to communicate securely and efficiently with X-tee.

The problem arose during the development of the Objekti Kontrollimise ja Andmekogu Süsteem 2.0 (OKAS2) information system, which is used as a practical example throughout the thesis.

Throughout the thesis, different options for integrating the OKAS2 information system with X-tee were analysed, taking into account the technical requirements of both systems. Based on the analysis and risk assessment, the most appropriate solution was to create an adapter component and use RabbitMQ as a message broker.

The chosen solution was implemented in the OKAS2 system, and during the process, a practical guide was compiled, outlining the integration of a microservices-based information system with X-tee through a central adapter component and message-based communication. Additionally, the thesis presents an example of how a single X-tee service consumption flow was implemented in the context of OKAS2.

The thesis is written in Estonian and is 51 pages long, including 8 chapters, 25 figures and 3 tables.

Lühendite ja mõistete sõnastik

AMQP	*Advanced Message Queuing Protocol*, RabbitMQ kasutatav sõnumiedastusprotokoll
Andmestruktuur	Füüsiline või loogiline seos andmeüksuste ja andmete endi vahel
Andmevahetus	Kontekstist sõltuv andmete saatmine ja vastuvõtt nende infosisu muutmata
API	*Application Programming Interface*, rakendusliides
Asünkroonne	Protsesside sõltumatus mingitest spetsiifilistest sündmustest, näiteks ühisest ajastusest
Docker	Konteineriseerimist ehk operatsioonisüsteemi tasemel virtualiseerimist võimaldav platvorm
HTTP	*Hypertext Transfer Protocol*, hüperteksti edastuse protokoll
Infosüsteem	Kogum rakendusi, teenuseid, infotehnoloogilisi varasid või muid informatsiooni käitluse komponente
Java	Objektorienteeritud programmeerimiskeel ja platvorm
JAXB	Java objektide konverteerimise tööriist. Tööriist võimaldab Java objekte komplekteerida XML-iks ja vastupidi
JSON	*JavaScript Object Notation*, andmevahetusvorming, mis põhineb JavaScripti alamhulgal
KeyStore	Võtmeladu, võtmehoidla
KeMIT	Keskkonnaministeeriumi Infotehnoloogiakeskus
Klaster	Mingiks otstarbeks ühe tervikuna käsitletav valdkonnaspetsiifiline sarnaste objektide rühm mitmesugustes mudelites
Konteiner	Isoleeritud täitmiskeskkond virtualiseeritud operatsioonisüsteemituuma kasutava tarkvara käitluseks
Kubernetes	Konteinerite ja klastrite haldussüsteem
mTLS	*Mutual Transport Layer Security*, mõlema suhtluspoole vaheline TLS
OCSP	*Online Certificate Status Protocol*, võrgusertifikaadi staatuse protokoll
OKAS	Objekti Kontrollimise ja Andmekogu Süsteem
OKAS2	Objekti Kontrollimise ja Andmekogu Süsteem versioon 2.0

OpenAPI	Spetsifikatsioon, mis kirjeldab veebiteenuseid masinloetaval kujul
Protokoll	Andmevahetuse vormingute ja reeglite formaalne määrang
RabbitMQ	Sõnumiedastus- ja vahendustarkvara
REST	*Representational State Transfer*, hajustöötluse paradigma
SOAP	*Simple Object Access Protocol*, lihtne objektipöörduse protokoll
Spring	Raamistik ja platvorm Java rakenduste loomiseks
Spring Boot	Avatud lähtekoodiga Java raamistik, mis lihtsustab iseseisvate ja tootmiskõlblike Spring ökosüsteemi kasutavate rakenduste loomist
Sõnumimaakler	Programmimoodul, mis edastab sõnumeid rakenduste või protokollide vahel
Sünkroonne	Protsesside sõltuvus mingitest spetsiifilistest sündmustest, näiteks ühisest ajastusest
TLS	*Transport Layer Security*, protokoll rakenduste vaheliseks andmete krüpteerimiseks
TrustStore	Usaldatavate TLS sertifikaatide hoidla
WSDL	*Web Services Description Language*, XML-põhine liidesemääratluskeel veebiteenuste kirjeldamiseks
XML	*Extensible Markup Language*, platvormist sõltumatu laiendatav märgistuskeel
X-Road	Rahvusvaheline andmevahetuskiht, mis põhineb X-tee tehnoloogiatel
X-tee	Eesti riigi andmevahetuskiht infosüsteemide vaheliseks turvaliseks suhtluseks
YAML	*YAML Ain't Markup Language*, inimloetav andmete jadastuse keel

Sisukord

Joonistе loetelu

Tabelite loetelu

1. Sissejuhatus

Kaasaegsete infosüsteemide arenduses kasutatakse laialdaselt mikroteenustel põhinevat arhitektuuri, mis võimaldab arendada modulaarseid ja skaleeritavaid süsteeme. Mikroteenuste arhitektuuri on rakendatud OKAS2 infosüsteemis, tegu on Keskkonnaministeeriumi Infotehnoloogiakeskuse (KeMIT) hallatava süsteemiga. OKAS2 ülesanne on keskkonna järelevalve ja kontrolliprotsesse automatiseerida ja hallata.

OKAS2 on jagatud väiksemateks mooduliteks ning iga moodul on iseseisev mikroteenus. OKAS2 arenduse käigus tekkis probleem, kuidas on võimalik ning mõistlik liidestada moodulid Eesti riigi andmevahetuskihi X-teega.

X-tee on loodud infosüsteemide vaheliseks turvaliseks andmevahetuseks, kuid X-teel puudub otsene tugi mikroteenustega suhtlemiseks. Lõputöö loomise hetkel ei toeta X-tee ühtegi mikroteenuste arhitektuuril põhinevat lahendust infosüsteemi ja X-tee vahelise suhtluse loomiseks. X-tee toetab ainult SOAP ja REST tehnoloogiatel põhinevaid protokolle, mille tõttu puudub võimalus mikroteenuste otsesuhtluseks X-teega ilma täiendavate vahenditeta. OKAS2 koosneb mitmest erinevast mikroteenusest ja peab suhtlema üle X-tee mitme erineva infosüsteemiga. Seetõttu kerkis vajadus uurida, kuidas luua selline liidestuslahendus, mis vastab X-tee ja OKAS2 nõuetele, on hallatav, taaskasutatav, tõhus ja turvaline.

1.1 Ülesanne

Lõputöö keskne probleem on, kuidas liidestada mikroteenuste arhitektuuril üles ehitatud infosüsteem X-tee keskkonnaga. OKAS2 infosüsteemi kontekstis tähendab see lahenduse leidmist, mis võimaldab mikroteenuste ja X-tee liidestamist, arvestades X-tee piiranguid ja tehnilisi nõudeid.

1.2 Eesmärk

Käesoleva lõputöö eesmärk on analüüsida ja välja töötada sobivaim lahendus mikroteenustel põhineva infosüsteemi liidestamiseks X-tee keskkonnaga, võttes aluseks OKAS2 infosüsteemi arhitektuursed ja funktsionaalsed vajadused.

Töö käigus võrreldakse erinevaid lahendusvariante X-teega liidestuse loomisel, arvestades

X-tee piirangute ja tehniliste nõuetega. Võrdlusele järgneb riskianalüüs, mis analüüsib iga variandi mõju süsteemi töökindlusele ja turvalisusele.

Seejärel valitakse võrdlusest sobivaim lähenemine ning rakendatakse seda mikroteenuste liidestamisel X-teega OKAS2 infosüsteemis. Töö tulemusena valmib praktiline juhend, mis kirjeldab mikroteenuste X-teega liidestamist valitud lahenduse alusel ning mida saab kasutada ka teistes sarnase arhitektuuriga infosüsteemides.

Töös luuakse lahendus, mida saab laiemalt taaskasutada, pakkudes üldistatavat arhitektuurimustrit mikroteenustel põhinevate infosüsteemide X-teega liidestamiseks. Kuigi valminud lahendus ei pruugi olla teistes süsteemides üks ühele kasutusele võetav, saab selle põhimõtteid ja tehnilisi lahendusi rakendada pärast väiksemate kohanduste tegemist.

1.3 Töö ülesehitus

Töö algab X-tee ja mikroteenuste arhitektuuri käsitlemisega, et luua tehniline taust mõistmaks, millised tehnilised piirangud ja eripärad X-tee liidestamisel mikroteenustega kaasnevad. Kirjeldatakse X-tee andmevahetuse toimimispõhimõtteid, kasutatavaid protokolle, mikroteenuste arhitektuuri ning sõnumipõhist suhtlust.

Järgmisena antakse ülevaade OKAS2 liidestamisel mikroteenustega vajaminevatest tehnoloogiatest, keskendudes nendele, mida kasutatakse OKAS2 süsteemi X-teega liidestamisel. Esitatakse ülevaade olulistest tööriistadest ja raamistikest, mis lihtsustavad mikroteenuste ja X-tee vahelise andmevahetuse loomist.

Seejärel käsitletakse OKAS2 tausta, kirjeldades vajadust uue infosüsteemi loomiseks. Antakse ülevaade süsteemi arhitektuurist ning liidestusvajadustest, keskendudes X-tee kaudu toimuvale andmevahetusele mitmete väliste registritega.

Järgnevalt analüüsitakse erinevaid võimalusi mikroteenuste X-teega liidestamiseks. Esitatakse kolm võimalikku lähenemisviisi – otsene liidestus, tsentraliseeritud adapterkomponendi kasutamine ning adapteri kasutamine koos sõnumimaakleriga. Iga variandi puhul võrreldakse vastavust OKAS2 ja X-tee nõuetele ning hinnatakse lahendusega kaasnevaid riske.

Valitud lahendusena kirjeldatakse X-tee adapteri loomist koos RabbitMQ sõnumimaakleriga. Peatükis antakse ülevaade adapteri arhitektuurist, kasutatavatest tehnoloogiatest ning liidestusest X-tee turvaserveriga. Lisaks esitatakse praktiline näide, mis illustreerib ühe SOAP teenuse tarbimise protsessi OKAS2 mikroteenuste ja X-tee vahel, kasutades

adapterkomponenti ja RabbitMQ sõnumipõhist andmevahetust.

Seejärel hinnatakse loodud lahenduse toimivust ja vastavust töö eesmärkidele. Analüüsitakse, kas lahendus vastas tehnilistele ja ärilistele nõuetele, ning antakse hinnang töökindlusele ja kasutatavusele.

Töö lõppeb kokkuvõttega, kus võetakse kokku töö käigus tehtud järeldused ning esitatakse soovitusi tulevikus probleemi edasisel uurimisel.

2. Taust

Antud peatükis antakse ülevaade taustast ja teoreetilisest osast, mis lihtsustab lõputööst arusaamist. Alljärgnevalt kirjeldatakse X-tee andmevahetuskihti, SOAP ja REST protokolle ning käsitletakse kokkuvõtvalt mikroteenuseid, sellel põhinevat arhitektuuri ja sündmustepõhist andmevahetust. Nende tehnoloogiate ja arhitektuuride (X-tee, mikroteenused, sõnumipõhine suhtlus) toimimispõhimõtete ja piirangute mõistmine on kriitilise tähtsusega, et analüüsida ja lahendada mikroteenustel põhineva infosüsteemi X-teega liidestamise väljakutseid, mida käesolev töö käsitleb.

2.1 X-tee

X-tee on andmevahetuskiht, mis võimaldab turvalist internetipõhist andmevahetust avaliku ja erasektori organisatsioonide vahel [1]. Tegu on lahendusega, mis võimaldab asutustel turvaliselt infot vahetada. Selle tõttu kasutavad lõputöö skoobis olevad OKAS2 infosüsteemi liidestused X-teed andmevahetuseks.

Andmevahetuse teostamiseks X-tee kaudu loob organisatsioon infosüsteemis ühe või mitu teenust, mis põhinevad eelnevalt kokku lepitud andmestruktuuril. Pärast teenuse loomist registreerib organisatsioon loodud teenuse X-tees. See võimaldab teistel X-teega liitunud organisatsioonidel registreeritud teenuseid tarbida. [2]

2.1.1 Ajalugu ja areng

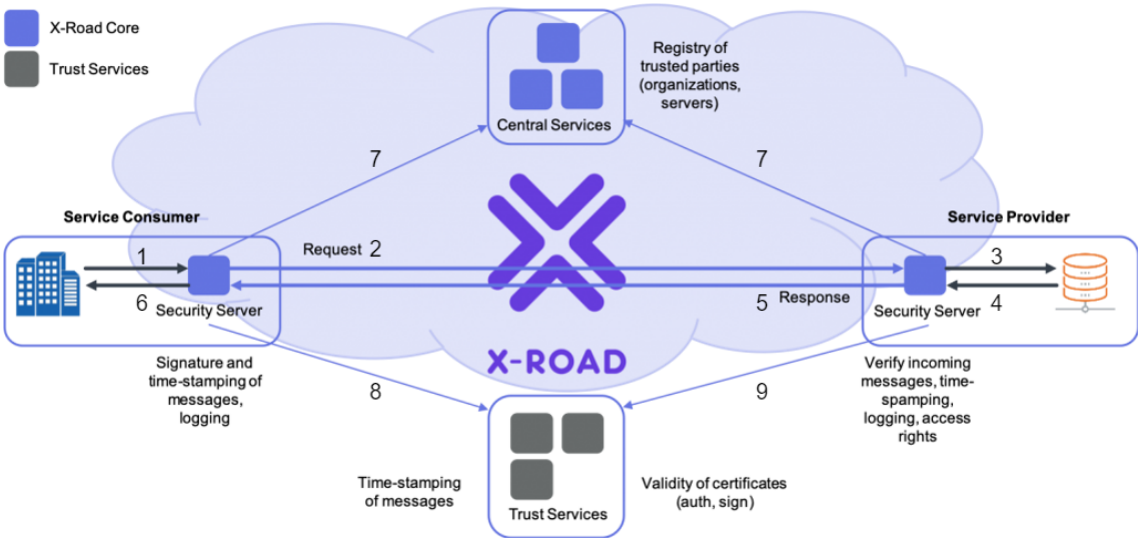
X-tee arendamine sai alguse 2001. aastal vajadusest luua standardiseeritud andmevahetuskiht Eesti riigi avaliku sektori infosüsteemide vahel [3]. Eesti valitsus soovis parandada riiklike infosüsteemide omavahelist suhtlust ning tagada andmevahetuse turvalisus ja tõhusus. Esimene X-tee versioon töötati välja Riigi Infosüsteemi Ameti (RIA) eestvedamisel [1].

Aastate jooksul on X-tee arhitektuur ja tehnoloogia läbinud mitmeid uuendusi. 2013. aastal võttis Soome X-tee baastehnoloogia kasutusele eesmärgiga arendada välja oma riiklik andmevahetuskiht. Eesti ja Soome koostöö tulemusel loodi 2017. aastal Eesti ja Soome vahel Nordic Institute for Interoperability Solutions (NIIS), mis vastutab edaspidise X-tee rahvusvahelise versiooni X-Road arengu eest. [3]

Praeguseks on X-teest kujunenud tsentraliseeritud lahendus, mida kasutatakse laialdaselt erinevates riigi- ja eraettevõtetes. Näiteks 2025. aasta 1. märtsi seisuga on Eesti X-tee keskkonnas liidestatud 1831 infosüsteemi [4]. X-tee loomine ja areng on olnud oluline riiklike infosüsteemide integratsiooni seisukohalt, sest see võimaldab luua turvalist ja standardiseeritud andmevahetust avaliku- ja erasektori organisatsioonide vahel.

2.1.2 Andmevahetuse kirjeldus

Joonis 1 kirjeldab andmevahetuse protsessi kahe X-teega liitunud liikme vahel [5].



Joonis 1. *X-tee andmevahetuse protsess*

Andmevahetuse protsessi käigus:

1. Teenuse tarbija (*Service Consumer*) ehk kliendi rollis infosüsteem koostab päringu ning pöördub sellega kliendi turvaserveri poole.
2. Kliendi turvaserver edastab infosüsteemilt saadud päringu andmeteenust pakkuva (*Service Provider*) infosüsteemi turvaserverile.
3. Teenuse pakkuja turvaserver edastab päringu teenuse pakkuja infosüsteemile.
4. Teenuse pakkuja infosüsteem töötleb päringut ning koostab vastuse, edastades selle pakkujapoolsele turvaserverile.
5. Teenuse pakkuja turvaserver edastab päringu kliendi turvaserverile.
6. Kliendi turvaserver edastab saadud vastuse teenuse tarbija rollis infosüsteemile, kus kasutatakse päringust tulnud informatsiooni edasiseks andmetöötluseks või andmete kuvamiseks.
7. Mõlemad turvaserverid pärivad asünkroonselt keskserveri teenustelt (*Central Services*) globaalset konfiguratsiooni, kus on kirjeldatud kõik X-tee liikmed, turvaserve-

rid ja alamsüsteemid.

8. Andmevahetusega seotud päringuid tehes pöörduvad turvaserverid usaldusteenuste (*Trust Services*) poole. Kliendi poolt tehtavad päringud tembeldatakse ajatempliteenustega.
9. Turvaserverite sertifikaate valideeritakse OCSP (*Online Certificate Status Protocol*) usaldusteenusega, et kinnitada sertifikaatide kehtivus andmevahetuse toimumise ajal.

2.1.3 Suhtlusprotokollid

X-tees on implementeeritud SOAP (*Simple Object Access Protocol*) ja REST (*Representational State Transfer*) arhitektuuril põhinevad protokollid [2]. Protokolle kasutatakse informatsiooni transportimiseks turvaserverite ja infosüsteemide vahel. Järgnevalt on kirjeldatud X-tee protokollide definitsioonid ja sõnumiformaadid. Näidissõnumid on esitatud järgnevates peatükkides, mis kajastavad lõputöö raames OKAS2 infosüsteemis loodud X-tee teenuseid.

SOAP protokoll

X-tee SOAP-protokoll on realiseeritud profiilina, mis põhineb SOAP 1.1 protokollil, kasutades selle üldist mudelit, transpordimehhanismi ja veakäsitlust. Infot transporditakse HTTP (*Hypertext Transfer Protocol*) protokolli kasutades. SOAP sõnumid järgivad XML (*Extensible Markup Language*) sõnumiformaati. SOAP päringute edastamiseks kasutatakse HTTP POST meetodit. [6]

SOAP sõnum sisaldab endas ümbrikut (*envelope*), päist (*header*), keha (*body*) ja vea tekkimisel viga (*fault*). Ümbrik on SOAP sõnumi juurelement ning sisaldab endas päise ja keha elemente. Päis sisaldab metaandmeid sõnumi kohta. Keha sisaldab tegelikku sõnumi sisu ning viga sisaldab selle tekkimise korral vea kohta informatsiooni. [6]

X-tee andmevahetuskihis sisaldavad SOAP sõnumite päised lisaks metaandmeid X-tee protokolli, teenuse, kliendi ja teenust tarbiva kasutaja kohta [7].

WSDL definitsioon

SOAP teenuste spetsifikatsiooni loomiseks kasutatakse WSDL (*Web Service Description Language*) 1.1. WSDL on standardiseeritud veebiteenuste kirjeldamise keel, mis pakub XML formaadis mudelit veebiteenuste defineerimiseks. Definitsioon määratleb detailselt teenuste sisendi, väljundi, operaatorid ja andmestruktuuri. Lisaks võib WSDL spetsifikatsioon sisaldada äriloogikaga seotud valideerimisi. Ühes WSDL definitsioonis võib olla defineeritud üks või mitu teenust. [8]

SOAP-põhise X-tee teenuse registreerimiseks andmevahetuskihis tuleb luua WSDL kirjeldus ning see publitseerida turvaserveris. X-tee turvaserver jagab WSDL definitsiooni organisatsioonidele, võimaldades kliendi rollis süsteemidel teenuseid WSDL-faili alusel tarbida. Pärast teenuse loomist ning definitsiooni avalikustamist on võimalik teenust tarbida üle X-tee. [7]

REST protokoll

Lisaks X-tee SOAP protokollile on X-tees loodud REST arhitektuuristiilil põhinev protokoll. Protokolli loomisega alustati 2018. aasta novembris ning OpenAPI 3.0 toega versioon võeti kasutusele 2019. aasta oktoobris. [9]

REST arhitektuuril põhinevat protokolli hakati kasutama, et lihtsustada X-tee teenuste loomist organisatsioonide vahel. REST toe lisamise peamine eesmärk oli pakkuda teenusepakkujatele ja tarbijatele lihtsamat ja efektiivsemat meetodit teenuste loomiseks, vältides SOAP-ist tulenevaid keerulisi nõudeid ning võimaldades andmevahetust erinevates formaatides. [10]

X-tee REST päringud liiguvad HTTP protokolli vahendusel ning teenuseid kutsutakse välja vastava URL-i kaudu. REST päringuid saab lisaks HTTP POST meetodile teostada ka GET, PUT ja DELETE meetoditega. Erinevalt SOAP-päringutest ei kasuta REST XML-põhist sõnumistruktuuri, andmeid saab edastada mistahes formaadis, kuid üldiselt rakendatakse REST API-sid luues JSON formaati [11]. JSON-i kasutades on mitmeid eeliseid, näiteks on JSON sõnumeid kasutav andmevahetus kiirem ning kasutab vähem ressursse kui XML [12].

X-tee REST teenuse kasutamiseks peab klient pöörduma turvaserveri poole korrektse HTTP päringuga. Metaandmed võimaldavad turvaserveril suunata päringu õigesse süsteemi. Teenuse kliendil on kohustuslik esitada HTTP päis `X-Road-Client`, mis sisaldab kliendi enda metaandmeid. REST-päringu töötlemisel kontrollib turvaserver kliendi õigusi ja suunab päringu vastava teenusepakkuja poole. Erinevalt SOAP-päringutest ei muuda turvaserver REST-päringu sisu ega valideeri andmeformaati. [11]

OpenAPI definitsioon

X-tee REST teenuste kirjeldamisel on soovituslik kasutada OpenAPI spetsifikatsiooni [11], see tagab ühtse ja masinloetava infosüsteemis sisalduvate API-de struktuuri. Infosüsteemi OpenAPI definitsioon on dokumentatsioon süsteemi API-de ülesehitusest, sisaldades täielikku infokogumit olemasolevatest teenustest, toetatud HTTP meetoditest ning teenuste sisend- ja väljundandmete andmestruktuurist. [13]

OpenAPI definitsiooni kirjeldamise formaat on kas JSON või YAML. Sõnumiformaatidest on toetatud nii JSON kui ka XML. OpenAPI definitsiooni alusel saab genereerida tarkvara lähtekoodi, lihtsustades infosüsteemide vahelist liidestamise protsessi. [13]

TLS autentimine

X-tee andmevahetuskihi nõue on, et turvaserveri klientrakendus alati autendib ennast, ehk edastab turvaserveri poole pöördudes TLS sertifikaati. Lisaks on tugevalt soovituslik kasutada HTTPS ühenduse loomisel turvaserveri ja klientrakenduse vahel mõlemapoolset sertifikaatide kontrollimist ehk mTLS. mTLS kasutamine lisab klientrakenduse poolse turvaserveri sertifikaadi valideerimise. See tähendab, et klientrakendus kontrollib turvaserveri sertifikaati, kui turvaserver vahendab välise infosüsteemi päringut klientrakendusele. [14]

2.2 Mikroteenused

Mikroteenused on teenused, mis järgivad mikroteenustel põhinevat arhitektuurimustrit. See lähenemisviis võimaldab infosüsteeme üles ehitada väikeste, iseseisvalt hallatavate komponentidena, suurendades süsteemide paindlikkust, skaleeritavust ja hooldatavust.

Käesolevas töös on arhitektuurimustri mõistmine oluline, et uurida, kuidas mikroteenustel põhinevat süsteemi saab liidestada Eesti riigi andmevahetuskihiga X-tee. See toob esile vajaduse mõista mikroteenuste olemust ning arhitektuurilisi põhimõtteid, et hinnata nende sobivust X-tee tehniliste ja arhitektuuriliste nõuetega. Mikroteenuste ja X-tee arhitektuuride erinevuste tõttu tekivad konkreetsed väljakutsed, mida käesolev töö analüüsib ja millele pakub sobivaid lahendusvariante.

2.2.1 Mikroteenustel põhinev arhitektuur

Mikroteenuste arhitektuuril põhineva infosüsteemi komponente käsitletakse eraldiseisvate, väikeste teenustena. Arhitektuuristiili kasutatakse eesmärgiga jagada suurem rakendus või infosüsteem väiksemateks eraldiseisvateks tükkideks, millest igaüks sisaldab mingit loogilist funktsionaalsust. Väiksemate tükkidena funktsionaalsuse loomine tagab, et iga tükk vastutab mingi konkreetse probleemi eest. [15]

Mikroteenuste arhitektuuristiilil on kaks peamist printsiipi – teenused on sõltumatult juurutatavad (*independently deployable*) ning lõdva sidestusega (*loosely coupled*) [15].

Sõltumatu juurutamine tähendab, et mikroteenust saab eraldiseisvalt paigaldada, käivitada ning hallata. Teenuse põhifunktsionaalsust saab isoleeritult kasutada sõltumatult teistest

mikroteenustest. [16]

Lõtv sidestatus tähendab, et mikroteenus on nõrgalt seotud teiste süsteemi komponentidega. See tagab, et muudatused ühes mikroteenuses mõjutavad minimaalselt teiste teenuste toimimist. Iga teenus kasutab minimaalselt teadmisi teiste teenuste sisemisest struktuurist ja implementatsioonist. Selline lähenemine võimaldab mikroteenuseid arendada, uuendada ja asendada kogu süsteemi häirimata. Lõdva sidestusega süsteemides toimub teenuste vaheline suhtlus konkreetsete liideste ja sõnumipõhiste mehhanismide abil, mis vähendavad sõltuvust teenuste sisemistest muudatustest. [17]

2.2.2 X-tee ja mikroteenused

X-tee on loodud turvaliseks andmevahetuseks erinevate infosüsteemide vahel, kuid see ei ole algselt disainitud mikroteenuste arhitektuuri toetama. Mikroteenused eeldavad hajusat ja paindlikku kommunikatsiooni, kus teenused saavad omavahel suhelda otse. X-tee seevastu on tsentraliseeritud lahendus, mis tugineb kindlatele SOAP- ja REST-põhistele andmevahetusprotokollidele.

X-tee piirang tuleneb sellest, et mikroteenuste otsesuhtlus täiendavate vahenditeta ei ole võimalik. Mikroteenuste arhitektuur toetab erinevaid suhtlusmustreid, sealhulgas sündmustepõhist andmevahetust ja asünkroonset kommunikatsiooni, kuid X-tee pakub ainult SOAP- ja REST-põhiseid sünkroonseid päringuid. Seetõttu ei saa mikroteenused omavahel otse X-tee kaudu suhelda täiendavate tehnoloogiate või erilahendusteta.

Need arhitektuurilised erinevused muudavad mikroteenuste X-tee liidestamise keeruliseks ning nõuavad lisalahendusi, et mikroteenustevahelist suhtlust X-tee kaudu efektiivselt korraldada. Käesolevas töös analüüsitakse neid väljakutseid ning otsitakse võimalikke lahendusi mikroteenuste ja X-tee integreerimiseks.

2.3 Sõnumipõhine arhitektuur

Sõnumipõhine arhitektuur (*Message-Driven Architecture*) ning sündmustepõhine arhitektuur (*Event-Driven Architecture*) on kommunikatsioonimustrid, mida kasutatakse sageli mikroteenuste loomisel. Olemuselt on tegu sarnaste mustritega, kuid käesoleva töö kontekstis keskendutakse sõnumipõhisele arhitektuurile, kuna see vastab paremini vajadusele vahendada konkreetseid päringuid ja vastuseid X-tee teenuste ja mikroteenuste vahel, erinevalt laiemapõhjalisest sündmuste teavitamisest.

2.3.1 Sõnumimaakleri kasutamine mikroteenustes

Sõnumipõhise arhitektuuriga süsteem on süsteem, kus komponendid või teenused suhtlevad omavahel sõnumite kaudu. Arhitektuurilise lähenemise peamine eesmärk on võimaldada süsteemi eri osadel (mikroteenustel) omavahel andmeid vahetada, kasutades sõnumimaaklerit (*message broker*) või otsest sõnumiedastust komponentide vahel. Arhitektuur sisaldab mehhanisme, et tagada sõnumite edastamise usaldusväärsus, et sõnumid jõuaksid alati saatjast vastuvõtjani. [18]

Sõnumipõhine arhitektuur erineb sündmustepõhisest arhitektuurist, sest see keskendub konkreetselt sõnumite edastamisele ja tarbimisele. Sündmustepõhine arhitektuur levitab sündmusi, millele võivad reageerida mitmed süsteemi osad vastavalt vajadusele. [19], [20]

Sõnumipõhine arhitektuur toetab nii sünkroonset kui ka asünkroonset suhtlust. X-tee ja mikroteenuste kontekstis on oluline, et sõnumeid on võimalik töödelda asünkroonselt, võimaldades teenustel töötada sõltumatult ning vältida X-teest tulenevaid päringute sünkroonsuse piiranguid.

3. Kasutatud tehnoloogiad

Lõputöö lahenduse loomisel kasutatakse peamiselt tehnoloogiaid, mis on juba eelnevalt OKAS2 infosüsteemis kasutusele võetud. Selline lähenemine lihtsustab lahenduse hilisemat kasutuselevõttu OKAS2 süsteemis. Lisaks jälgitakse tehnoloogiate valikul seda, et neid kasutades on OKAS2 mikroteenuste liidestamine X-teega võimalikult vähese vaevaga teostatav.

3.1 Java

Java [21] on 1995. aastal välja töötatud töökindel ja efektiivne programmeerimiskeel, mis sobib hästi infosüsteemide arenduseks. Java suurimad eelised on pikaajalise toega LTS (*Long-Term-Support*) versioonid [22], laialdane kasutus, suuremahuline tehniline dokumentatsioon, skaleeritavus ja võimekus mikroteenuste loomiseks ning ühilduvus erinevate raamistike ja tehnoloogiatega. Nende eeliste tõttu võeti Java kasutusele uues OKAS2 infosüsteemis. [23]

3.2 Spring

Spring [24] on Java tarkvaraarenduses laialdaselt kasutatud ökosüsteem. Spring lihtsustab veebirakenduste loomist, sisaldades endas kõike vajalikku, et luua täiemahulist infosüsteemi. Springi suurim eelis OKAS2 kontekstis on tööriistade olemasolu, mis lihtsustab mikroteenuste loomist ja haldamist.

3.2.1 Spring Boot

Spring Boot [25] raamistik on Spring ökosüsteemi üks osa, mis lihtsustab iseseisvalt seadistatavate ja käivitatavate teenuste loomist. See sobib hästi OKAS2 rakenduse mikroteenuste loomiseks.

3.3 Docker

Docker [26] on tööriist rakenduste konteineriseerimiseks. Docker lihtsustab mikroteenuste arenduse, testimise ja juurutamise protsessi, vähendades keskkonnaerinevustest tingitud probleeme. OKAS2 arhitektuuris kasutatakse Dockerit, et konteineriseerida mikroteenused

ning paigaldada need Kubernetese klastrisse.

3.4 Kubernetes

Kubernetes [27] on avatud lähtekoodiga platvorm, mis võimaldab paigaldada ning hallata konteineriseeritud rakendusi. OKAS2 kontekstis kasutatakse Kubernetest mikroteenuste paigaldamiseks, seadistamiseks ning teenustevahelise suhtluse loomiseks.

3.5 RabbitMQ

RabbitMQ [28] on mikroteenustes laialdaselt kasutatav sõnumiedastustarkvara, mis võimaldab erinevatel süsteemikomponentidel andmeid vahetada sõnumite kaudu. See sobib hästi mikroteenuste arhitektuuriga, toetades nii sünkroonset kui ka asünkroonset suhtlust. Komponendi sõnumivahetus toimub AMQP (*Advanced Message Queuing Protocol*) protokolli kasutades.

RabbitMQ töö põhineb sõnumite järjekorda seadmise ja edastamise kontseptsioonil. Sõnumid saadetakse maaklerile (*broker*), mis suunab need vastavatesse järjekordadesse (*queue*) määratud seotiste (*binding*) alusel. Sõnumi edastamine toimub esmalt vahetuspunkti (*exchange*) kaudu, mis vastutab sõnumi suunamise eest õigetesse järjekordadesse. Sõnumeid talletatakse järjekorras seni, kuni neid töötlevad kuulajad (*listener*). Selline tööpõhimõte võimaldab lahtisidestatud (*decoupled*) andmevahetust – sõnumi saatja ja vastuvõtja ei pea olema samaaegselt aktiivsed ega omavahel otseselt seotud.

OKAS2 süsteemis kasutatakse RabbitMQ-d sõnumipõhise arhitektuuri alusel suhtluse loomiseks erinevate moodulite vahel. RabbitMQ kasutamine lihtsustab mikroteenuste vahelist andmeedastust, võimaldades kontrollida sõnumite saatmis- ja vastuvõtmisloogikat, järjekorda ning muid sõnumite transpordiga seotud aspekte. Tööriista kasutamine tagab, et teenustevaheline andmevahetus on lahtisidestatud.

4. OKAS2 infosüsteemi arendus ja liidestused

Käesolevas peatükis antakse ülevaade OKAS2 struktuurist ning komponentidest, mille mõistmine on eelduseks mikroteenuste ja X-tee vahelise liidestuslahenduse loomisel. Fookuses on süsteemi arhitektuur, moodulid ja moodulitevaheline suhtlus ning X-tee liidestusvajadused.

Peatükk algab süsteemi eelkäija Objekti Kontrollimise ja Andmekogu Süsteem 1.0 (OKAS1) tutvustusega, tuues esile erinevad piirangud ja tehnoloogilised takistused, mille tõttu tekkis vajadus uue infosüsteemi loomiseks. Seejärel kirjeldatakse OKAS2 ülesehitust, mikroteenuste loogikat ja moodulite funktsiooni, keskendudes nendele komponentidele, mis on liidestamise seisukohalt kriitilised. Peatüki lõpus käsitletakse X-tee liidestusvajadusi, tuues välja mõned konkreetsed registrid ja teenused, millega OKAS2 peab suhtlema. Esitatud kontekst loob aluse järgmises peatükis kirjeldatavate liidestuslahenduste hindamiseks ja rakendamiseks.

4.1 OKAS1

4.1.1 Taust

Objekti Kontrollimise ja Andmekogu Süsteem 1.0 loodi 2009. aastal eesmärgiga lihtsustada Keskkonnaameti ja Keskkonnainspektsiooni järelevalve- ja haldusprotsesse.

OKAS1 koosnes mitmest moodulist, mis talletasid andmeid objektide ja subjektide kontrollimiste, ebaseaduslike püügivahendite, kalalaevade eelteadete ning haldusmenetluste kohta.

Süsteemi põhieesmärk oli inspektori töö lihtsustamine, koondades tööprotsessideks vajaliku informatsiooni ühtsele töölauale ning tagades kogu kontrolli käigus kogutud andmete elektroonilise säilitamise.

4.1.2 Probleemid

Kuigi OKAS1 täitis algusaastatel oma eesmärki, ilmnes aja jooksul mitmeid olulisi piiranguid, mis pärssisid tööprotsesside tõhusust, süsteemi jätkusuutlikkust ning vastavust kaasaegsetele nõuetele.

Infosüsteem põhines vananenud tehnoloogiaplatvormil, millele mahukate arendustööde tegemine ei olnud enam majanduslikult ega tehniliselt otstarbekas.

Üheks kesksekс probleemiks oli OKAS1 võimetus pakkuda tuge X-tee REST-liidestele. Süsteemis ei olnud võimalik REST-liideseid luua ning olemasolevate SOAP-liidestuste muutmine oli väga kulukas.

Näiteks ei olnud võimalik luua liidest Teadus- ja harrastuskalapüügi andmekogu (TEHA) registri andmete tarbimiseks RESTi kaudu ega ajakohastada liiklusregistri SOAP-liidest pärast ärinõuete muutumist.

OKAS1 kasutamisel ilmnesid ka mitmed õiguslikud ja ärilised probleemid, sealhulgas õigusaktide muutumisest tingitud tööprotsesside ja kasutusvoogude ebasobivus, puudulik tugi kaasaegsetele andmevahetusstandarditele ning piiratud võimalused tööprotsesside optimeerimiseks.

Muutunud vajadused ja regulatiivsed nõuded tõid kaasa olukorra, kus olemasoleva süsteemi edasiarendamine ei olnud enam otstarbekas, mistõttu otsustati välja töötada uus ja kaasaegne lahendus – OKAS2.

4.2 Uus infosüsteem OKAS2

OKAS1 tehniline arhitektuur ja äriprotsessid ei vastanud enam Keskkonnaameti ning Keskkonnaministeeriumi Infotehnoloogiakeskuse ootustele seoses kaasaegsete infosüsteemide toimimise ja haldamisega. Seetõttu otsustati alustada uue infosüsteemi, OKAS2, arendusega.

Uue infosüsteemi nõuded hõlmasid paindlikumat andmevahetust teiste asutustega, paremat tuge haldusmenetluste dokumenteerimisele ja järelevalvetoimingute läbiviimisele, võimalust kasutada süsteemi nutiseadmetes ning tuge tegevuspõhisele kuluarvestusele. Samuti oli oluline, et süsteem toetaks avaandmete avalikustamist ja võimaldaks kasutajakeskseid ning dünaamiliselt kohanduvaid sisestus- ja päringuvorme.

OKAS2 arendamisel asendati vananenud tehnoloogiaplatvorm kaasaegse ja modulaarse lahendusega, mis kasutab mikroteenuseid süsteemi funktsionaalsuste loogiliseks eraldamiseks ja parema hallatavuse tagamiseks. Uues süsteemis loodi tugi X-tee protokolli andmevahetuseks nii REST- kui ka SOAP-teenuste kaudu.

OKAS2 infosüsteemi arendusega alustati 2022. aasta lõpus. Süsteem võeti aastatel 2023 ja

2024 kasutusele mitmes etapis, vastavalt teenuste valmimisele arenduse käigus. Täiemahuline kasutuselevõtt (LIVE) toimus 2025. aasta jaanuaris. Süsteemi loomisega paranes kasutajamugavus, töövood muutusid sujuvamaks ning suurenes ametnike ja teiste huvigruppide töö efektiivsus ja kvaliteet.

4.3 OKAS2 moodulid

Lõputöö raames kajastatakse minimaalse näitena kahte OKAS2 infosüsteemi moodulit.

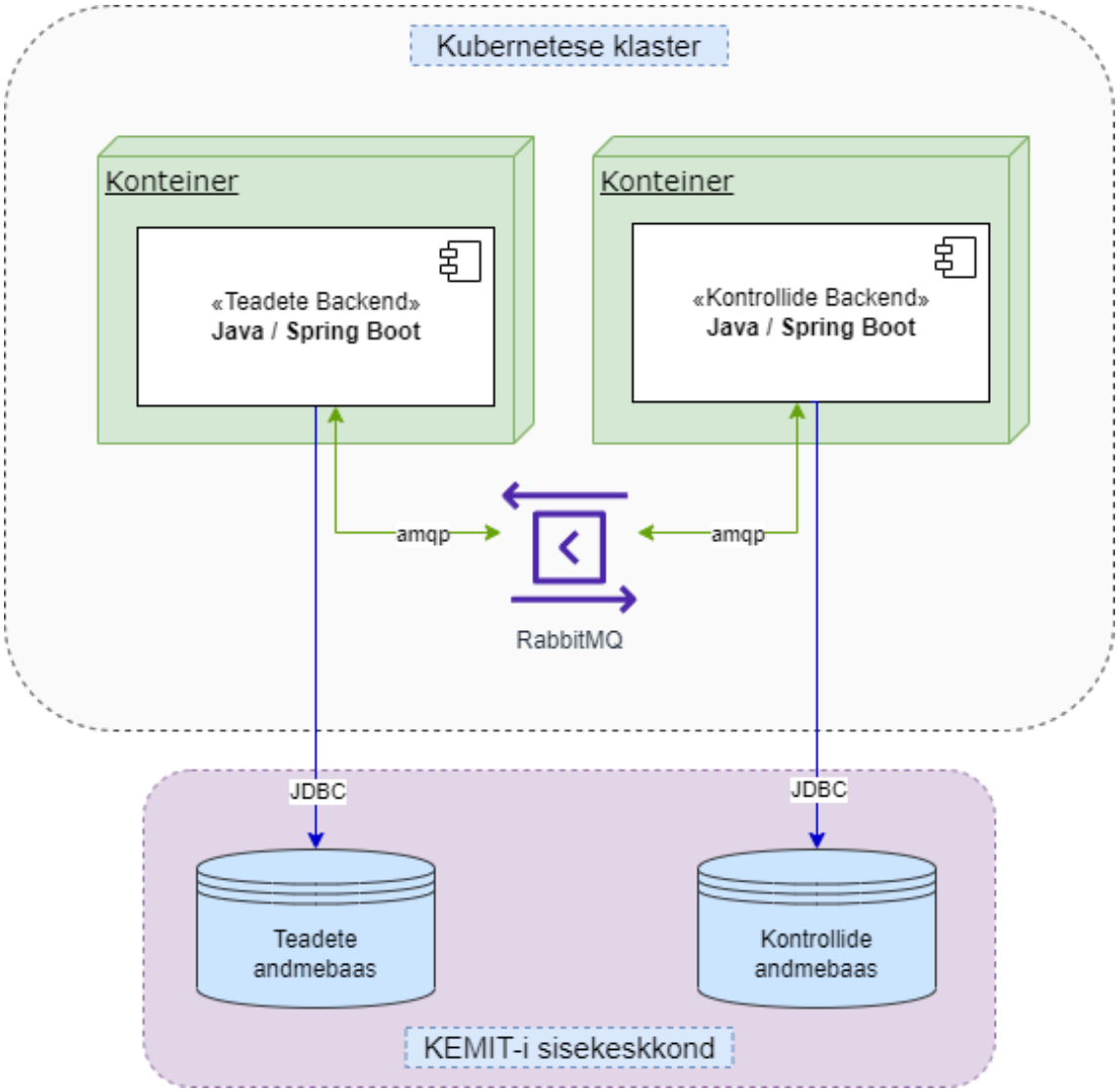
Kontrollide moodul vastutab järelevalvetoimingute dokumenteerimise ja töötlemise eest, sealhulgas kontrollide registreerimise, seostamise objektide ja menetlustega ning seotud protokollide haldamise eest.

Teadete moodul võimaldab keskkonnateadete ja muude teadete (nt kalateated) registreerimist, säilitamist ja töötlemist, pakkudes tuge nii operatiivseks reageerimiseks kui ka statistiliseks analüüsiks.

4.4 OKAS2 arhitektuur

OKAS2 infosüsteemi loomisel kasutati mikroteenustel põhinevat arhitektuurimustrit. Lõputöö kajastab süsteemi arhitektuuri osaliselt, keskendudes süsteemi osadele, mille mainimine on X-tee liidestamisvõimaluste võrdluse koostamise seisukohalt oluline. Lisa 2 on esitatud OKAS2 infosüsteemi täielik arhitektuur LIVE mineku seisuga.

Joonis 2 esitab ülevaate OKAS2 infosüsteemi arhitektuurist.



Joonis 2. *OKAS2 füüsiline vaade*

Joonisel on ülevaade Kubernetese klastrisse paigaldatud komponentidest ning KeMIT-i sisekeskkonnas asuvatest komponentidest. Joonisel ei ole kajastatud kõiki OKAS2 süsteemi komponente ning sisaldab neid minimaalselt, et oleks võimalik analüüsida lõputöös käsitletavat probleemi.

4.4.1 Teadete mooduli komponendid

Teadete *backend* on API rakendus, mis on kirjutatud Java programmeerimiskeeles. Rakendus on loodud, kasutades Spring Boot raamistikku. Rakendus on konteineriseeritud Docker tehnoloogiat kasutades.

Teadete andmebaasis hoitakse teadetega seotud andmeid. Teadete API suhtleb teadete andmebaasiga.

4.4.2 Kontrollide mooduli komponendid

Kontrollide *backend* on API rakendus, mis on kirjutatud Java programmeerimiskeeles. Rakendus on loodud, kasutades Spring Boot raamistikku. Rakendus on konteineriseeritud Docker tehnoloogiat kasutades.

Kontrollide andmebaasis hoitakse kontrollidega seotud andmeid. Kontrollide API suhtleb kontrollide andmebaasiga.

4.4.3 Üldised komponendid

RabbitMQ kasutatakse teadete ja kontrollide API omavahelises suhtluses. Tegemist on klastrisse paigaldatud tervikliku komponendiga, mida ei ole vaja eraldi Dockeriga konteineriseerida.

4.5 X-tee liidestused

Keskkonnaametil on soov luua andmevahetus OKAS2 süsteemis mitme erineva registriga. Lõputöös kajastatakse neist nelja, praktiline näide edastatakse ühe SOAP registri andmete tarbimisest. Registrid ning teenused on valitud sooviga kajastada mõlemapoolset suhtlust ning erinevaid suhtlusprotokolle.

Liiklusregister (LR) on infosüsteem, mis peab arvestust Eesti riigis registreeritud liiklusvahendite üle. OKAS2-l on soov kasutada LR-i pakutavat SOAP teenust sõiduki detailandmete pärimiseks [29].

Hädaabiteadete ning abi- ja infoteadete andmekogu (HKSOS) on riiklik andmekogu, kus töödeldakse andmeid hädaabiteadete ning abi- ja infoteadete menetlemiseks [30]. OKAS2 soovib pakkuda HKSOS-ile teenust teadete vastuvõtmiseks. HKSOSi eripärade tõttu on vajalik teenuse loomisel kasutada X-tee SOAP protokolli. Teated sisaldavad keskkonnajuhtumite, näiteks keskkonnakahjudega seotud andmeid, mille alusel alustatakse hiljem kontrolliprotsesse.

Teadus- ja harrastuspüügi andmekogu (TEHA) on andmekogu, mis haldab eripüügi-, harrastuskalapüügi- ja asustamislubadega seotud andmeid. OKAS2 on vajadus kontrolliprotsesside käigus kontrollida isikuga seotud eripüügilubasid ning elektripüügivahendi tõendeid. TEHAs on loodud teenused andmete pärimiseks, kasutades REST protokolli [31].

Kutselise kalapüügi register (KIR) on andmekogu, mis võimaldab teostada järelevalvet ja pidada arvestust kutselise kalapüügiga tegelevate isikute, nende tegevuse ja kalalaevade üle [32]. KIRil on soov OKAS2-le saata kalanduse eelteadetega seotud andmeid. Tegu on uue vajadusega, mistõttu on selle loomine mõistlik, kasutades X-tee REST protokolli.

Peatükis välja toodud liidesed illustreerivad OKAS2 infosüsteemis erinevaid vajadusi suhelda mitmete väliste infosüsteemidega. Liideste loomine toob esile vajaduse leida lahendus, mis lihtsustab mitme infosüsteemi teenuste tarbimist mikroteenustega infosüsteemis.

5. Mikroteenuste liidestamine X-teega

Käesolev peatükk annab ülevaate erinevatest võimalustest mikroteenuste liidestamiseks X-teega. Esmalt esitatakse kokkuvõtlikult OKAS2 infosüsteemi tehnilised ja funktsionaalsed nõuded, mille mõistmine on eelduseks lahendusvariantide võrdluse loomisel. Seejärel tuuakse välja X-tee andmevahetuskihi tehnilised nõuded ja piirangud, mille täitmine peab olema tagatud sõltumata valitud lahendusest.

Lahendusvariantide analüüsis käsitletakse mitut realistlikku lähenemisviisi mikroteenuste liidestamiseks X-teega. Iga variandi puhul esitatakse lühike kirjeldus ning tuuakse välja eelised ja puudused.

Võrdluse tulemusel koostatakse riskianalüüs, kus hinnatakse iga lahendusvariandi rakendamisega seotud riske, sealhulgas teostatavuse, turvalisuse, hooldatavuse ja skaleeritavuse aspekte.

Peatüki eesmärk on pakkuda terviklikku alusanalüüsi, mille põhjal on võimalik otsustada, milline lahendusvariant on OKAS2 infosüsteemi kontekstis kõige sobivam, arvestades nii X-tee tehniliste piirangute kui ka mikroteenuste arhitektuuri spetsiifikaga.

5.1 OKAS2 infosüsteemi nõuded

Peatükis 4.5 on kirjeldatud OKAS2 infosüsteemi eesmärki luua vajalikud X-tee liidesed väliste infosüsteemidega suhtlemiseks. Lahendusvariantide hindamisel tuleb arvesse võtta järgmisi süsteemile seatud nõudeid:

- Võimalus luua liideseid nii REST kui ka SOAP protokolli spetsifikatsioonide järgi.
- Liideste loomise ja kasutusele võtmise protsess peab olema võimalikult lihtne.
- Arenduskulud peavad olema madalad.
- Halduskulud peavad olema madalad ja haldusprotsess võimalikult lihtne.

5.2 X-tee nõuded ja piirangud

Peatükkides 2.1.2 ja 2.1.3 on esitatud X-tee andmevahetuse tehnilised detailid, mille järgimine on lahendusvariantide hindamisel vajalik. Võrdluses arvestatakse järgnevate nõuetega:

- X-tee toetab ainult SOAP ja REST protokolle.
- X-teed kasutades on vajalik liidestada infosüsteem X-tee turvaserveriga.
- X-tee turvaserveriga suheldes peab olema tagatud turvaline TLS-põhine ühendus.
- Päringutes peavad olema kaasas vajalikud päised ja metaandmed vastavalt X-tee sõnumiprotokolli nõuetele.

5.3 Lahendusvariandid

Lahendusvariantide analüüs viiakse läbi, hinnates iga variandi sobivust eeltoodud OKAS2 ja X-tee nõuete täitmisel. Kõik analüüsitavad lahendused on lõputöö kirjutamise hetkel Eesti X-tee keskkonnas tehniliselt teostatavad ning vastavad kehtivatele SOAP ja REST protokollide nõuetele.

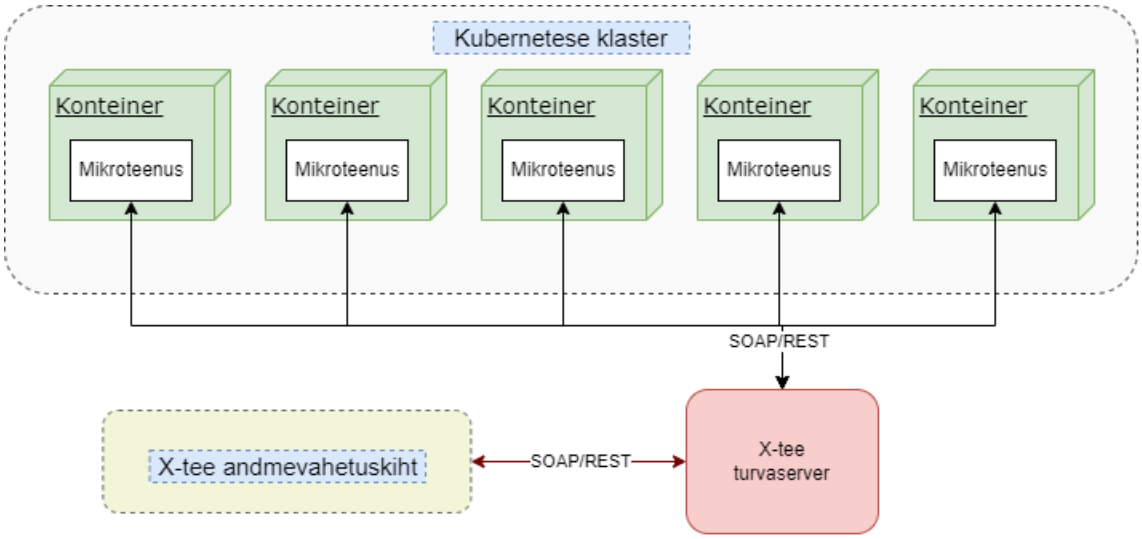
Töös ei käsitleta lahendusvariante, mis eeldavad muudatusi X-tee andmevahetuskihis. Näiteks Siim Düüna magistritöös esitletud lahendus lisab X-tee turvaserverile GraphQL toe, võimaldades X-tee liikmetel esitada päringuid GraphQL-vormingus [33]. Kuigi see lähenemine sobib hästi mikroteenuste arhitektuuriga ja toetab kaasaegsete API-tehnoloogiate kasutuselevõttu, eeldab see muudatusi X-tee andmevahetuskihis. Muudatuste teostamine X-tee tasemel sisaldab mitmeid riske ning ei ole OKAS2 projekti ajaraami ning skoobi tõttu teostatav.

5.3.1 Otsene mikroteenuste liidestamine X-teega

Kirjeldus

Käesolevas lahendusvariandis liidestatakse iga mikroteenus eraldi otse X-teega. See tähendab, et mikroteenused kas pakuvad või tarbivad X-tee teenuseid iseseisvalt, sõltumata infosüsteemi teistest osadest. X-tee liidestamisega seotud loogika, sealhulgas päringute tegemine, päiste määramine ning autentimine, arendatakse igas mikroteenuses eraldi.

Joonis 3 on kujutatud lahendusvariandi arhitektuur, kus liidestatakse mikroteenused otse X-teega.



Joonis 3. *Mikroteenuste otsene liidestamine X-teega*

Iga Kubernetese klastrisse paigaldatud mikroteenus suhtleb X-tee turvaserveriga iseseisvalt, kasutades SOAP- ja/või REST-protokolle.

Eelised

Lähenemisviisi peamised eelised on:

- Võimalik kontrollida ja hallata iga teenust ning seotud liideseid eraldi.
- Teenused võivad olla loodud eri tehnoloogiaid kasutades.
- Pole vaja kasutusele võtta muid tööriistu või tehnoloogiaid.

Puudused

Lähenemisviisi peamised puudused on:

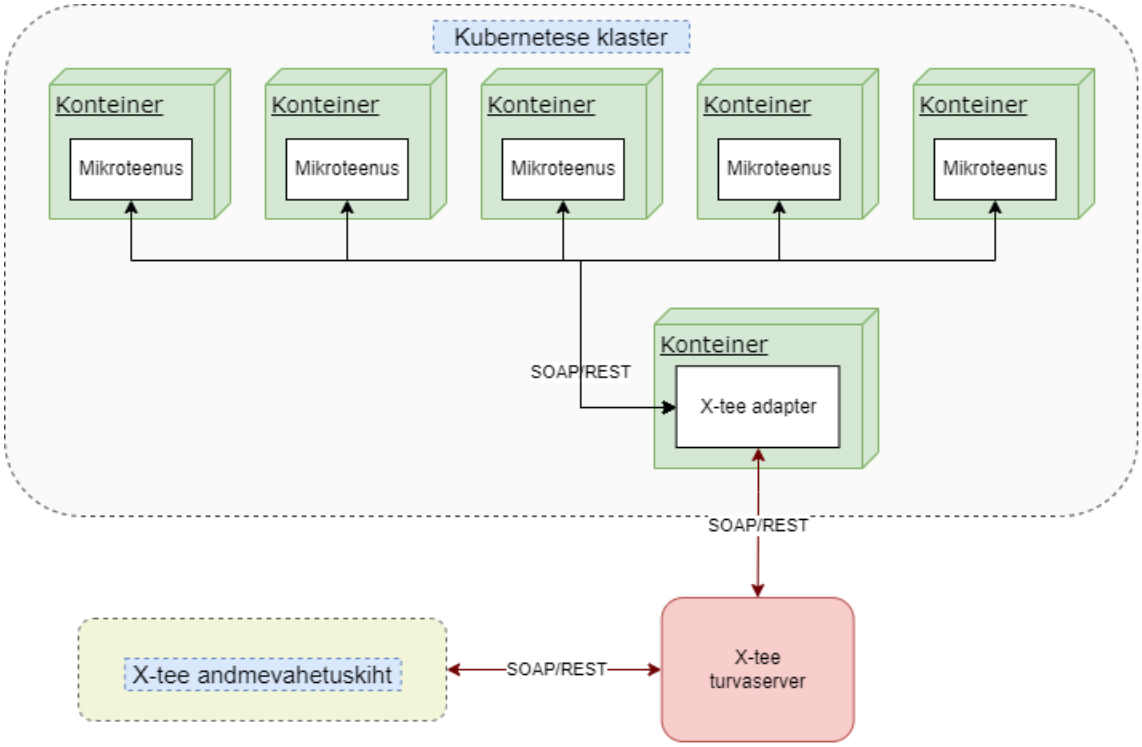
- Liideste loomisel peab loogikat dubleerima igas mikroteenuses, mis tähendab suuri arenduskulusid.
- Liideste arvukuse tõttu on teenuste haldus keeruline ning kulukas.
- Teenuseid on keeruline skaleerida.
- Keeruline on luua ühtset logimislahendust, mis suurendab halduskulusid.

5.3.2 Tsentraliseeritud adapteri kasutamine

Üks variant infosüsteemi ning X-tee andmevahetuskihi vahel suhtluse loomiseks on teha seda adapterkomponenti kasutades. Adapter on komponent infosüsteemi ning X-tee turvaserveri vahel, mille ülesanne on toimida keskse vahelülina, vahendades suhtlust info-

süsteemi mikroteenuste ning X-tee turvaserveriga. Adapter suhtleb X-tee turvaserveriga, kasutades SOAP ja REST protokolle, ning mikroteenustega, kasutades vabalt valitud protokolli või tööriista.

Joonis 4 on esitatud lahendusvariandi arhitektuur, kus kasutatakse tsentraliseeritud adapterit andmevahetuse loomisel.



Joonis 4. *Mikroteenuste liidestamine X-teega, kasutades adapterkomponenti*

Kõik Kubernetese klastrisse paigaldatud mikroteenused suhtlevad otse klastris paikneva X-tee adapteriga. Adapteriga suhtlemiseks kasutavad teenused SOAP- ja/või REST-protokolle. Adapterkomponent vahendab andmevahetust samade protokollide abil X-tee andmevahetuskihiga.

Eelised

Lähenemisviisi peamised eelised on:

- Ainult ühekordne andmevahetuse loomine X-tee adapteri ning turvaserveri vahel.
- X-tee teenuse ning infosüsteemi äriloogika on eraldatud, võimalus andmestruktuuri kohandada infosüsteemile sobivaks.
- Standardiseeritud ja tsentraliseeritud liideste loomine ja haldamine.
- Liideseid saab taaskasutada, vähendades arendus- ja halduskulusid.
- Adapteri haldamine ja uuendamine ei mõjuta teisi mikroteenuseid.

- Adapteri ja mikroteenuste vahel saab kasutada X-tee andmevahetuskihist sõltumatuid tehnoloogiaid.
- Adapteris on X-tee liidestega seotud logimine hallatav ühest kohast.

Puudused

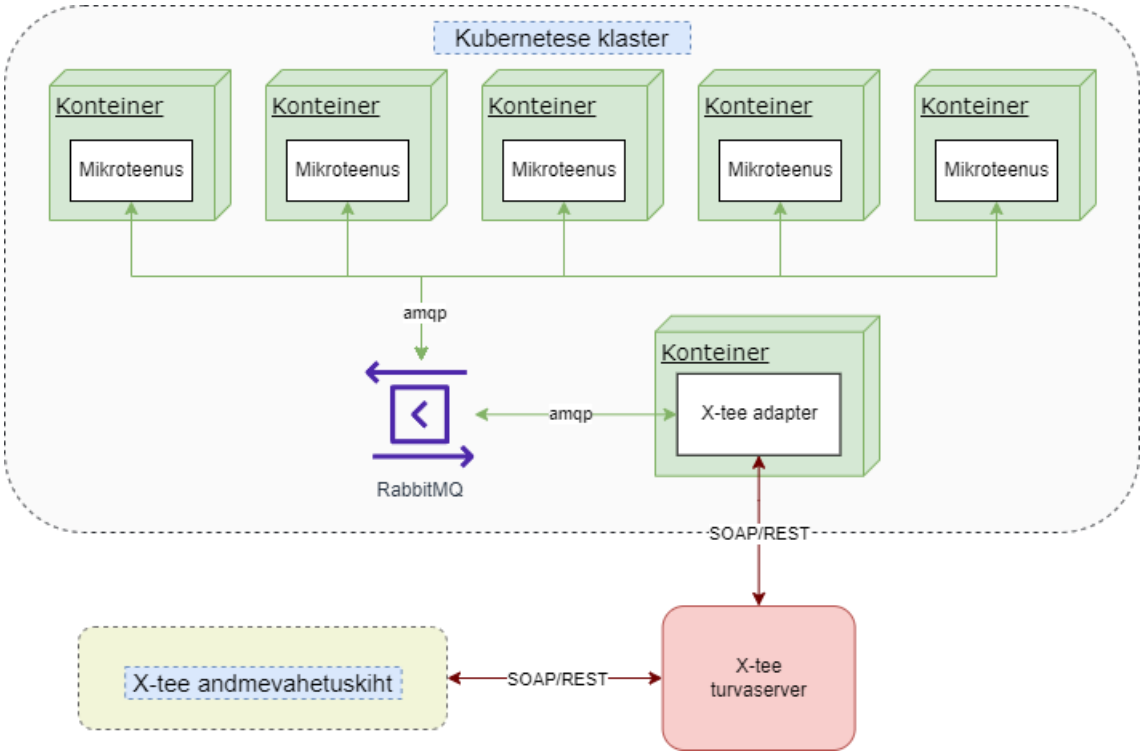
Lähenemisviisi peamised puudused on:

- Vajalik arendada ja hallata adapterit kui eraldiseisvat mikroteenust.
- Adapter on potentsiaalne nõrk lüli, mille tõrge halvab kogu süsteemi.
- Adapteri ja mikroteenuste vahel tuleb luua eraldi liidestused. Iga uus mikroteenus tähendab uut liidestust.
- Suhtlus on sünkroonne, st adapter saab edastada korraga ühte sõnumit.
- Vea tekkimisel lähevad sõnumid kaotsi.

5.3.3 Adapteri kasutamine koos sõnumipõhise andmevahetusega

Lisaks X-tee adapteri kasutusele saab vähendada tsentraliseeritud adapteri puudusi, kasutades sõnumipõhist andmevahetust pakkuvat tööriista, näiteks RabbitMQ-d. Adapter toimib sarnaselt eelmainitule keskse komponendina, kuid edastab X-tee päringud mikroteenustele, suunates need sõnumitena RabbitMQ komponendi maaklerile. Seejärel edastab RabbitMQ sõnumid infosüsteemi mikroteenustele edasiseks töötlemiseks.

Joonis 5 on kujutatud valitud lahendusvariandi arhitektuur, kus kasutatakse tsentraliseeritud adapterit koos sõnumimaakleri RabbitMQ-ga.



Joonis 5. *Mikroteenuste liidestamine X-teega, kasutades adapterkomponenti ja RabbitMQ-d*

Kubernetese klastrisse paigaldatud mikroteenused ei suhtle X-tee adapteriga otse, vaid edastavad sõnumeid läbi RabbitMQ sõnumiedastusteenuse ning AMQP protokolli. Adapterkomponent vahendab andmevahetust RabbitMQ ja X-tee turvaserveri vahel, suheldes sellega otse, kasutades SOAP- ja/või REST-protokolle.

Eelised

Lähenemisviis sisaldab endas kõiki tsentraliseeritud adapteri kasutamise eeliseid. Lisaks eelnevatele lisanduvad järgnevad eelised sõnumite vahendamise tööriista kasutades:

- Suhtlus X-tee adapteri ning infosüsteemi vahel on lahtisidestatud.
- Vea tekkimisel on võimalus sõnumeid tagantjärgi töödelda.
- Sõnumeid saab töödelda asünkroonselt.

Puudused

Lähenemisviisi peamised puudused on:

- Vajalik arendada ja hallata adapterit kui eraldiseisvat mikroteenust.
- Adapter on potentsiaalne nõrk lüli, mille tõrge halvab kogu süsteemi.
- Adapteri ja mikroteenuste vahel tuleb luua eraldi liidestused. Iga uus mikroteenus

tähendab uut liidestust.

- Sõnumipõhise andmevahetuse kasutamine lisab tehnilist keerukust.
- Suurenevad halduskulud seoses andmevahetust vahendava tööriista kasutuselevõtuga.

5.4 Lahendusvariantide võrdlus

Käesolev peatükk koondab peatükis 5.3 käsitletud lahendusvariandid ning võrdleb neid OKAS2 infosüsteemile ja X-tee keskkonnale esitatud nõuete alusel. Võrdluse eesmärk on hinnata, milline lähenemine võimaldab kõige paremini liidestada mikroteenustel põhinev infosüsteem X-teega.

5.4.1 OKAS2 nõuetele vastavus

Selles alapeatükis hinnatakse, mil määral iga lahendusvariant vastab OKAS2 infosüsteemile seatud nõuetele. Arvesse võetakse loomise lihtsust, arendus- ja halduskulusid ning protokollide toe olemasolu. Tabel 1 on esitatud lõputöös käsitlevate lahendusvariantide vastavus nimetatud nõuetele.

Tabel 1. *Lahendusvariantide vastavus OKAS2 nõuetele*

Nõue	**Otsene liidestus**	**Adapterkomponent**	**Adapterkomponent ja RabbitMQ**
Lahendus peab toetama SOAP ja REST protokolle	Kõik protokollipõhised integratsioonid peab eraldi looma.	Protokollipõhised integratsioonid peab arendama ühekordselt, luues need X-tee adapteris.	Protokollipõhised integratsioonid peab arendama ühekordselt, luues need X-tee adapteris.
Liideste loomine peab olema lihtne	Liidesed tuleb luua igas teenuses eraldi. Iga mikroteenuse vahelise suhtluse peab eraldi arendama.	Liidesed tuleb luua ühekordselt tsentraalses X-tee adapteris.	Liidesed tuleb luua ühekordselt tsentraalses X-tee adapteris. Mikroteenuste vahelist suhtlust vahendab RabbitMQ komponent.

Tabel 1. *Lahendusvariantide vastavus OKAS2 nõuetele*

Arenduskulud peavad olema võimalikult madalad	Integratsioonide ja liideste loomisel peab loogikat igas mikroteenuses dubleerima.	Integratsioonide ja liideste loomisel saab kasutada keskset X-tee adapterit. Loogikat peab dubleerima mikroteenuste vahelise suhtluse loomisel.	Integratsioonide ja liideste loomisel saab kasutada keskset X-tee adapterit. Mikroteenuste vahelise suhtluse vahendajana saab kasutada RabbitMQ komponenti.
Teenuste haldamise kulu ning keerukus	Iga teenust peab eraldi haldama.	Peab haldama ainult X-tee adapterkomponenti.	Peab haldama X-tee adapterkomponenti ning lisaks RabbitMQ komponenti.

5.4.2 X-tee nõuetele vastavus

Käesolevas alapeatükis võrreldakse lahendusvariantide vastavust X-tee keskkonna tehnilistele nõuetele, keskendudes X-tee turvaserveri liidestamisele, TLS autentimise loomisele ning X-tee sõnumite päiste ja metaandmete konfigureerimisele. Tabel 2 on koondatud ülevaade iga lahendusvariandi sobivusest X-tee nõuete täitmiseks.

Tabel 2. *Lahendusvariantide vastavus X-tee nõuetele*

Nõue	**Otsene liidestus**	**Adapterkomponent**	**Adapterkomponent ja RabbitMQ**
Liidestus X-tee turvaserveriga	Iga teenuse peab eraldi turvaserveriga liidestama ja konfigureerima.	Ainult X-tee adapteri peab turvaserveriga liidestama.	Ainult X-tee adapteri peab turvaserveriga liidestama.

Tabel 2. *Lahendusvariantide vastavus X-tee nõuetele*

TLS ühenduse loomine	Iga teenuse ja turvaserveri vahelise TLS autentimise peab eraldi looma.	Jah – TLS-ühendus rakendatakse adapteri ja turvaserveri vahel tsentraalselt.	Jah – nagu eelmine, RabbitMQ kasutamine ei mõjuta TLS-i nõuet X-tee tasemel.
X-tee päiste ja metaandmete seadistamine	Igas teenuses tuleb päiste konfiguratsioon ning metaandmete väärtustamine eraldi luua.	Päiste konfiguratsioon ning metaandmete väärtustamine tuleb luua ainult X-tee adapteris.	Päiste konfiguratsioon ning metaandmete väärtustamine tuleb luua ainult X-tee adapteris.

5.5 Riskianalüüs

Käesolevas peatükis antakse ülevaade iga lahendusvariandiga seotud peamistest riskidest ning hinnatakse riski realiseerumise tõenäosust ja mõju OKAS2 infosüsteemile. Riskide mõju ja tõenäosuse hindamisel kasutatakse kolmetasandilist skaalat väärtustega *madal*, *keskmine* ja *kõrge*.

Hinnangute andmisel on arvesse võetud lõputöö autori eelnevat praktilist kogemust sarnaste tehnoloogiate (nt RabbitMQ, X-tee, mikroteenused) kasutamisel teiste avaliku sektori infosüsteemide arendamisel. Lisaks on osade riskide käsitlemisel tuginetud asjakohastele välistele allikatele.

Riskianalüüs ei ole esitatud klassikalise maatriksina, vaid on kombineeritud lahendusvariantide võrdluse tulemustega, et pakkuda paremat ülevaadet seoses konkreetsete lahendusvariantidega. Analüüsi tulemused toetavad järelduste tegemist, milline lahendusvariant on mikroteenuste liidestamisel X-teega kõige sobivam.

Tabel 3 on esitatud lahendusvariantidega seotud olulisemad riskid koos hinnangutega nende tõenäosusele ja mõjule.

Tabel 3. *Lahendusvariantide riskianalüüs tõenäosuse ja mõju alusel*

Lahendus(ed)	**Risk**	**Hinnang riskile**	**Tagajärg**	**Maandamine**
Adapter; Adapter ja RabbitMQ	Tsentraalne adapter kui üks tõrkepunkt [34].	**Tõenäosus:** Keskmine **Mõju:** Kõrge	Adapteri rikke korral peatub kogu X-tee kaudu toimiv andmevahetus infosüsteemis.	Adapteri klasterdamine. Logimine ja monitooring.
Adapter; Adapter ja RabbitMQ	Tsentraalses adapteris tekkivad turvariskid.	**Tõenäosus:** Madal **Mõju:** Kõrge	Ühe komponendi haavatavus ohustab kogu X-tee kaudu toimiva andmevahetuse toimimist.	Regulaarne infosüsteemi hooldus, sõltuvuste uuendamine ning turvatestide läbi viimine.
Otseliidestus; Adapter	Sünkroonsest suhtlusest tekkivad jõudlusprobleemid.	**Tõenäosus:** Keskmine **Mõju:** Keskmine	Koormuse suurenemisel suureneb päringute ooteaeg, süsteem aeglustub.	Koormustestide läbi viimine, teenuste klasterdamine.
Adapter ja RabbitMQ	Andmevahetuses kasutatava tööriista RabbitMQ rike [35].	**Tõenäosus:** Madal **Mõju:** Kõrge	RabbitMQ rikete korral peatub kogu X-tee kaudu toimiv andmevahetus infosüsteemis.	RabbitMQ komponendi klasterdamine. Logimine ja monitooring.

Tabel 3. *Lahendusvariantide riskianalüüs tõenäosuse ja mõju alusel*

Adapter ja RabbitMQ	Tõrgete tekkimisel on oht, et sõnumid võivad jääda töötlemata või kaduma minna [35].	**Tõenäosus:** Madal **Mõju:** Kõrge	Lähevad kaotsi X-tee andmevahetuses tekkinud andmed.	Korrektne RabbitMQ konfiguratsioon. RabbitMQ komponendi klasterdamine. Logimine ja monitooring.
Otseliidestus	Arendus- ja halduskoormuse tekkimine seoses X-tee liideste loomisega.	**Tõenäosus:** Kõrge **Mõju:** Keskmine	Lisakulud infosüsteemi arendus- ja hooldusfaasides.	Risk on süsteemne ja ei ole arhitektuurilise muudatuseta maandatav.

Analüüs näitab, et kuigi adapterkomponendi ning RabbitMQ kasutamisel esineb potentsiaalseid riske, on nende realiseerumise tõenäosus madal ning maandamine suuremate takistusteta teostatav. Järgmises alapeatükis antakse lõplik hinnang lahendusvariantide sobivusele, tuginedes riskianalüüsi tulemustele.

5.6 Järeldused

Lahendusvariantide võrdluse ning riskianalüüsi tulemuste põhjal saab järeldada, et X-tee adapteri ja RabbitMQ kombinatsioon on käesoleva töö kontekstis kõige sobivam lahendus OKAS2 infosüsteemi mikroteenuste liidestamiseks X-tee keskkonnaga. Tegu on lähenemisega, mis vastab X-tee tehnilistele nõuetele ja sobitub OKAS2 arhitektuuriga.

Valitud lahendus on OKAS2 süsteemis lihtsasti kasutusele võetav, sest kasutab juba olemasolevaid tehnoloogiaid ning sobib hästi infosüsteemi olemasoleva taristuga. RabbitMQ on infosüsteemis juba kasutusel mikroteenuste vahelise sõnumivahetuse maaklerina, mistõttu uue mikroteenuse X-tee adapteri loomine ning selle sidumine sõnumipõhise suhtlusega

on võimalik oluliste muudatusteta süsteemi arhitektuuris. Samuti on kõik mikroteenused Dockeriga konteineriseeritud ja paigaldatud Kubernetese klastrisse, mis võimaldab adapteri loomisel kasutada analoogset lahendust. Kõik lahendusvariandi põhikomponendid on süsteemis eelnevalt olemas, valitud lahenduse realiseerimine ei tekita uut tehnilist võlga ega eelda täiendavate tööriistade kasutuselevõttu.

Lähtudes peatükis 5.5 esitatud riskianalüüsist on kõik peamised riskid OKAS2 kontekstis maandatavad. Adapteri kui keskse komponendi tõrkeoht on lahendatav selle paigaldamisega Kubernetese klastrisse. Klastris saab luua mitu identset mikroteenust, millest igaüks täidab adapteri rolli. RabbitMQ töökindlus on tagatud selle korrektse seadistuse ja paigaldusega, mõlemad tööd on juba eelnevalt OKAS2 infosüsteemis tehtud. RabbitMQ töökindlus mõjutab otseselt sõnumite säilimist ja töötlemist, seega on sõnumite kadumise risk madal. Lisaks on OKAS2 infosüsteemis kasutusel tsentraalsed logimise ja monitooringu tööriistad, mis võimaldavad vigade varajast avastamist ja reageerimist. Samad lahendused saab kasutusele võtta adapterkomponendi loomisel. Turvariskide maandamine, süsteemi regulaarne hooldus ning komponentide uuendamine ei erine oluliselt X-tee adapteri puhul teistest süsteemi komponentidest, sest adapteri loomisel kasutatakse samu tehnoloogiaid (Java, Spring jne). Seega võib järeldada, et kõik peamised riskid on maandatud ning süsteemi töökindlus on tagatud ka pärast valitud lahenduse kasutuselevõttu.

Lahendusvariandi kasutuselevõtt on kõige kergemini teostatav, sobitudes olemasoleva OKAS2 infosüsteemi ja arhitektuuriga, ning riskid on maandatud. Tegu on lahendusega, mille loomisel kasutatakse süsteemi olemasolevaid komponente ja tehnoloogiaid. See kinnitab, et X-tee adapteri ja RabbitMQ kasutamine mikroteenuste liidestamisel X-teega OKAS2 näitel on kõige mõistlikum ja praktilisem lahendus.

6. X-tee adapter OKAS2 infosüsteemis

Käesolevas peatükis kirjeldatakse peatükis 5.6 valitud lahendusvariandi realiseerimist OKAS2 infosüsteemis. Esmalt antakse ülevaade loodava komponendi arhitektuurist, ülesehitusest ning kasutatavatest tehnoloogiatest. Seejärel tutvustatakse OKAS2 infosüsteemi arhitektuuri muudatusi seoses adapteri kasutuselevõtuga. Järgnevalt käsitletakse X-tee turvaserveriga liidestamise tehnilisi aspekte ning esitatakse terviklik näide adapteri kasutusest, tuginedes OKAS2 infosüsteemi Liiklusregistri SOAP teenuse tarbimise protsessile. Lõpuks demonstreeritakse RabbitMQ kasutamist sama teenuse X-tee päringute edastamisel. Peatüki tulemusena valmib terviklik ülevaade RabbitMQ kasutavast X-tee adapterkomponendist ning selle rollist OKAS2 süsteemi ja X-tee andmevahetuses. Lugejale antakse praktiline ülevaade adapteri arhitektuurist, kasutuselevõtust ning turvaserveriga liidestamisest.

Praktilise näite kirjeldamisel esitatakse paralleelselt koodinäiteid OKAS2 infosüsteemi lähtekoodist ja konfiguratsioonist. Koodinäited on minimeeritud, eemaldatud on sõltuvuste sisselaadimine (*importid*), annotatsioonid, *getterid*, *setterid* ja muu, mis ei toeta X-tee adapteri lahenduse kirjeldust.

6.1 Adaptermuster

Adaptermuster (*Adapter Pattern*) on tarkvaraarhitektuuri muster, mis võimaldab ühendada omavahel erineva liidese või protokolliga komponente, pakkudes vahekihina toimivat tõlkijat. Mikroteenuste kontekstis kasutatakse adaptereid sageli eri tüüpi liidestega teenuste vaheliseks suhtluseks. Mustri kasutamine võimaldab infrastruktuurile spetsiifilisi detaile muuta abstraktseks ning hõlbustab teenuste ühendamist ja paindlikku ümberkonfigureerimist konteineritele orienteeritud platvormidel, näiteks Kubernetest kasutades. [36]

Käesolevas töös rakendatakse adaptermustrile tuginevat lahendust X-tee ja OKAS2 mikroteenuste vaheliseks integreerimiseks, kus adapter toimib keskse vahendajana, mis teisendab X-tee päringud mikroteenustele sobivasse vormingusse ja suunab vastused tagasi. Selline lähenemine võimaldab hoida teenused lahtisidestatuna, lihtsustades nende hallatavust, skaleeritavust ning edasiarendatavust, vältides X-tee liidestustega seotud keerukuse kordamist igas mikroteenuses.

6.2 Adapterkomponendi ülesehitus ja tehnoloogiad

Adapterkomponent on mikroteenuste arhitektuuriga infosüsteemi (nagu OKAS2) ja X-tee vaheline tsentraliseeritud lüli, mille peamine ülesanne on lihtsustada andmevahetust. Adapterit kasutades ei pea mikroteenused X-tee tehnilisi nõudeid tundma ega täitma, adapter vastutab X-tee turvaserveriga suhtlemise, X-tee teenuste publitseerimise ning tarbimise eest. Adapteri ja mikroteenuste vahelise andmevahetuse loomisel kasutatakse RabbitMQ tehnoloogiat ja sõnumipõhist arhitektuuri, mis võimaldab lahtisidestatud suhtlust ja asünkroonset sõnumivahetust.

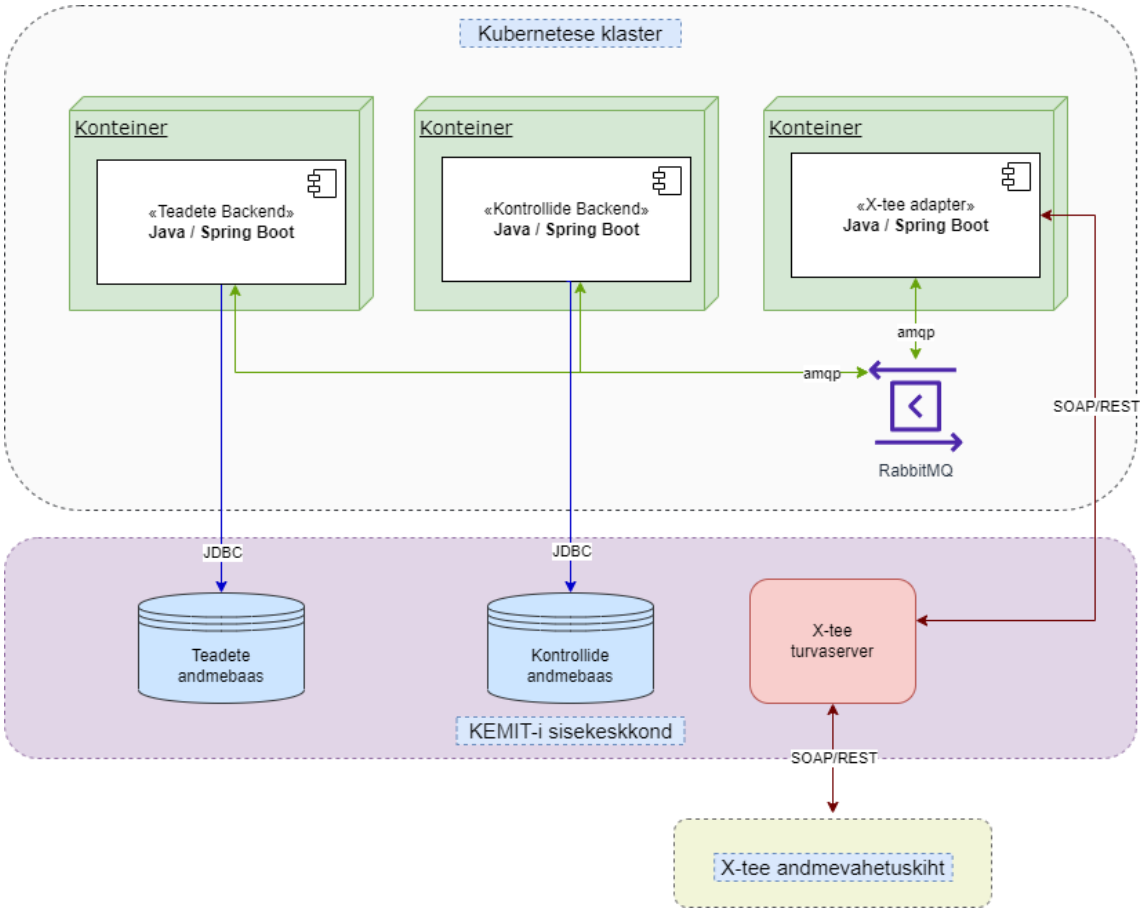
Adapterkomponendi loomine ja kasutuselevõtt tekitab mikroteenuste ja X-tee turvaserveri vahele standardiseeritud vahekihi, mis lihtsustab liideste loomist ning haldamist. Lahendus järgib lõputöös kirjeldatud arhitektuurilisi põhimõtteid, mille kohaselt X-tee liideste loogika ja turvanõuded on koondatud ühte keskselt hallatavasse komponenti. Adapter realiseeritakse iseseisva API-na, mis pakub ühtset ja korduvkasutatavat liidest X-tee teenuste tarbimiseks ja publitseerimiseks kõigile OKAS2 mikroteenustele.

Kasutatavad tehnoloogiad

Komponendi tehnoloogiate valik tugines olemasolevale OKAS2 arhitektuurile ning teistes mikroteenustes kasutatavatele tehnoloogiatele. Adapterkomponendi lähtekood on Java programmeerimiskeeles ning kasutab Spring Boot raamistikku. Komponent integreeritakse süsteemis juba kasutusel oleva RabbitMQ sõnumimaakleriga. Adapter konteineriseeritakse Dockeriga ning paigaldatakse Kubernetese klastrisse. Logimine ja monitooring toimub samuti klastris olemasolevate tööriistadega. Pärast klastrisse paigaldamist on valminud mikroteenus, mida saab kasutada edaspidi X-tee adapterina.

6.3 OKAS2 täienenud arhitektuur

Joonis 6 esitab ülevaate OKAS2 infosüsteemi täiendatud arhitektuurist.



Joonis 6. *OKAS2 füüsiline vaade koos X-tee adapteriga*

Võrreldes Joonis 2 kujutatuga on süsteemi Kubernetese klastrisse lisandunud uus komponent – X-tee adapter. Samuti on KeMIT-i sisekeskkonda lisandunud X-tee turvaserver ning sellega seotud andmevahetuskiht. Turvaserver vastutab andmevahetuse edastamise eest X-tee ning OKAS2 infosüsteemi X-tee adapteri vahel.

Adapter on Docker-tehnoloogia abil konteineriseeritud rakendus, mis on ülesehituselt ja tööpõhimõttelt analoogne teistele Kubernetese klastris paiknevatele komponentidele. Komponent toimib vahelülina, suheldes RabbitMQ komponendiga AMQP protokolli ning X-tee turvaserveriga SOAP ja REST protokolle kasutades.

6.4 X-tee turvaserveri liidestamine

Käesolevas alapeatükis käsitletakse X-tee adapterkomponendi liidestamist X-tee turvaserveriga. Kirjeldatakse turvaserveri seadistust, mille abil tagatakse turvaline andmevahetus OKAS2 infosüsteemi ja X-tee vahel.

6.4.1 TLS autentimine

OKAS2 näitel kasutatakse turvaserveri ja klientrakenduse vahelise autentimisvoo loomiseks mõlemapoolset sertifikaatide valideerimist, ehk mTLS.

Krüptograafilised võtmehoidlad

Java ***KeyStore*** on krüptograafilise failiformaadina pakitud võtmehoidla, mis võib sisaldada klientrakendusele kuuluvat privaatvõtit, sertifikaate koos avalike võtmetega või lihtsalt seotud salasõnu. KeyStore kasutatakse OKAS2 sertifikaadi edastamiseks X-tee turvaserverile.

Java ***TrustStore*** on samamoodi krüptograafiline arhiiv, kuid see tavaliselt sisaldab ainult teiste osapoolte (OKAS2 puhul KeMIT-i turvaserveri) avalikke sertifikaate või usaldatud sertifikaadiahelaid.

mTLS autentimise loomiseks tuleb sertifikaatide valideerimiseks luua KeyStore ja TrustStore failid ning need rakenduse konfiguratsiooni lisada. KeyStore ja TrustStore failid luuakse eelnevalt väljastatud sertifikaadi ja võtmepaaride, või lihtsalt sertifikaadi alusel. Seotud faile saab pakkida, kasutades Java sisse ehitatud *keytool* tööriista, *OpenSSL*i või mingit muud tööriista. Lõputöö raames krüptograafiliste arhiivide loomist ei käsitleta ning eeldatakse, et KeyStore ja TrustStore failid on eelnevalt loodud.

PKCS12 on krüptograafias faili arhiivi formaat. Mainitud formaati kasutatakse OKAS2 KeyStore ja TrustStore failides. Failiformaadi laiend on *p12*. Javat kasutades on vajalik, et KeyStore ja TrustStore peavad olema samas formaadis. [37]

OKAS2 sertifikaadi edastamine turvaserverile

Joonis 7 kujutab X-tee klientrakenduse TLS ühenduse seadistamiseks vajalikke parameetreid.

```
server:
  ssl:
    key-store-type: PKCS12
    key-store: classpath:tls/keystore.p12
    key-store-password:
```

Joonis 7. *Klientrakenduse TLS KeyStore konfiguratsiooniparameetrid*

YAML formaadis on esitatud järgnevad parameetrid:

- server.ssl.key-store-type – KeyStore võtmehoidla arhiveerimise tüüp. Samamoodi

määrab parameeter TrustStore arhiveerimise tüübi. OKAS2 puhul alati PKCS12.
- server.ssl.key-store – KeyStore faili asukoht. Väärtus *classpath* tähistab, et faili otsitakse Spring Boot rakenduse ressurssidest.
- server.ssl.key-store-password – KeyStore faili salasõna.

Esitatud parameetrid tuleb lisada Spring Boot rakenduse konfiguratsioonifaili. Kui KeyStore sisaldab korrektselt lisatud OKAS2 infosüsteemi privaatvõtit ja sertifikaati, võimaldab Spring Boot raamistik rakendusel suhelda X-tee turvaserveriga TLS-protokolli abil. REST- ja SOAP-päringute tegemisel esitatakse automaatselt kliendisertifikaat ja allkirjastatud andmed. Edukaks andmevahetuseks peab ka turvaserveri poolel olema lisatud OKAS2 sertifikaat usaldatud allikana, et autentimist aktsepteerida.

OKAS2 turvaserveri sertifikaadi kontrollimine

Joonis 8 esitab klientrakenduse poolt valideeritava turvaserveri TLS konfigureerimiseks vajalikud parameetrid.

```
server:
  ssl:
    trust-store: classpath:tls/keystore.p12
    trust-store-password:
```

Joonis 8. *Klientrakenduse TLS TrustStore konfiguratsiooniparameetrid*

YAML formaadis on esitatud järgnevad parameetrid:

- server.ssl.trust-store – TrustStore faili asukoht.
- server.ssl.trust-store-password – KeyStore faili salasõna.

Lisaks KeyStore parameetritele peab ka TrustStore parameetrid lisama rakenduse konfiguratsioonifaili. Pärast korrektset parameetrite väärtustamist valideerib süsteem turvaserveri esitatud TLS sertifikaati ning kontrollib, et see on valiidne ning lisatud OKAS2 TrustStore faili. Pärast KeyStore ja TrustStore failide korrektset konfigureerimist on OKAS2 ja X-tee turvaserveri vahel mTLS autentimine loodud.

SSL konteksti konfigureerimine lähtekoodis

KeyStore ja TrustStore konfigureerimine küll loob autentimisvõimalused klientrakenduse ja turvaserveri vahel, kuid ei garanteeri, et autentimist kasutatakse konkreetsetes sõltuvustes, mida kasutatakse klientrakenduse poolsete päringute tegemisel. Lisaks võib tekkida vajadus teha muudatusi sõltuvuste kasutamise loogikas, näiteks OKAS2 puhul eemaldada Spring Boot poolt vaikimisi määratud SOAP päised. Nendest sõltuvustest tulenevalt peab tegema

lähtekoodis muudatusi, seadistades KeyStore ja TrustStore failide asukohad ning kasutused.

Joonis 9 on esitatud näide Spring Boot SSL konteksti loomisest.

```
private SSLContext sslContext() {
try {
  File storeFile = new File(ResourceUtils.getURL(keyStore).toURI());
  KeyStore store = KeyStore.getInstance(storeFile,
                                        keyStorePassword.toCharArray());
  return new SSLContextBuilder()
      .loadKeyMaterial(store, keyStorePassword.toCharArray())
      .loadTrustMaterial(ResourceUtils.getURL(trustStore),
                         trustStorePassword.toCharArray())
      .build();
} catch (Exception e) {
  throw new XRoadClientException(e);
}
}
```

Joonis 9. *Spring Boot SSL konteksti loomine*

SSL konteksti konfigureerimiseks asendatakse vaikimisi Spring Boot kasutuses olev SSL kontekst uuega, kasutades eelnevalt defineeritud KeyStore ja TrustStore väärtusi.

Joonis 10 on esitatud näidis Spring Boot `HttpClient` objekti loomisest.

```
@Bean
public HttpClient httpClient() {
SSLConnectionSocketFactory factory =
    new SSLConnectionSocketFactory(sslContext());

HttpClientConnectionManager cm =
    PoolingHttpClientConnectionManagerBuilder
    .create()
    .setSSLSocketFactory(factory)
    .build();

return HttpClientBuilder.create()
                        .setConnectionManager(cm)
                        .addRequestInterceptorFirst(
       new HttpComponents5MessageSender.RemoveSoapHeadersInterceptor())
                        .build();
}
```

Joonis 10. *Spring Boot HttpClient loomine*

HTTP klienti konfigureeritakse eelnevalt loodud SSL kontekstiga. Lisaks on näites Springi vaikimisi seadistatud SOAP päised eemaldatud, et neid hiljem uuesti luua ja käsitsi hallata.

Joonis 11 esitab näite, kuidas `HttpClient` objekti Springi ökosüsteemis kasutusele võtta.

```
@Bean
public RestTemplate restTemplate() {
return new RestTemplate(requestFactory());
}

@Bean
public HttpComponentsClientHttpRequestFactory requestFactory() {
return new HttpComponentsClientHttpRequestFactory(httpClient());
}

@Bean
public HttpComponents5MessageSender httpComponents5MessageSender() {
return new HttpComponents5MessageSender(httpClient());
}
```

Joonis 11. *Spring Boot HttpClient seadistamine*

Konkreetse näite puhul uuendatakse `HttpComponents5MessageSender` implementatsiooni SOAP põhiste teenuste tarbimisel ning `RestTemplate` implementatsiooni REST teenuste tarbimisel.

6.5 X-tee liidese loomine

Käesoleva alapeatüki eesmärk on esitada ülevaade lõputöö käigus loodud X-tee liidesest OKAS2 infosüsteemi ja Liiklusregistri vahel. Ülevaade antakse Liiklusregistri SOAP teenuse `paring2/v2` liidestamisest [38]. Esitatavad koodinäited põhinevad valmis tehtud liidestusel ja konfiguratsioonil, mida kasutati OKAS2 infosüsteemi arendamisel.

6.5.1 SOAP teenuse tarbimine

Alapeatükis kirjeldatakse, kuidas on realiseeritud SOAP teenuse tarbimine X-tee kaudu OKAS2 adapterkomponendis. Näited on esitatud Liiklusregistri andmete tarbimisest. Lisaks käsitletakse eraldi ka kliendi ja teenuse konfiguratsiooni, SOAP kliendi loomist, päiste seadistamist ning päringu tegemist.

Sõltuvused

SOAP teenuste tarbimiseks Spring Boot ökosüsteemis on vaja kasutusele võtta teek `org.springframework.boot:spring-boot-starter-web-services` [39]. Teegi versioon lisatakse automaatselt Spring Boot versiooni alusel.

Konfiguratsioonifail ning klassid

Joonis 12 on esitatud näidis YAML konfiguratsioonist, kus on määratletud X-tee kliendi OKAS2 ja teenusepakkuja Liiklusregistri metaandmed.

```
xroad:
  client:
    security-server-url: https://security-server-url.ee/
    instance: ee-dev
    member-class: GOV
    member-code: 70008658
    subsystem-code: okas
  motor-register:
    instance: ee-dev
    member-class: GOV
    member-code: 70001490
    subsystem-code: liiklusregister
```

Joonis 12. *X-tee kliendi ja teenuse konfiguratsiooni YAML*

Konfigureeritud väärtused on aluseks päiste genereerimisel ning SOAP päringute teostamisel. Spring Boot laeb väärtused rakendusse automaatselt ning teeb ökosüsteemis kättesaadavaks.

Joonis 13 esitab abstraktse klassi X-tee päiste metaandmetest.

```
public abstract class XRoadHeaderProperties {
  private String instance;
  private String memberClass;
  private String memberCode;
  private String subsystemCode;
}
```

Joonis 13. *XRoadHeaderProperties klassi koodinäide*

Abstraktset klassi kasutatakse nii kliendi kui ka teenusepakkuja konfiguratsiooni loomisel.

Joonis 14 näitab abstraktse klassi `XRoadHeaderProperties` kasutust kliendi metaandmete loomisel.

```
@ConfigurationProperties(prefix = "xroad.client")
public class XRoadClientProperties extends XRoadHeaderProperties {
  @NotEmpty
  private String securityServerUrl;
}
```

Joonis 14. *XRoadClientProperties klassi koodinäide*

Metaandmed sisaldavad lisaks eelnevatele ka turvaserveri aadressi, kuhu pöörduda päringute tegemisel.

Joonis 15 on esitatud klass `MotorRegisterProperties`.

```
@ConfigurationProperties(prefix = "xroad.motor-register")
public class MotorRegisterProperties extends XRoadHeaderProperties {
}
```

Joonis 15. *MotorRegisterProperties klassi koodinäide*

Klassi loomine on vajalik, et Spring Boot raamistikus saaks eelnevalt YAML failis defineeritud Liiklusregistri metaandmeid kasutada.

SOAP kliendi loomine ja päringu tegemine

Joonis 16 esitatud näitel luuakse SOAP klient `MotorRegisterClient`, mida kasutatakse edaspidi Liiklusregistri X-tee päringute tegemisel.

```
public class MotorRegisterConfig {
  private final XRoadClientProperties xRoadClientProperties;
  private final MotorRegisterProperties motorRegisterProperties;

  @Bean
  public MotorRegisterClient motorRegisterClient(
                              HttpComponents5MessageSender sender) {
    MotorRegisterClient client = new MotorRegisterClient(
                                    xRoadClientProperties,
                                    motorRegisterProperties);
    client.setDefaultUri(xRoadClientProperties.getSecurityServerUrl());
    XRoadSoapClientUtils.addGenericMarshaller(client,
                                     "eu.x_road.liiklusregister");
    client.setMessageSender(sender);
    XRoadSoapClientUtils.addGenericInterceptors(client);
    return client;
  }
}
```

Joonis 16. *Liiklusregistri SOAP kliendi konfigureerimine*

Kliendi loomisel kasutatakse eelnevalt seadistatud X-tee turvaserveri ning liiklusregistri konfiguratsiooniklasse, andes edasi eeldefineeritud metaandmed. Lisaks seadistatakse liiklusregistri kliendis X-tee turvaserveri aadress, kasutades `.setDefaultUri` meetodit.

OKAS2 lahendus genereerib Liiklusregistri andmestruktuuri automaatselt, kasutades rakenduse ressurssidesse lisatud WSDL faili. Lisa 3 on esitatud näide JAXB tööriista kasutamisest koodifailide genereerimisel. Koodifailid tekivad Java pakki `eu.x_-`

road.liiklusregister. Abiklassi XRoadSoapClientUtils kasutatakse Joonis 16 toodud näites koodifailide paki asukoha määramisel ja JAXB konfigureerimisel. Lisa 4 on välja toodud abiklassi täispikk lähtekood.

Joonis 17 näitab, kuidas Springi WebServiceGatewaySupport klassi laiendada.

```
@RequiredArgsConstructor
public class MotorRegisterClient extends WebServiceGatewaySupport {
  private final ObjectFactory objectFactory = new ObjectFactory();
  private final XRoadClientProperties xRoadClientProperties;
  private final MotorRegisterProperties motorRegisterProperties;
  ...
}
```

Joonis 17. *Liiklusregistri SOAP klient MotorRegisterClient*

Klassi laiendamine on kõige tähtsam osa X-tee liideste loomisel, see võimaldab Spring Boot rakenduse funktsionaalsust Liiklusregistri SOAP kliendis MotorRegisterClient kasutada. WSDL alusel JAXB tööriistaga genereeritud klasside kasutamiseks ning SOAP objektide loomiseks kasutatakse tööriista poolt genereeritud abiklassi ObjectFactory.

Joonis 18 näitab Liiklusregistri päringu teostamist üle X-tee, kasutades ära eelnevalt loodud ja defineeritud klasse.

```
  public Paring2Vastus getRequest2Response(
         Request2RequestDto requestDto) {
    Paring2Paring request = Request2RequestMapper.mapRequest(requestDto,
        objectFactory);
    JAXBElement<Paring2Paring> jaxbRequest =
                objectFactory.createParing2(request);
    JAXBElement<Paring2Vastus> responseElement =
        (JAXBElement<Paring2Vastus>) getWebServiceTemplate()
        .marshalSendAndReceive(jaxbRequest,
                               getRequest2ResponseCallback());
    return responseElement.getValue();
  }
```

Joonis 18. *Liiklusregistri SOAP päringu tegemine*

Paring2Paring klass on WSDL alusel genereeritud Liiklusregistri andmestruktuur, meetodi sisendisse edastatud OKAS2 andmestruktuur tõlgendatakse Liiklusregistri struktuuri. Seejärel kasutatakse ObjectFactory klassi SOAP elementide loomisel. SOAP elemendid tõlgendatakse Springi funktsionaalsust kasutades XML päringuks. Päring edastatakse turvaserverile ning meetod tagastab Liiklusregistri vastuse Paring2Vastus.

Päiste määramine

Joonis 19 on toodud näide abiklassi `XRoadSoapHeaderUtils` kasutusest SOAP päiste määramisel.

```
private WebServiceMessageCallback getRequest2ResponseCallback() {
  return message -> XRoadSoapHeaderUtils.addClientServiceHeaders(
                      message,
                      xRoadClientProperties,
                      motorRegisterProperties,
                      "paring2",
                      "v2",
                      "");
}
```

Joonis 19. *Liiklusregistri päringus kliendi ja teenuse SOAP päiste määramine*

Abiklassiga lisatakse X-tee klientrakenduse (OKAS2) ja tarbitava teenuse (Liiklusregister) päised. Lisa 5 on esitatud abiklassi `XRoadSoapHeaderUtils` lähtekood.

Joonis 20 on esile toodud `XRoadSoapHeaderUtils` klassi `addClientHeaders` meetod.

```
private static void addClientHeaders(
      SOAPHeader header,
      XRoadHeaderProperties clientProperties)
          throws SOAPException {
  SOAPHeaderElement client = header.addHeaderElement(
      new QName(XRD_NS, "client", "xrd"));
  client.addAttribute(
      new QName(ID_NS, "objectType", "id"), "SUBSYSTEM");
  client.addChildElement("xRoadInstance", "id")
      .setValue(clientProperties.getInstance());
  client.addChildElement("memberClass", "id")
      .setValue(clientProperties.getMemberClass());
  client.addChildElement("memberCode", "id")
      .setValue(clientProperties.getMemberCode());
  client.addChildElement("subsystemCode", "id")
      .setValue(clientProperties.getSubsystemCode());
}
```

Joonis 20. *OKAS2 SOAP päiste väärtustamine metaandmetega*

Koodijupiga väärtustatakse klientrakenduse OKAS2 metaandmed.

Joonis 21 on esile toodud `XRoadSoapHeaderUtils` klassi `addServiceHeaders` meetod.

```
private static void addServiceHeaders(
      SOAPHeader header,
      XRoadHeaderProperties serviceProperties,
      String serviceCode,
      String serviceVersion)
    throws SOAPException {
  SOAPHeaderElement service = header.addHeaderElement(
      new QName(XRD_NS, "service", "xrd"));
  service.addAttribute(
      new QName(ID_NS, "objectType", "id"), "SERVICE");
  service.addChildElement("xRoadInstance", "id")
      .setValue(serviceProperties.getInstance());
  service.addChildElement("memberClass", "id")
      .setValue(serviceProperties.getMemberClass());
  service.addChildElement("memberCode", "id")
      .setValue(serviceProperties.getMemberCode());
  service.addChildElement("subsystemCode", "id")
      .setValue(serviceProperties.getSubsystemCode());
  service.addChildElement("serviceCode", "id")
      .setValue(serviceCode);
  if (serviceVersion != null) {
    service.addChildElement("serviceVersion", "id")
      .setValue(serviceVersion);
  }
}
```

Joonis 21. *Liiklusregistri SOAP päiste väärtustamine metaandmetega*

Koodijupiga väärtustatakse teenust pakkuva infosüsteemi Liiklusregister metaandmed.

Esitatud koodinäidete alusel on valminud funktsionaalsus Liiklusregistri SOAP teenuse `paring2/v2` tarbimiseks X-tee adapteris. Selleks, et teised OKAS2 mikroteenused saaksid kasutada loodud funktsionaalsust Liiklusregistri teenuse tarbimiseks, tuleb kasutusele võtta sõnumeid vahendav komponent.

6.6 RabbitMQ kasutuselevõtt

Mikroteenuste ja adapterkomponendi vahelise suhtluse korraldamiseks kasutatakse peatükis 3.5 kirjeldatud AMQP protokollil põhinevat RabbitMQ sõnumimaaklerit. Tööriista kasutuselevõtt võimaldab mitmel mikroteenusel saata X-tee adapterile sõnumeid asünkroonselt.

Alapeatükis esitatakse jätk Liiklusregistri SOAP teenuse `paring2/v2` tarbimise näitele, andes ülevaate RabbitMQ konfigureerimisest ja kasutusest. Spring Boot pakub RabbitMQ integreerimiseks sisseehitatud mehhanisme, võimaldades lihtsalt konfigureerida vahetusi, järjekordi, seotisi ning kuulajaid, kasutades Spring ökosüsteemis olemasolevaid lahendusi.

6.6.1 RabbitMQ lisamine mikroteenustesse

Järgnevas osas esitatud tegevused on vajalikud mõlemal RabbitMQ osapoolel, ehk X-tee adapteris ja X-tee teenuseid tarbivates mikroteenustes.

Sõltuvused

RabbitMQ kasutuseks Spring Booti ökosüsteemis tuleb süsteemi lisada teek `org.springframework.boot:spring-boot-starter-amqp` [40]. Teegi versiooni määramine ei ole oluline, see lisatakse automaatselt Spring Boot versiooni alusel.

Konfiguratsioon

Pärast teegi lisandumist tuleb Spring Boot rakenduse konfiguratsioonis määrata järgnevad parameetrid:

- spring.rabbitmq.host – RabbitMQ komponendi host.
- spring.rabbitmq.port – RabbitMQ komponendi port.
- spring.rabbitmq.username – RabbitMQ komponendi kasutajanimi.
- spring.rabbitmq.password – RabbitMQ komponendi salasõna.
- spring.rabbitmq.template.routing-key – RabbitMQ marsruutimise võti.

Parameetrite määramine võib erineda infosüsteemi iseärasustest, hetkel on esitatud OKAS2 seadistatud väärtused.

Seadistamine

Joonis 22 on kujutatud konfiguratsiooniklassi `RabbitMqConfig`, milles luuakse marsruutimise võtit kasutades `DirectExchange` tüüpi vahetus.

```
public class RabbitMqConfig {
  @Value("${spring.rabbitmq.template.routing-key}")
  private String routingKey;

  @Bean
  public DirectExchange rabbitExchange() {
    return new DirectExchange(routingKey);
  }

  @Bean
  public MessageConverter messageConverter() {
    return new Jackson2JsonMessageConverter();
  }

  @Bean
  public AmqpTemplate template(ConnectionFactory connectionFactory) {
    final RabbitTemplate rabbitTemplate =
        new RabbitTemplate(connectionFactory);
    rabbitTemplate.setMessageConverter(messageConverter());
    return rabbitTemplate;
  }

  @Bean
  public AsyncRabbitTemplate asyncRabbitTemplate(
         RabbitTemplate rabbitTemplate){
    return new AsyncRabbitTemplate(rabbitTemplate);
  }
}
```

Joonis 22. *RabbitMQ seadistamine Spring Boot ökosüsteemis*

Sõnumite JSON kodeerimiseks kasutatakse Springis olemasolevat `Jackson2JsonMessageConverter` klassi. Lisaks luuakse ja seadistatakse `RabbitTemplate` sünkroonsete päringute ning `AsyncRabbitTemplate` asünkroonsete päringute tegemiseks.

6.6.2 Liiklusregistri päringu vahendamine

Järgnevas alapeatükis tutvustatakse RabbitMQ rolli Liiklusregistri päringu `paring2/v2` näitel, andes ülevaate RabbitMQ kasutamisest mikroteenuste ja adapterkomponendi vahelise suhtluse loomisel.

Järjekorra ja seotise loomine

Joonis 23 esitab X-tee adapteris loodud RabbitMQ sõnumijärjekorra `Queue` ning seotise `Binding` konfiguratsioonid.

```
@Configuration
@RequiredArgsConstructor
public class MotorRegisterMqConfig {

    private final DirectExchange exchange;

    @Bean
    public Queue motorRegisterRequestQueue() {
        return QueueBuilder.durable("motor.register.request").build();
    }

    @Bean
    public Binding motorRegisterRequestBinding() {
        return BindingBuilder.bind(motorRegisterRequestQueue())
                             .to(exchange)
                             .with("motor.register.request");
    }
}
```

Joonis 23. *RabbitMQ järjekorra ja seotise loomine ja seadistamine*

Komponentide loomine võimaldab päringute tegemiseks vajaliku järjekorra loomist. Järjekorra ja seotise loomisel on kasutatud väärtust `motor.register.request`, tegu on unikaalse väärtusega ühes konkreetses RabbitMQ süsteemis, mis võimaldab kontrollida järjekorda, kuhu sõnumid lõpuks jõuavad.

Sõnumi kuulamine X-tee adapteris

Joonis 24 on esitatud adapterisse lisatud RabbitMQ sõnumite kuulaja, mis töötleb väärtusega `motor.register.request` seadistatud järjekorrast tulnud sõnumeid.

```
@RabbitListener(queues = "motor.register.request")
public MotorRegVehicleDetails getVehicleDetails(
       Request2RequestDto request) {
    Paring2Vastus response = getRequest2Response(request);
    return new MotorRegVehicleDetailsMapper.map(response);
}
```

Joonis 24. *RabbitMQ päringu kuulaja Liiklusregistri teenuse tarbimiseks*

Sõnumite kuulaja on defineeritud Spring raamistiku annotatsiooniga `@RabbitListener`. Sõnumite alusel teostatakse X-tee päring ning edastatakse vastus algsele sõnumi saatjale.

Sõnumi saatmine mikroteenusest adapterile

Joonis 25 on esitatud näide, kuidas kutsutakse teenust tarbivas mikroteenuses välja Liiklusregistri päringut, kasutades ära olemasolevat sõnumite järjekorda, seotist ning kuulajat.

```
@RequiredArgsConstructor
@Service
public class MotorRegisterService {

    private final AsyncRabbitTemplate asyncRabbitTemplate;

    public MotorRegVehicleDetails getVehicleDetails(String regNr,
                                                    String vin) {
        RabbitConverterFuture<MotorRegVehicleDetails> response =
            asyncRabbitTemplate.convertSendAndReceiveAsType(
                "motor.register.request",
                toRequest(regNr, vin),
                new ParameterizedTypeReference<>() {}
            );
        return response;
    }
}
```

Joonis 25. *RabbitMQ päringu väljakutse*

Sõnum edastatakse marsruutimisvõtme alusel X-tee adapterile. Adapteris sõnum töödeldakse, teostatakse Liiklusregistri `paring2/v2` X-tee päring ning edastatakse vastus sõnumi välja saatnud mikroteenusele.

RabbitMQ komponendi lisamisega on loodud võimalus OKAS2 mikroteenustes kasutada X-tee adapterit Liiklusregistri SOAP teenuse `paring2/v2` tarbimiseks. Adapteriga suhtlemiseks on kasutusele võetud RabbitMQ komponent, mis edastab sõnumeid mikroteenuste ja adapteri vahel.

6.7 Adapteri töökindlus klastris

Peatükis 5.5 selgus, et adapterkomponenti kasutades on oht, et see on süsteemi nõrk lüli, mille tõrke korral peatub kogu infosüsteemi ja X-tee vaheline andmevahetus. Kättesaadavuse suurendamiseks paigaldatakse adapter Kubernetes klastrisse mitme eksemplarina, mis tähendab, et samast adapterkomponendist töötab korraga mitu identset teenust. Kubernetese klastri sisemehhanismid jagavad automaatselt päringuid aktiivsete eksemplaride vahel. Selline lähenemine tagab kõrgkäideldavuse ja ennetab üksikrikke tekkimist ning sellest tulenevat mõju.

6.8 Testimine ja valideerimine

X-tee adapterkomponendi testimise eesmärk oli veenduda, et loodud lahendus toimib ootuspäraselt ning suudab korrektselt vahendada päringuid X-tee ja mikroteenuste vahel. Testimise käigus kontrolliti adapterkomponendi töökindlust, päringute korrektset toimimist

ning süsteemi käitumist suurema koormuse tekkimisel.

Ärifunktsionaalsuse käsitsi testimiseks kasutati SoapUI [41] tööriista. Testide käigus teostati SOAP-päringud otse adapterkomponendi või turvaserveri suunas. Lisaks ärifunktsionaalsusele kontrolliti testidega, et adapter suudab õigesti töödelda päringuid ja tagastada vastused, järgides X-tee sõnumiprotokolli nõudeid ning säilitades vajalikud päised ja metaandmed.

Koormustestide läbiviimiseks kasutati Gatling [42] tööriista, mille abil kontrolliti rakenduse vastupidavust suurele päringute hulgale lühikese aja jooksul. Testide eesmärk oli hinnata adapterkomponendi jõudlust ja töökindlust suurenenud koormuse tingimustes. Tulemused kinnitasid, et adapterkomponent on suuteline käsitlema suurt hulka päringuid ilma tõrgeteta ning sellest saab järeldada, et nõrga lüli risk on maandatud ning adapteri töökindlus tagatud.

7. Tulemused

Lõputöö raames analüüsiti, hinnati ning võrreldi kolme erinevat lahendusvarianti mikroteenuste liidestamiseks X-teega. Iga variandi puhul hinnati sobivust OKAS2 infosüsteemi arhitektuuri, tuginedes süsteemi ja X-tee andmevahetuskihi nõuetele. Lisaks viidi läbi riskianalüüs, mille käigus käsitleti võimalikke tehnilisi riske. Riskianalüüsi tulemused mängisid olulist rolli kõige sobivama ja praktilisema lahendusvariandi valikul.

Analüüsi tulemusel valiti sobivaimaks lahenduseks tsentraliseeritud adapterkomponent, mis kasutab RabbitMQ sõnumimaaklerit, et vahendada andmevahetust mikroteenuste ja X-tee vahel. Valitud lahendus võimaldas tsentraliseerida X-tee liidestamise loogika, vältides selle dubleerimist mikroteenustes, ning kasutada sõnumipõhist suhtlust X-tee teenuste liidestamiseks sõltumata mikroteenuste kasutatavatest tehnoloogiatest.

Valitud lahendusvariant arendati ja võeti kasutusele OKAS2 süsteemis, mille raames loodi adapterkomponent ning integreeriti see olemasolevate mikroteenustega. Lõputöö tulemusena valmis praktiline juhend, mis kirjeldab adapterkomponendi põhise lahenduse kasutamist mikroteenuste liidestamisel X-teega. Töö on praktiline juhend teistele infosüsteemidele, kus on vajadus tarbida X-tee teenuseid erinevates mikroteenustes.

Loodud lahendus võeti OKAS2 infosüsteemis kasutusele 2025. aasta jaanuaris, mil uut süsteemi hakati täiemahuliselt kasutama. Adapterit kasutatakse reaalseks andmevahetuseks mitme X-tee teenusega, sealhulgas lõputöös kajastatud Transpordiameti Liiklusregistri andmete pärimiseks üle SOAP protokolli.

Reaalses kasutuses on lahendus seni toiminud ootuspäraselt ning suuremaid probleeme ei ole esinenud. RabbitMQ sõnumimaakleri kasutuselevõtuga ei esinenud olulisi tõrkeid X-tee päringutega seotud andmevahetuses. Süsteem suudab käsitleda lühiajalisi katkestusi või mikroteenuse mittesaadavust, kuna sõnumeid talletatakse järjekorras ja neid saab töödelda hiljem. Vähenenud on probleemide lahendamiseks käsitsi sekkumise vajadus, mis ilmnes ka OKAS2 süsteemis, kus pärast vigade tekkimist piisas mõningatest parandustest ning komponentide taaskäivitamisest. Ainsaks väljakutseks kujunes RabbitMQ korrektne kasutamine sünkroonsete päringute kontekstis, mis eeldas lisakonfiguratsiooni ning muutis arenduse keerukamaks.

OKAS1 puhul nõudis eelnevalt iga uus X-tee liides individuaalset SOAP-liidese aren-

dust ning süsteemi täiendamist käsitsi. Lisaks oli X-tee liideste loomine kulukas ning tehnoloogilise võla tõttu keeruline. OKAS2 tsentraliseeritud adapteri ja RabbitMQ kasutuselevõtuga on uute liideste loomine lihtsustunud ja liidestamise ajakulu vähenenud märkimisväärselt. Liideste loogikat ei dubleerita igas mikroteenuses ning standardiseeritud adapteri kasutamine võimaldab lihtsalt uusi liideseid luua ja kasutusele võtta.

Pärast adapterkomponendi ja RabbitMQ kasutuselevõttu on OKAS2 arhitektuur oluliselt paranenud nii töökindluse, arendusmugavuse kui ka halduse osas. Mikroteenuste arhitektuur koos sõnumipõhise andmevahetusega võimaldab liidestusi lisada ja kasutada erinevates komponentides ilma suuremate muudatusteta. Uute X-tee liideste loomise protsess on muutunud märkimisväärselt lihtsamaks ja kiiremaks. Adapteri kasutamine on vähendanud loogika ja funktsionaalsuse dubleerimist, RabbitMQ kasutus tagab parema süsteemi vastupidavuse ja asünkroonse töövõime. Süsteemi integratsioonikiht on nüüd selgemini hallatav ja vajadusel lihtsalt laiendatav, mis omakorda vähendab hoolduskulusid ning võimaldab kiiremini reageerida ärinõuete muutumisele. Lahendus on stabiilsem ning loob tugeva aluse süsteemi edasiseks laiendamiseks ja arendamiseks vastavalt tulevastele vajadustele.

Kuigi lõputöö keskendus OKAS2 infosüsteemile, on välja töötatud lahendus rakendatav ka teistes mikroteenustel põhinevates süsteemides, kus on vaja turvaliselt ja efektiivselt X-teega suhelda. Selline arhitektuur sobib hästi näiteks juhtudel, kus süsteem peab teostama suure koormusega X-tee päringuid või vajab asünkroonset suhtlust. Näiteks 2024. aasta oktoobris on Eksamite Infosüsteemis esinenud tõrkeid seoses koormustaluvusega [43]. Spekuleerides, et juhul, kui tõrked on tingitud X-tee liideste päringute jõudlusest, siis võib olla reaalne X-tee adapteri abil süsteemi töökindluse ja koormustaluvuse tõstmine.

Lõputöös esitatud X-tee adapterlahendust saab rakendada ka rahvusvahelises kontekstis, kuna rahvusvaheline X-Road ja Eesti X-tee on olemuselt sarnased andmevahetusplatvormid [2]. See suurendab lõputöös esitatud lahenduse üldist väärtust ja rahvusvahelist kasutuspotentsiaali.

8. Kokkuvõte

Käesoleva bakalaureusetöö eesmärk oli sobiva lahenduse leidmine ja rakendamine mikroteenustel põhineva infosüsteemi liidestamiseks Eesti andmevahetuskihi X-teega, kasutades praktilise näitena OKAS2 infosüsteemi. Lõputöö eesmärk sai täidetud, loodud lahendus vastab süsteemi arhitektuurilistele ja funktsionaalsetele vajadustele ning praktiliselt kasutusele võetud.

Töö keskne probleem seisnes küsimuses, kuidas liidestada mikroteenuseid X-teega nii, et need suudaksid turvaliselt ja tõhusalt teenuseid pakkuda ja tarbida. Probleemi lahendamist käsitleti OKAS2 infosüsteemi näitel, mille kontekstis loodi arhitektuuriline lahendus ja praktiline juhend sobivaima liidestusviisi rakendamiseks.

Töö käigus analüüsiti kolme võimalikku lahendusviisi: mikroteenuste otseliidestust, tsentraliseeritud adapterkomponendi kasutust ning adapterkomponendi kasutust koos sõnumipõhise andmevahetusega (konkreetse näitena RabbitMQ kasutusega). Iga variandi puhul hinnati vastavust nii OKAS2 kui ka X-tee nõuetele. Seejärel anti ülevaade iga lahendusvariandiga seotud peamistest riskidest, millega arvestada lahendusvariandi valiku tegemises.

Lahendusvariantide võrdlusele ning riskianalüüsile tuginedes osutus kõige optimaalsemaks valikuks tsentraliseeritud adapterkomponendi kasutamine koos RabbitMQ sõnumimaakleriga. Töö käigus võeti adapterkomponent edukalt kasutusele OKAS2 süsteemis, mille alusel valmis praktiline juhend mikroteenuste liidestamiseks X-teega.

Lõputöö probleemile vastates saab üldistada, et mikroteenuste ja X-tee vahelise suhtluse loomisel tasub eelistada tsentraliseeritud adapteri kasutust, eriti olukordades, kus süsteem koosneb paljudest eraldiseisvatest moodulitest või teenustest. Lisaks osutus sõnumipõhise andmevahetuse loomine RabbitMQ kasutamise näitel sobivaks lahenduseks mikroteenuste vahelise suhtluse korraldamiseks, võimaldades X-tee teenustega seotud päringuid hallata eraldi komponendina ning hoida need äriloogikast lahus.

Töö tulemused on rakendatavad igas mikroteenustel põhinevas infosüsteemis, kus tuleb liidestada infosüsteem X-tee või teiste standardiseeritud ja rangelt defineeritud andmevahetuskihtidega. Valitud lahendus – tsentraliseeritud adapteri kasutamine koos RabbitMQ sõnumipõhise andmevahetusega – võimaldab eraldada X-tee spetsiifilise loogika mikroteenuste äriloogikast. Lahenduse kasutamine muudab süsteemi lihtsamini hooldatavaks ja

skaleeritavaks. Lisaks pakuvad töö tulemused laiemat taaskasutusvõimalust, võimaldades valminud lahenduse põhimõtteid ja tehnilisi lahendusi kasutada ka teistes sarnaste nõuetega infosüsteemides.

Tulevikus tasub probleemi edasi uurida, analüüsides probleemi X-tee andmevahetuskihi tasemel. Mõistlik on kaaluda mõne standardse tehnoloogia kasutuselevõttu, mis on mikroteenuste ökosüsteemis laialt levinud. Selline lähenemine vähendaks vajadust vahekihtide loomiseks ning lihtsustaks X-tee liidestamist mikroteenustega.

Kasutatud kirjandus

[1] Riigi Infosüsteemi Amet. “Andmevahetuskiht X-tee”. Accessed: Mar. 1, 2025. [Online]. Available: https://www.ria.ee/riigi-infosusteem/andmevahetuse-platvormid/andmevahetuskiht-x-tee.

[2] Nordic Institute for Interoperability Solutions (NIIS). “X-road architecture, version 1.13, 01.06.2023”. Accessed: Mar. 1, 2025. [Online]. Available: https://docs.x-road.global/Architecture/arc-g_x-road_arhitecture.html.

[3] Nordic Institute for Interoperability Solutions (NIIS). “X-road® history”. Accessed: Mar. 1, 2025. [Online]. Available: https://x-road.global/xroad-history.

[4] “X-tee faktileht”. Accessed: Mar. 1, 2025. [Online]. Available: https://www.x-tee.ee/factsheets/EE/.

[5] Riigi Infosüsteemi Amet. “X-tee andmevahetuse kirjeldus”. Accessed: Mar. 3, 2025. [Online]. Available: https://abi.ria.ee/xtee/andmevahetuse-kirjeldus.

[6] W3C. “Simple Object Access Protocol (SOAP) 1.1”. (2000, May. 08). [Online]. Available: https://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[7] “X-Road: Message Protocol v4.0”. Version: 4.0.25. (2023, June. 16). [Online]. Available: https://x-tee.ee/docs/live/xroad/pr-mess_x-road_message_protocol.html.

[8] W3C. “Web Services Description Language (WSDL) 1.1”. (2001, Mar. 15). [Online]. Available: https://www.w3.org/TR/2001/NOTE-wsdl-20010315.

[9] Riigi Infosüsteemi Ameti blogi. “X-Road REST tugi”. Accessed: Mar. 18, 2025. [Online]. Available: https://blog.ria.ee/x-road-rest-tugi/.

[10] P. Kivimäki. “Two Steps from X-Road REST Support”. (2019, Mar. 27). [Online]. Available: https://www.niis.org/blog/2019/3/25/two-steps-from-the-x-road-rest-support.

[11] “X-Road: Message Protocol for REST”. Version: 1.0.3. (2023, June. 15). [Online]. Available: https://www.x-tee.ee/docs/live/xroad/pr-rest_x-road_message_protocol_for_rest.html.

[12] A.-R. Breje, R. Gyorödi, C. Gyorödi, D. Zmaranda, and G. Pecherle, "Comparative study of data sending methods for xml and json models", *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 12, 2018. DOI: 10.14569/IJACSA.2018.091229. [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2018.091229.

[13] SmartBear Software. "OpenAPI Specification. Version 3.1.1". Accessed: Mar. 18, 2025. [Online]. Available: https://swagger.io/specification/.

[14] Riigi Infosüsteemi Amet. "Turvaserveri ja rakenduse vahelise ühenduse turvamine", Accessed: Apr. 19, 2025. [Online]. Available: https://abi.ria.ee/xtee/turvaserveri-ja-rakenduse-vahelise-uhenduse-turvam.

[15] Chris Richardson. "Microservice Architecture pattern". Accessed: Mar. 19, 2025. [Online]. Available: https://microservices.io/patterns/microservices.html.

[16] Chris Richardson. "Essential characteristics of the microservice architecture: independently deployable". Accessed: Mar. 19, 2025. [Online]. Available: https://microservices.io/post/architecture/2022/05/04/microservice-architecture-essentials-deployability.html.

[17] Chris Richardson. "Essential characteristics of the microservice architecture: loosely coupled". Accessed: Mar. 19, 2025. [Online]. Available: https://microservices.io/post/architecture/2023/03/28/microservice-architecture-essentials-loose-coupling.html.

[18] A. Gómez, M. Iglesias-Urkia, L. Belategi, X. Mendialdua, and J. Cabot, "Model-driven development of asynchronous message-driven architectures with AsyncAPI", *Software and Systems Modeling*, vol. 21, no. 4, pp. 1583–1611, Aug. 2022, ISSN: 1619-1374. DOI: 10.1007/s10270-021-00945-3. [Online]. Available: https://doi.org/10.1007/s10270-021-00945-3.

[19] GeeksforGeeks. "Message-Driven Architecture vs. Event-Driven Architecture". Accessed: Mar. 19, 2025. [Online]. Available: https://www.geeksforgeeks.org/message-driven-architecture-vs-event-driven-architecture/.

[20] A. Singh, V. Singh, A. Aggarwal, and S. Aggarwal, "Event Driven Architecture for Message Streaming data driven Microservices systems residing in distributed version control system", in *2022 International Conference on Innovations in Science and Technology for Sustainable Development (ICISTSD)*, Aug. 2022, 308–312. DOI: 10.1109/ICISTSD55159.2022.10010390. Accessed: Apr. 24, 2025. [Online].

Available: https://ieeexplore.ieee.org/abstract/document/10010390.

[21] Oracle. “Java”. Accessed: Mar. 28, 2025. [Online]. Available: https://www.java.com/en/.

[22] Oracle Java SE Support Roadmap. “Java”. Accessed: Mar. 28, 2025. [Online]. Available: https://www.oracle.com/java/technologies/java-se-support-roadmap.html.

[23] “Benefits and Drawbacks of Java”, Innowise, Accessed: May 7, 2025. [Online]. Available: https://innowise.com/blog/benefits-and-drawbacks-of-java/.

[24] Spring. “Spring Framework”. Accessed: Mar. 28, 2025. [Online]. Available: https://spring.io/projects/spring-framework.

[25] Spring. “Spring Boot”. Accessed: Mar. 28, 2025. [Online]. Available: https://spring.io/projects/spring-boot.

[26] Docker Inc. “What is Docker?” Accessed: Mar. 28, 2025. [Online]. Available: https://docs.docker.com/get-started/docker-overview/.

[27] “Kubernetes Overview”. Accessed: Mar. 28, 2025. [Online]. Available: https://kubernetes.io/docs/concepts/overview/.

[28] CloudAMQP. “Part 1: RabbitMQ for beginners - What is RabbitMQ?” Accessed: Mar. 28, 2025. [Online]. Available: https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html.

[29] Riigi Infosüsteemi Amet. “Liiklusregister”. Accessed: Mar. 29, 2025. [Online]. Available: https://www.riha.ee/Infos%C3%BCsteemid/Vaata/liiklusregister.

[30] Riigi Infosüsteemi Amet. “Hädaabiteadete ning abi- ja infoteadete andmekogu”. Accessed: Mar. 29, 2025. [Online]. Available: https://www.riha.ee/Infos%C3%BCsteemid/Vaata/hksos.

[31] Riigi Infosüsteemi Amet. “Teadus- ja harrastuskalapüügi andmekogu”. Accessed: Mar. 29, 2025. [Online]. Available: https://www.riha.ee/Infos%C3%BCsteemid/Vaata/teha.

[32] Riigi Infosüsteemi Amet. “Kutselise kalapüügi register”. Accessed: Mar. 29, 2025. [Online]. Available: https://www.riha.ee/Infos%C3%BCsteemid/Vaata/kir.agri.

[33] S. Düüna, “SOAP-i ja GraphQL-i võrdlus X-tee näitel”, 2021. Accessed: May 7, 2025. [Online]. Available: https://digikogu.taltech.ee/et/item/b074ec62-81ef-4e30-8151-d565ec0beb67.

[34] "What is a single point of failure (SPOF) and how to avoid them?", Search Data Center, Accessed: Apr. 8, 2025. [Online]. Available: https://www.techtarget.com/searchdatacenter/definition/Single-point-of-failure-SPOF.

[35] ScaleGrid Inc. "Best Practices For Scaling RabbitMQ", Accessed: Apr. 8, 2025. [Online]. Available: https://scalegrid.io/blog/scaling-rabbitmq/.

[36] V. Yussupov, U. Breitenbücher, C. Krieger, F. Leymann, J. Soldani, and M. Wurster, "Pattern-based Modelling, Integration, and Deployment of Microservice Architectures", in *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*, Oct. 2020, pp. 40–50. DOI: 10.1109/EDOC49727.2020.00015. Accessed: Apr. 24, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9233043.

[37] OpenSSL Documentation. "openssl-pkcs12", Accessed: Apr. 19, 2025. [Online]. Available: https://docs.openssl.org/3.3/man1/openssl-pkcs12/.

[38] Riigi Infosüsteemi Amet. "X-tee iseteenindus – /gov/70001490/liiklusregister/paring2/v2", Accessed: Apr. 20, 2025. [Online]. Available: https://www.x-tee.ee/et/service-catalog/paring2/services/341922693/70001490/liiklusregister/EE/v2.

[39] MvnRepository. "Spring-boot-starter-web-services", Accessed: Apr. 20, 2025. [Online]. Available: https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-web-services.

[40] MvnRepository. "Spring boot starter amqp", Accessed: Apr. 16, 2025. [Online]. Available: https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-amqp.

[41] SmartBear Software. "SoapUI", Accessed: Apr. 24, 2025. [Online]. Available: https://www.soapui.org/.

[42] Gatling Corp. "Gatling", Accessed: Apr. 24, 2025. [Online]. Available: https://gatling.io.

[43] K. Koppel. "Eksamite infosüsteemi tabasid tasemetööde ajal järjekordsed tõrked", ERR, Accessed: Apr. 24, 2025. [Online]. Available: https://www.err.ee/1609485685/eksamite-infosusteemi-tabasid-tasemetoode-ajal-jarjekordsed-torked.

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

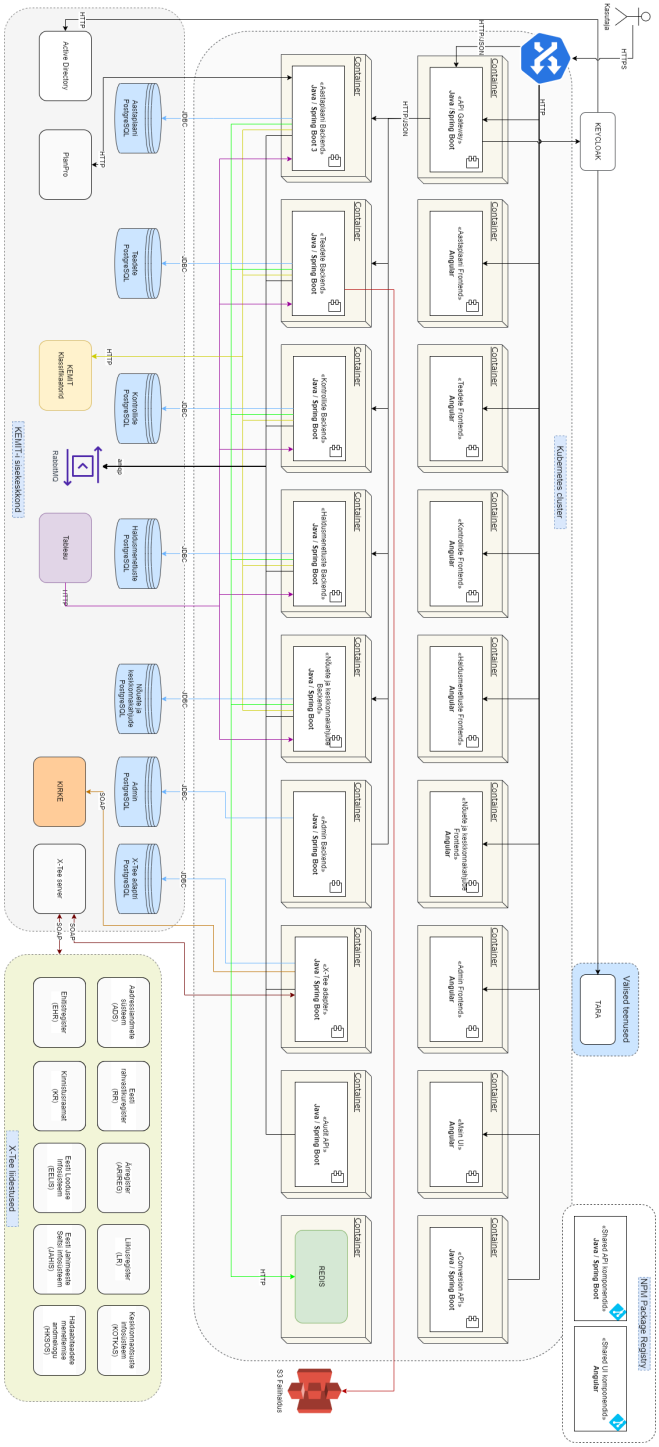
Mina, Mikk Merimaa

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Mikroteenuste liidestamine X-teega OKAS2 näitel", mille juhendajad on Tarvo Treier ja Anton Anikin
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

15.05.2025

[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 - OKAS2 arhitektuur



Lisa 3 - JAXB tööriistaga koodifailide genereerimine

```
 <plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>jaxb2-maven-plugin</artifactId>
    <version>3.2.0</version>
    <executions>
        <execution>
            <id>xjc-non-conflicting</id>
            <goals>
                <goal>xjc</goal>
            </goals>

            <configuration>
                <arguments>
                    <arg>-XautoNameResolution</arg>
                </arguments>
                <failOnNoSchemas>true</failOnNoSchemas>

                <sourceType>wsdl</sourceType>
                <sources>
                    <source>
                        ${xroad.dir}/consumers/motorregister/lr.wsdl
                    </source>
                </sources>
            </configuration>
        </execution>
    </executions>
</plugin>
```

Lisa 4 - XRoadSoapClientUtils klassi lähtekood

```
public final class XRoadSoapClientUtils {
  public static void addGenericMarshaller(
        WebServiceGatewaySupport client,
        String contextPath) {
    Jaxb2Marshaller marshaller = getGenericMarshaller(contextPath);
    client.setMarshaller(marshaller);
    client.setUnmarshaller(marshaller);
  }

  public static void addGenericInterceptors(
        WebServiceGatewaySupport client,
        ClientInterceptor... clientInterceptors) {
    client.setInterceptors(getGenericInterceptors(clientInterceptors));
  }

  private static Jaxb2Marshaller getGenericMarshaller(
        String contextPath) {
    Jaxb2Marshaller marshaller = new Jaxb2Marshaller();
    marshaller.setPackagesToScan(contextPath);
    return marshaller;
  }

  private static ClientInterceptor[] getGenericInterceptors(
        ClientInterceptor... clientInterceptors) {
    List<ClientInterceptor> interceptors = new ArrayList<>();
    interceptors.addAll(List.of(clientInterceptors));
    return interceptors.toArray(new ClientInterceptor[0]);
  }
}
```

Lisa 5 - XRoadSoapHeaderUtils klassi lähtekood

```
@Slf4j
public final class XRoadSoapHeaderUtils {
  private static final String XRD_NS =
      "http://x-road.eu/xsd/xroad.xsd";
  private static final String ID_NS =
      "http://x-road.eu/xsd/identifiers";

  public static <T> T addHeadersFromRequest(
      MessageContext messageContext,
      XRoadHeaderThrowableFunction<XRoadHeader, T> function)
            throws RemoteException, ServiceFaultException {
    try {
      copyHeaders((SaajSoapMessage) messageContext.getRequest(),
                  (SaajSoapMessage) messageContext.getResponse());
      return function.apply(readXRoadHeader(messageContext));
    } catch (ServiceFaultException | RemoteException e) {
      log.error(e.getMessage(), e);
      throw e;
    } catch (Exception e) {
      log.error(e.getMessage(), e);
      throw new RemoteException("Internal␣error");
    }
  }

  private static void copyHeaders(
      SaajSoapMessage request,
      SaajSoapMessage response)
            throws TransformerException,
                        SOAPException {
    SOAPEnvelope envelope = response.getSaajMessage()
                                    .getSOAPPart()
                                    .getEnvelope();
    Transformer transformer = TransformerFactoryProvider
                                    .getTransformerFactory()
                                    .newTransformer();
    Iterator<SoapHeaderElement> itr = request
                                    .getSoapHeader()
                                    .examineAllHeaderElements();
    while (itr.hasNext()) {
      SoapHeaderElement element = itr.next();
      envelope.addNamespaceDeclaration(
                            element.getName().getPrefix(),
```

```
                        element.getName().getNamespaceURI());
    transformer.transform(element.getSource(),
                        response.getSoapHeader().getResult());
  }
}

private static XRoadHeader readXRoadHeader(MessageContext context)
      throws JAXBException {
  SaajSoapMessage saajSoapMessage =
      (SaajSoapMessage) context.getRequest();
  JAXBContext ctx = JAXBContext.newInstance(XRoadHeader.class);
  Unmarshaller unmarshaller = ctx.createUnmarshaller();
  return (XRoadHeader) unmarshaller.unmarshal(
      saajSoapMessage.getSoapHeader().getSource());
}

public static void validateHeader(XRoadHeader header)
      throws RemoteException {
  if (header == null || header.getService() == null) {
    throw new RemoteException("Unable to read SOAP header!");
  }
  if (!header.getService().getServiceVersion().equals("v1")) {
    throw new RemoteException(
      "Unknown service version "
          + header.getService().getServiceVersion());
  }
}

public static void addClientServiceHeaders(
      WebServiceMessage message,
      XRoadHeaderProperties clientProperties,
      XRoadHeaderProperties serviceProperties,
      String serviceCode,
      String serviceVersion,
      String userId) {
  try {
    SOAPMessage soapMessage =
          ((SaajSoapMessage) message).getSaajMessage();
    SOAPHeader header = soapMessage.getSOAPHeader();
    addGeneralHeaders(header, userId);
    addClientHeaders(header, clientProperties);
    addServiceHeaders(header,
                      serviceProperties,
                      serviceCode,
                      serviceVersion);
  } catch (Exception e) {
    log.error("Exception occurred when adding X-Road headers", e);
```

```
    throw new XRoadClientException(e);
  }
}

public static void addIssueHeader(WebServiceMessage message,
                                  String issue) {

  try {
    SOAPMessage soapMessage =
          ((SaajSoapMessage) message).getSaajMessage();
    SOAPHeader header = soapMessage.getSOAPHeader();
    header.addHeaderElement(new QName(XRD_NS, "issue", "xrd"))
          .setValue(issue);
  } catch (Exception e) {
    log.error("Exception␣occurred␣when␣adding␣X-Road␣issue␣header",
              e);
    throw new XRoadClientException(e);
  }
}

private static void addGeneralHeaders(SOAPHeader header,
                                      String userId)
      throws SOAPException {
  header.addHeaderElement(new QName(XRD_NS, "userId", "xrd"))
        .setValue("EE" + userId);
  header.addHeaderElement(new QName(XRD_NS, "id", "xrd"))
        .setValue(UUID.randomUUID().toString());
  header.addHeaderElement(new QName(XRD_NS, "protocolVersion", "xrd"))
        .setValue("4.0");
}

private static void addClientHeaders(
      SOAPHeader header,
      XRoadHeaderProperties clientProperties)
          throws SOAPException {
  SOAPHeaderElement client = header.addHeaderElement(
      new QName(XRD_NS, "client", "xrd"));
  client.addAttribute(
      new QName(ID_NS, "objectType", "id"), "SUBSYSTEM");
  client.addChildElement("xRoadInstance", "id")
      .setValue(clientProperties.getInstance());
  client.addChildElement("memberClass", "id")
      .setValue(clientProperties.getMemberClass());
  client.addChildElement("memberCode", "id")
      .setValue(clientProperties.getMemberCode());
  client.addChildElement("subsystemCode", "id")
      .setValue(clientProperties.getSubsystemCode());
```

```
  }

  private static void addServiceHeaders(
        SOAPHeader header,
        XRoadHeaderProperties serviceProperties,
        String serviceCode,
        String serviceVersion)
            throws SOAPException {
    SOAPHeaderElement service = header.addHeaderElement(
        new QName(XRD_NS, "service", "xrd"));
    service.addAttribute(
        new QName(ID_NS, "objectType", "id"), "SERVICE");
    service.addChildElement("xRoadInstance", "id")
        .setValue(serviceProperties.getInstance());
    service.addChildElement("memberClass", "id")
        .setValue(serviceProperties.getMemberClass());
    service.addChildElement("memberCode", "id")
        .setValue(serviceProperties.getMemberCode());
    service.addChildElement("subsystemCode", "id")
        .setValue(serviceProperties.getSubsystemCode());
    service.addChildElement("serviceCode", "id")
        .setValue(serviceCode);
    if (serviceVersion != null) {
      service.addChildElement("serviceVersion", "id")
        .setValue(serviceVersion);
    }
  }
}
```