TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Holden Karl Hain

Keskse volituste haldamise infosüsteemiga Pääsuke liidestumine Patsiendiohutuse andmekogu näitel

Bakalaureusetöö (9 EAP)

Juhendajad: Margus Jäger, Eero Vainikko

Tartu 2024

Keskse volituste haldamise infosüsteemiga Pääsuke liidestumine Patsiendiohutuse andmekogu näitel

Lühikokkuvõte:

Käesoleva töö eesmärk oli arendada välja Eesti Patsiendiohutuse andmekogu rakenduse sisselogimisfunktsionaalsus ja seega panustada Eesti e-riigi koodivaramusse. Sisselogimise käigus kasutatakse Riigi Infosüsteemi Ameti poolt loodud keskset pääsuhaldussüsteemi Pääsuke. Töö keskendub tänapäeval tarkvaratehnikas levinud tehnoloogiatele ja arhitektuuri mustritele nagu mikroteenused. Lisaks annab töö ülevaate Pääsukese kasutamisest ja kogu sellega liidestumise protsessist käesoleva rakenduse näitel, andes ülevaate Pääsukese kasutamiseks sarnastes projektides nii teoreetilise kui ka praktilise poole pealt. Valminud rakendus täidab projektile seatud funktsionaalseid ja mittefunktsionaalseid nõudeid ning kasutab edukalt Pääsukest volituste haldamiseks.

Võtmesõnad:

Autentimine, pääsuhaldus, mikroteenused, Pääsuke, SpringBoot

CERCS: P175 Informaatika, süsteemiteooria

Integration of central authorizations management information system Pääsuke on the example of Patient Safety Database

Abstract:

The aim of this thesis was to develop the authentication and authorization functionality for the Estonian Patient Safety Database application and thereby contribute to the Estonian e-Government codebase. As part of the login process, the central access management system Pääsuke, created by the Estonian Information System Authority, is utilized. The thesis focuses on modern software engineering technologies and architectural patterns such as microservices. Additionally, it provides an overview of the usage of Pääsuke and the entire integration process using the example of the given application, offering insights into the utilization of Pääsuke in similar projects from both theoretical and practical perspectives. The developed application fulfills both functional and non-functional requirements set for the project and successfully utilizes Pääsuke for authorization management.

Keywords:

Authentication, Access control, microservices, Pääsuke, SpringBoot

CERCS: P175 Informatics, systems theory

Sisukord

Sissejuhatus

IT valdkonnas on viimastel aastakümnetel olnud näha mitmeid eri tendentse. Tänu pilveteenuste arengule ja üldisele digitaliseerimisele koguneb iga aastaga aina rohkem andmeid, mis tähendab omakorda seda, et aina rohkem ja suuremas mastaabis on tarvis hallata, kes millistele andmetele ligi pääseb [1]. Teisalt on liigutud aina hajusamate süsteemide poole, kus andmed paiknevad eri teenustes ja erinevates andmebaasides [2]. Selleks, et efektiivselt sellises keskkonnas andmetele ligipääsu hallata on töötatud välja uusi pääsuhaldusmudeleid ja lahendusi [3,1].

Üks selline hajus keskkond, kus on tihti tarvis eri andmebaasides paiknevatele olulistele andmetele ligipääsu hallata on Eesti e-riik. Sellele vajadusele keskendudes on Riigi Infosüsteemi Amet (edaspidi RIA) omalt poolt teinud olulisi samme, et analüüsida vajadusi ja pakkuda lahendust, milleks on RIA poolt arendatud keskne pääsuhalduse infosüsteem Pääsuke[1]. Pääsukese eesmärk on pakkuda keskset ja samas paindlikku lahendust ning hallata juurdepääsu volitusi ühes keskses kohas. Pääsuke on suunatud nii avaliku kui ka privaatse sektori asutustele ja selle edukus sõltub suuresti sellest kui paljud asutused sellega integreeruvad.

Töö esimene eesmärk on luua Patsiendiohutuse andmekogu (edaspidi POHAK) projekti nõuetele vastav sisselogimise lahendus ja seega anda panus Eesti e-riigi koodivaramusse. Töö tulemusena valmib POHAK rakenduse iseteeninduse keskkonda sisselogimise funktsionaalsus koos selleks vajaliku koodiga. Töö annab ülevaate tänapäeva tarkvara projektides kasutatavatest arhitektuurilistest mustritest, standardnõuetest ning tehnoloogiatest eelkõige mikroteenuste ja autentimise vaatenurgast.

Teine töö eesmärk on kirjeldada RIA uut volituste haldamise süsteemi Pääsuke ning selgitada ja dokumenteerida sellega liidestumise protsessi Patsiendiohutuse andmekogu näitel. Tööle lisab olulisust ja väärtust asjaolu, et see on üks esimesi Pääsukesega liidestuvaid rakendusi. Töös tuuakse välja kuidas sobitub Pääsuke laiemasse Patsiendiohutuse andmekogu autentimise voogu, ning antakse konkreetne näide Pääsukese X-tee liidese implementatsioonist.

Töö on jaotatud neljaks suuremaks peatükiks. Esimene peatükk annab laiema ülevaate pääsuhaldusest ning tutvustab erinevaid kasutuses olevaid mudeleid ning strateegiaid. Teine peatükk tutvustab lähemalt POHAK rakendust ja Pääsukese rolli rakenduses. Kolmas peatükk keskendub kasutatud tehnoloogiatele ja arenduse protsessile ning viimane peatükk analüüsib valminud rakendust, edasiarenduse võimalusi ning Pääsukese kasutamist projekti raames.

[1] Pääsukese koduleht: https://www.ria.ee/riigi-infosusteem/kesksed-platvormid-avalike-e-teenuste-pakkumiseks/paasuke

Töös kasutatavad mõisted ja terminid[2]

NPE (ingl *Non-person entity*) ehk digitaalidentiteediga olem, välja arvatud inimene. Selleks võib olla organisatsioon, seade, rakendusprogramm jne.

Pääsupoliitika (ingl *Access Control Policy*) ehk reeglistik, mis määratleb pääsu saamise tingimused.

Kõrgkäideldavus (ingl *high availability*) ehk kontekstist sõltuvalt kõrgematele nõuetele vastav olemi teovõime ja kättesaadavus volitatud kasutajale siis, kui see neid vajab.

Horisontaalne skaleerimine (ingl *horizontal scaling*) ehk mingi süsteemi või rakenduse viimine suuremasse mastaapi, lisades juurde arvutussõlmesid, selle asemel, et suurendada olemasolevaid sõlmesid, lisades neile mälu ja CPU tuumasid.

Pidevintegratsioon/pidevvalmidus (ingl *Continuous Integration/Continues Delivery;* CI/CD) ehk tarkvaraarenduse metoodika, mille puhul testitakse ja publitseeritakse uus kood automaatselt ja sagedasti. See annab võimaluse testida pidevalt uut funktsionaalsust, ning vältida pikki mestimisprotsesse.

Active Directory ehk Windowsi põhine andmebaas ja sellega seotud teenused, mis võimaldavad ühendada kasutajaid kindlate võrgu ressurssidega.

Ainulogimine (ingl *Single sign-on; SSO*) ehk autentimise skeem, kus kasutaja saab ühe kordse sisselogimisega ligipääsu mitmetele erinevatele, kuid omavahel seotud teenustele.

TARA ehk Eesti riigi autentimisteenus, mida haldab RIA ja kus saab autentida ID-kaardi, mobiil-ID, smart-ID ja Euroopa liidu eID kaudu.

JWT (ingl *json web token*) ehk interneti standard, mis määrab, mis vormis infot vahetada veebiteenuste vahel. Info võib olla digitembeldatud ja/või krüpteeritud, ning on JSON formaadis.

Domeenist juhinduv disain (ingl *Domain-driven disain; DDD*) ehk tarkvara disaini metoodika, mis peab oluliseks äri domeeni võimalikult hästi kirjeldamist ja sellest juhindumist

[2] Käesolevas töös on mõistete eesti keeled tõlkimisel ja lahti seletamisel on lähtutud Cybernetica andmekaitse ja infoturbe portaalist ja mõningatel juhtudel ka vallaste.ee e-teatmikust.
Cybernetica portal: https://akit.cyber.ee/
Vallaste e-teatmik: http://www.vallaste.ee/

tarkvara disainimisel. Tihti kaasatakse sellesse protsessi antud domeeni eksperte, et paremini aru saada domeeni olemusest ja probleemidest.

CVE (ingl *Common Vulnerabilities and Exposures*) ehk avalik list Informatsiooni süsteemide turvanõrkustest ja juhtumitest. Juhtumite klassifitseerimiseks kasutatakse CVSS (ingl *Common Vulnerability Scoring System*) skoori.

REST (ingl *representational state transfer*) on arhitektuuriline stiil, mis määrab kuidas peaksid veebiteenused omavahel üle HTTP (ingl *Hypertext Transfer Protocol*) protokolli suhtlema.

SOAP (ingl *simple object Access protocol*) on sõnumi protokoll, mis määrab ära, kuidas tuleks infot struktureerida. SOAP põhineb XML-il ja seda kasutakse enamasti veebiteenuste vahel suhtlemiseks.

CRUD (ingl *Create, Read, Update, Delete*) on lühend, mida kasutatakse, et kirjeldada põhilisi operatsioone loomine, lugemine, uuendamine ja kustutamine, mida on tarvis, et implementeerida mingi objekti püsivat salvestamist.

1 Teoreetiline ülevaade

Selles peatükis kirjeldatakse pääsu- ja volituste halduspõhimõtteid ja tutvustatakse erinevaid pääsuhalduse mudeleid. Seejärel antakse ülevaade RIA poolt loodud andmevahetuskihist X-tee ning sellele toetuvast uuest volituste haldamise süsteemist Pääsuke.

1.1 Pääsuhaldus

Pääsuhaldust või pääsu reguleerimist määratletakse laiemas kontekstis kui igasugust kellegi ligipääsu piiramist mingile ressursile. Kuigi pääsuhaldust ja pääsu reguleerimist kasutatakse tihti sünonüümidena viitab pääsuhaldus enamasti laiemale mõistele või üldisele protsessile, mis hõlmab õiguste haldust, pääsupoliitikate (ingl *access control policy*) läbivaatust ja muid sarnaseid tegevusi. Seevastu pääsu reguleerimine on mingi konkreetsem ligipääsu reguleeriv tegevus. Seda arvesse võttes kasutatakse edaspidi selles töös neid mõisteid nagu eelnevalt lahti seletatud.[3]

Pääsuhaldus kaitseb ressursse ja privaatset informatsiooni nende eest, kellel ei tohiks sellele ligipääsu olla. Antud kontekstis võib selleks ressurssiks olla näiteks mõni fail või kellegi andmed. Kuna digitaalseid andmeid on tänapäeval aina rohkem siis on ka üha suurem vajadus reguleerida ligipääsu nendele andmetele [4,5]. Seega on viimasel paaril kümnendil tehtud mitmeid uurimustöid just erinevate pääsupoliitikate ja mudelite kohta [6]. See on endaga kaasa toonud palju uut terminoloogiat, mis ei ole alati üheselt mõistetav. Erinevad uuringud võivad ka samu termineid kasutada erinevalt, mis tekitab veelgi rohkem segadust. Terminoloogiast ja eri pääsuhaldus strateegiatest ja mudelitest on andnud hea ülevaate Mohamed et al. enda süstemaatilises kirjanduse ülevaates [6], kus on muuhulgas pakutud välja, kuidas mõisteid ja erinevaid mudeleid liigitada selleks, et oleks lihtsam orienteeruda erinevate uuemate ja vanemate pääsuhaldus-mudelite seas. Uurimuses on välja toodud kolm mõistete taset: strateegia tase, mudeli tase ja olemi tase. Järgnevalt antakse neist tasemetest lühike ülevaade Mohamed et al. uurimusest lähtudes.

1.1.1 Strateegia tase

Pääsuhalduse strateegia annab üldise lähenemise, kuidas poliitikaid määratakse ja loob seeläbi raamistiku kogu pääsuhalduseks. Pääsuhalduse strateegiad jagunevad suures plaanis kaheks:

1. Diskretsionaarne pääsu reguleerimine (ingl *Discretionary Access Control*) edaspidi DAC ja
2. Mandatoorne pääsu reguleerimine (ingl *Mandatory Access Control*) edaspidi MAC.

DAC tähendab seda, et kasutajad reguleerivad ise ligipääsu kõigile ressurssidele, mida nad omavad. Pääsuhaldus on seeläbi n-ö omanikukeskne. MAC tähendab aga seda, et kogu pääsuhaldust kontrollib süsteemi administraator. Ehk siis kasutajatele ja ressurssidele määratakse õigused administraatori poolt. See on n-ö administratsioonikeskne pääsuhaldus.

[3] Antud mõisted on Cybernetica andmekaitse ja infoturbe portaalis antud definitsioonide alusel viidud vastavusse Mohamed et al. uuringus välja toodud ingliskeelsete mõistetega.

1.1.2 Mudeli tase

Pääsuhalduse mudel määrab, mille järgi otsustatakse, kas juurdepääs lubatakse või ei lubata. Pääsuhalduse mudelitel on tavaliselt 3 põhilist komponenti [7]:

- Subjekt- olem kellele/millele juurdepääs antakse (see võib olla kasutaja, protsess, programm)
- Objekt- objekt mille suhtes roll on määratud (see võib olla näiteks fail)
- Tegevus – määrab mida subjekt saab objektiga teha (tavaliselt näiteks kirjutada, lugeda)

Lisaks nendele põhilistele komponentidele on tihti vaja arvesse võtta ka lisakomponente olenevalt pääsuhalduse mudelist. Nendeks võivad olla näiteks rollid või keskkonna tunnused.

1.1.3 Olemi tase

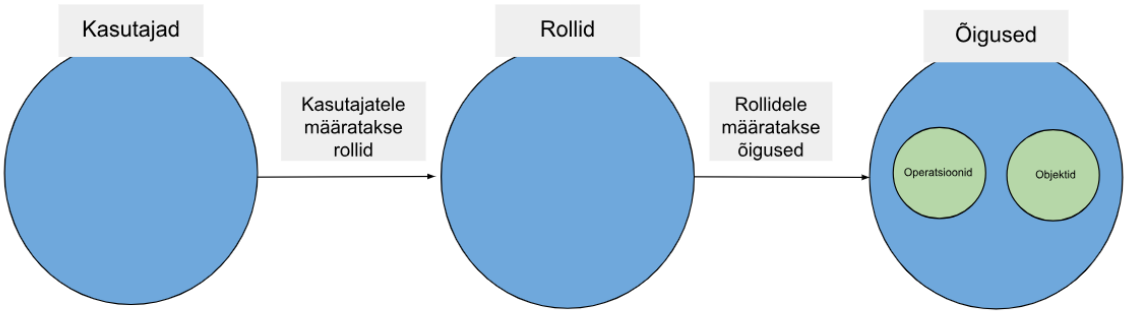
Pääsuhalduse olemid on konkreetsed pääsupoliitikad ja neid realiseerivad mehhanismid. Ehk võib öelda, et see on kõige madalam abstraktsiooni tase, kus ei ole tegu enam mudeliga, vaid konkreetse teostusega.

1.2 Pääsuhalduse mudelid

Kuigi pääsuhalduse mudeleid on palju, on valdavalt kasutuses siiski vaid enim tuntumad [6]. Sellest lähtudes antakse siin peatükis ülevaade kõige laialdasemalt levinud pääsuhalduse mudelitest ning tuuakse välja nende eelised ja puudused.

1.2.1 Rollipõhine pääsu reguleerimine

Rollipõhine pääsu reguleerimine (ingl RBAC ehk *Role-based Access Control*) on üks esimesi ja laialdasemalt levinud pääsu reguleerimise mudeleid, mis sai alguse mitme kasutaja süsteemidest 1970. a. ning millest on hea ülevaate andnud Sandhu *et. al* [8]. Rollipõhise pääsu reguleerimise peamine idee seisneb selles, et kasutajatele määratakse rollid. Rollid on omakorda seotud ka ressurssidega erinevate pääsulubade kaudu. See tähendab, et rollid on ühendav lüli objekti ja subjekti vahel nagu on näha ka joonisel 1. Näiteks võib olla kasutajale antud roll administraator ja ühtlasi on administraatoritele lubatud ligipääs teatud ressurssidele.



Joonis 1. Rollipõhine pääsuhalduse mudel

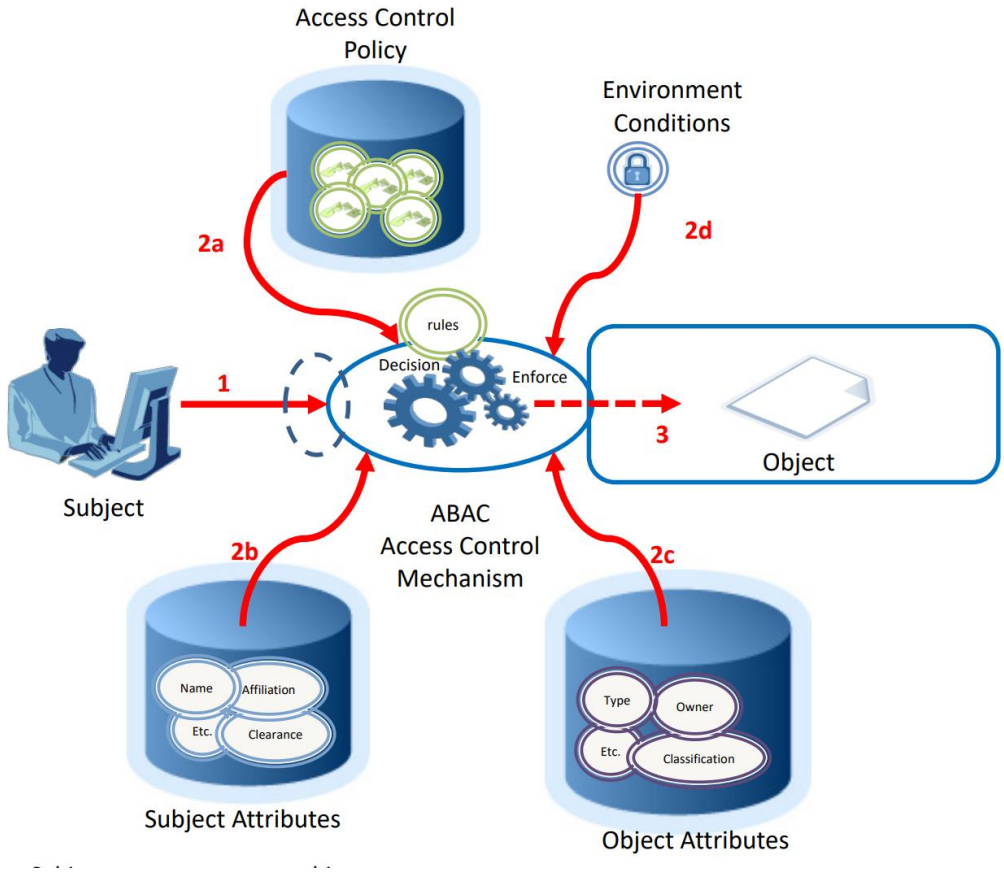
Rollipõhine pääsu reguleerimine sobib tihti erinevate organisatsioonide pääsuhalduseks, kuna rollid tulenevad enamasti loomulikult erinevatest tööülesannetest organisatsioonis. Samas on rollidega mugav hallata suurt hulka inimesi, võrreldes primitiivsema lahendusega nagu näiteks pääsuloend (ingl *access control list*), kus peaks igat kasutajat eraldi haldama.

Samas on Sandhu *et. al.* uurimustöös toodud välja ka RBAC lähenemise puudujäägid. Rollipõhise lähenemise põhiliseks piiranguks on see, et kui tahta täpsemalt määratleda õiguseid, võib kergesti tekkida väga palju erinevaid rolle, mida on keeruline hallata. Seda nimetatakse rollide plahvatuseks (ingl *role explosion*). Samamoodi võib ka mõningatel juhtudel olla keeruline modelleerida õiguste süsteemi rollipõhiselt [4,9].

1.2.2 Atribuudipõhine pääsu reguleerimine

Atribuudipõhine pääsureguleerimine on mitmeti vastus erinevatele puudustele, mis esinesid rollipõhisel lähenemisel [10]. Uuemat tüüpi rakendused nagu näiteks sotsiaalmeedia ja pilvelahendused vajavad tihtipeale ka palju täpsemat ligipääsu reguleerimist. Seda võimaldab ABAC, mida on põhjalikult kirjeldanud National Institute of Standards and Technology (NIST) enda väljaandes “Guide to Attribute Based Access Control (ABAC) Definition and Considerations “ [9]. Selle järgi seisneb ABAC-i põhiline idee selles, et nii objektidele kui ka subjektidele määratakse kindlad atribuudid ja just nende atribuutide võrdlusel langetatakse otsus, kas subjekt peaks selle objektiga mingit kindlat tegevust saama läbi viia (vt Joonis 2). See tähendab, et kui atribuudid ja poliitikad on hästi läbi mõeldud, ei ole vaja ressursi olemuse muutumisel tegeleda eri gruppidega, vaid saab hõlpsasti kas lisada või eemaldada vastava atribuudi ressursilt.

Lisaks võimaldab ABAC võtta arvesse ka keskkonnast tulenevaid muutujaid nagu näiteks kellaaeg või asukoht pääsuloa taotlemise hetkel. See kõik võimaldab lihtsamat ja täpsemat pääsuhaldust. Joonisel 2 on toodud välja põhilised ABAC mudeli komponendid ja kuidas need on üksteisega seotud.



Joonis 2. ABAC pääsuhalduse mudel [9]

Tegelikult võib väita, et RBAC on justkui ABAC mudeli alamliik, kuna rolle saab samuti kirjeldada kui kindlaid atribuute. Samas nagu Coyne on oma ABAC ja RBAC mudelite võrdluses välja toonud, pole siiski ABAC mudeli kasutamine koos rolle kirjeldavate atribuutidega praktikas hea idee [4]. Kui rollid muuta atribuutideks, kaovad sellega ka rollipõhise lähenemis eelised.

ABAC mudelil on aga samuti omad miinused. Üheks nendest on atribuutide kirjeldamine. See võib olla keeruline ja tülikas tegevus ja mõningal juhul võib olla raske keerulisi seoseid atribuutidega kirjeldada. Teiseks võib hajutatud keskkonnas olla ABAC mudeli implementeerimine üsna tülikas. Näiteks vajab objekti atribuutide kätte saamine tihti eraldi päringut, mis võib omakorda aega võtta. Mikroteenustega keskkonnas tähendab see, et infot on vaja pärida mitmetest allikatest ja kõik need allikad peavad olema kõrgkäideldavad, mis ei pruugi alati olla praktikas mõistlikult saavutatav [9].

1.2.3 Pääsuhaldusmudelite edasiarendused

Kuigi eelkirjeldatud mudelid on kaks põhilist pääsuhalduse mudelit, on tänapäeval ka mitmeid edasiarendusi ja hübriidmudeleid, mis proovivad ära kasutada mõlema mudeli eeliseid [6]. Üks viimase aja suuri edulugusid on Google'i-keskne pääsuhaldus-süsteem Zanzibar

[11]. Zanzibar kasutab ReBAC mudelit ehk (*Relationship-Based Access Control*), mis otsustab kas lubada ligipääs või mitte objekti ja subjekti vahelise seose põhjal [3,11]. Kuna Google kasutab Zanzibari, et hallata pääsu reguleerimist kõigis enda teenustes, on selge, et see mudel on hästi skaleeruv (ingl *scalable*), kiire ja paindlik, ning on inspireerinud mitmeid sarnaseid lahendusi nagu näiteks Auth0i FGA (*fine grained authorization*) [11].

Terminit ReBAC mainis esimest korda Bill Gates ja sellest saati on selle kohta tehtud mitmeid uuringuid ning aina rohkem projekte on hakanud kasutama seda mudelit [12,13]. ReBAC läheneb pääsuhaldusele veidi erinevalt, nimelt ei vasta ta küsimusele, mis rolli või atribuutide põhjal võib subjekt A teha mingit tegevust objekti B-ga, vaid vastab küsimusele, milline suhe on A ja B vahel. Lisaks võib B-l olla omakorda suhe kolmanda objektiga. Nii moodustub nö. suhete graaf mille põhjal saab teha otsuseid pääsuhalduse osas.

Siiski on see üpriski uus lähenemine pääsuhaldusele, mis sai alguse sotsiaalvõrgustike tõusuga [13].

1.3 Keskne volituste haldamise infosüsteem Pääsuke

Pääsuke on RIA ja AS Nortali koostöös loodud keskne volituste haldamise süsteem. Selle projekti põhiliseks eesmärgiks on tuua selgust avaliku sektori asutuste volituste süsteemi ja luua sellele ühine otspunkt [14].

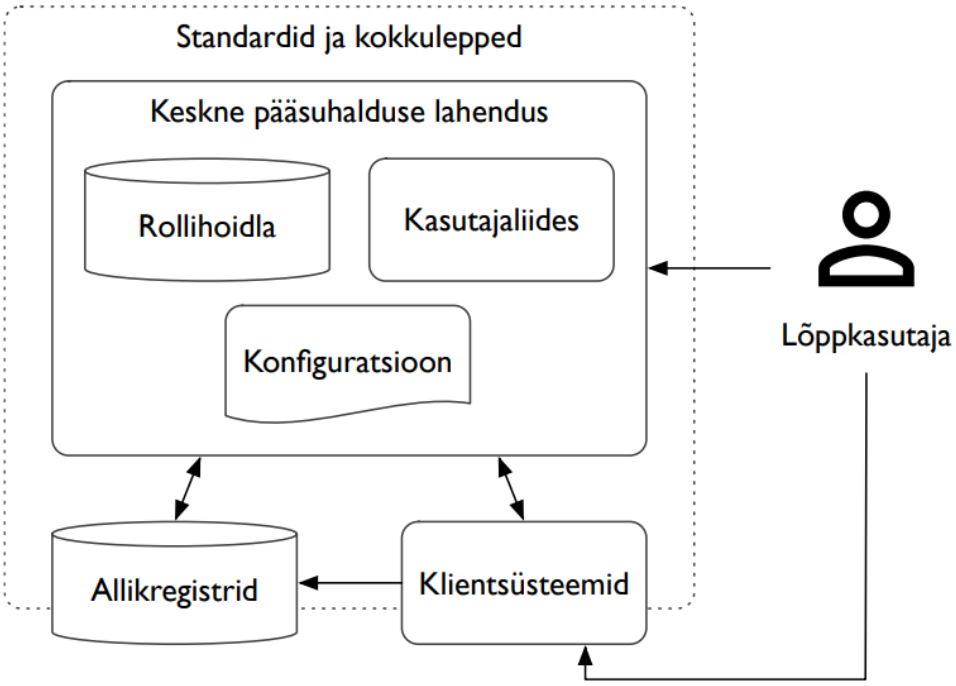
Nagu eespool välja toodud, on mitmeid eri mudeleid ja strateegiaid, kuidas pääsuhaldust korraldada ja neid üksteisega kokku viia. Eri süsteeme integreerida on tihti keeruline ja kulukas. Eesti avalikus sektoris ongi eri asutustel ka tihti erinevad pääsuhalduse süsteemid, mis teeb üleriikliku volituste haldamise keerukaks ja tülikaks [15]. Just seda probleemi proovib lahendada Pääsuke.

Järgnev selgitus Pääsukese funktsionaalsusest ja arhitektuurist pärineb RIA poolt avalikustatud analüüsi dokumentidest, mille on kirjutanud Proud Engineers OÜ [16,17].

Pääsuke on mõeldud funktsioneerima tsentraalse teenusena, mis suudaks hallata erinevaid rolle ja volitusi eri valdkondadest. Samas peab olema Pääsuke ka piisavalt paindlik, et seda oleks kerge kasutusele võtta eri asutustel ja et integreerumise keerukus ei oleks liiga suur. Seega võtab Pääsuke mingil määral eeskujuks just Google Zanzibari [3] lahendust, millel on sarnased nõuded. Google'i keskkond sarnaneb mõneti E-riigile just selle poolest, et seal on erinevaid teenuseid, millel on täiesti erinev funktsionaalsus ja visioon, kuid siiski on neil ka ühine kokkupuute punkt. Siiski on Pääsukese mudelit Eesti kontekstile kohandatud näiteks allikregistrite mõistega. Seega kasutab ka Pääsuke ReBAC pääsuhaldusmudelit, ning põhilisteks olemiteks ongi kahe objekti omavahelist seost kirjeldavad kolmikud, mis koosnevad objektist, seosest ja subjektist.

Toetudes ülaltoodud nõuetele, ei ole Pääsuke täielikult tsentraalne lahendus ega ka täielikult hajus, vaid midagi vahepealset, mida võiks võrrelda keskselt toestatud võrgustikuga. See koosneb kesksest rakendusest ja kasutajaliidesest, mis on liidestunud mitmete erinevate

allikregistrite[4] ja klientsüsteemidega. Klientsüsteemid saavad X-tee kaudu pärida Pääsukesest välja seoseid või siis kirjeldada enda otspunktid Pääsukese jaoks, et Pääsuke saaks ise pärida kliendi enda andmebaasist seoseid ning neid muuta nagu on välja toodud joonisel 3. Seejärel saab lõppkasutaja näha enda volitusi Pääsukese enda kasutajaliidesest ja neid kas seal samas muuta või siis muuta liikudes edasi kliendi rakendusse.



Joonis 3. Pääsukese arhitektuur

Siinkohal võib ka mõiste „seos“ kohta kasutada sõna „roll“. Näiteks Objektil A on roll/seos „juhatuse liige“ subjekti B suhtes. Siiski peab silmas pidama, et mõiste roll tähendab sisuliselt siiski siinkohal A ja B vahelist suhet või seost ja ei ole roll RBAC mõistes. Pääsuke kasutab erinevaid nimeruume eri asutuste rollide eristamiseks. Seetõttu on seosed/rollid Pääsukeses kujul `nimeruum:roll`. See võimaldab kiirelt leida rolle, mis on just mingi kindla nimeruumiga seoses.

Liidestamise mudeleid on erinevaid ja klientsüsteemid saavad ise otsustada, kuidas või millisel määral nad Pääsukesega liidestuda tahavad. Maksimaalselt integreerudes on võimalus lasta kõiki rolle hallata Pääsukesel endal, ja enda rakenduses lihtsalt neid Pääsukesest küsida. Teisalt võib ka kõiki rolle hoida enda infosüsteemis ning samuti nende haldust teha enda rakenduses. Sellisel juhul tuleks vaid defineerida liides, et Pääsuke saaks neid rolle pärida ja ka enda kasutajaliidesest kuvada. Kokkuvõttes tagab selline arhitektuur paindlikkuse erinevate kliendi soovide suhtes, tagades samal ajal mugavuse kasutajale.

[4] Allikregistrid on siin kontekstis andmekogud, mille ülesanne on jagada õiguslikult või funktsionaalselt autoriteetset infot volituste kohta. Näiteks, vaid äriregister on võimeline väljastama pädevat infot juhatuse liikmete rollide kohta.

1.4 Andmevahetuskiht X-tee

X-tee on RIA poolt hallatav turvaline andmevahetuskiht, mida kasutatakse Eestis, et turvaliselt riigi andmeid organisatsioonide vahel vahetada [18]. X-tee on osa laialdasemast tehnoloogiast X-road, mida arendab NIST (Nordic Institute for Interoperability Solutions), ning aastast 2018 kutsutakse Eesti X-road tehnoloogiat ka inglise keeles X-tee. Projektil on nii avatud lähtekood kui ka dokumentatsioon, millele tuginedes tuuakse siin lühikirjeldus X-tee peamistest põhimõtetest ja toimimisest. [19,18].

X-tee koosneb põhiliselt tsentraalsest serverist ja mingist hulgast Turvaserveritest. Igal kliendil, kes soovib X-teega ühineda on vaja enda turvaserverit, kuigi on võimalik, et mitu klienti kasutab jagatud turvaserverit. Tsentraalne server hoiab kõikide turvaserverite infot ja jagab seda perioodiliselt kõikide teiste turvaserveritega. Turvaserverid ise hoolitsevad selle eest, et saata ja võtta vastu sõnumeid teistest turvaserveritest. Selleks luuakse otseühendus turvaserverite vahel läbi avaliku interneti, kuid kasutades avaliku võtme krüptograafiat, et ühendus oleks turvaline. X-tee kasutab nii ajatembeldus teenust kui ka e-templit (digiallkirja analoog) et tagada info terviklikkus ja konfidentsiaalsus.

Selline arhitektuur tagab info turvalise vahetuse. Süsteem on hajutatud ning ei ole ühtegi kriitilist punkti, millest infovahetus sõltuks. Näiteks ei ole vajalik pidev ühendus tsentraalse serveriga. Samuti saavad kliendid ise määrata täpselt, millist infot kellega nad soovivad jagada ja saavad seega ise enda infosüsteemi käideldavuse tagada. Ühendus luuakse alati krüpteeritult otse klientide vahel ja seega ei ole ohtu kolmanda osapoole tõttu (andmed ei liigu kordagi läbi tsentraalse serveri). See kõik tagab info käideldavuse, terviklikkuse ja konfidentsiaalsuse.

X-teega liitumise protsessi haldab RIA ja liitumine nõuab eelnevalt kokkulepet RIA-ga. Selleks, et X-teega liituda tuleks sooritada järgnevad sammud:

1. hakata X-tee liikmeks see tähendab sõlmida kokkulepe RIA-ga,
2. Paigaldada või rentida turvaserver,
3. Sõlmida sobiva X-tee teenuse osutajaga kokkuleppe,
4. Töötada välja loogika soovitud teenuse sisendandmete loomiseks või vastuse töötluseks olenevalt sellest, kas soovitakse ainult infot tarbida või ka edastada.

X-teega liitumise täpsem protsess ja lisainfo on leitav RIA kodulehel[5].

[5] X-teega liitumise info: https://www.ria.ee/riigi-infosusteem/andmevahetuse-platvormid/andmevahetuskiht-x-tee#turvaline

2 Patsiendiohutuse andmekogu

Selles peatükis kirjeldatakse patsiendiohutuse andmekogu rakendust lähemalt. Antakse ülevaade nii rakenduse arhitektuurist kui ka tehnilistest nõuetest, lähtudes sisselogimise perspektiivist. Samuti selgitatakse lähemalt Pääsukese rolli kogu sisselogimise voos.

Patsiendiohutus on tervishoiuteenuse osutamisega kaasneva välditava tervisekahju riski eesmärgipärane vähendamine. Seni puudus Eestis tervishoiuteenuse osutajatel ühtne standard, mille alusel patsiendiohutusjuhtumeid registreerida ning nende järgi ühtset statistikat teha. Eesti patsiendiohutusjuhtumite esinemine ei ole teada ja peamine põhjus on ühtse raporteerimise standardi puudus. Siiani registreeris ja analüüsis patsiendiohutusjuhtumeid iga tervishoiuteenuse osutaja (edaspidi TTO) eraldi, kasutades selleks oma protsessi ja oma infosüsteeme. Suuremad haiglad nagu TÜK ja PERH teevad seda küll vastavalt standardile, kuid mõlemad oma väljatöötatud protsessi alusel, seevastu paljudes väiksemates TTO-des on see protsess puudulik[6].

Patsiendiohutuse andmekogu eesmärk on luua patsiendiohutusjuhtumite kogumiseks ja töötlemiseks keskne standardiseeritud infosüsteem ja keskkond, mille kaudu kõik TTO-d hakkavad edastama andmeid patsiendiohutusjuhtumite kohta.

Patsiendiohutuse andmekogu aluseks on Tervishoiuteenuse osutaja kohustusliku vastutuskindlustuse seadus, mis jõustub 01.07.2024, ning sätestab POHAK süsteemi nõuded.

POHAK peab võimaldama minimaalselt [20]:

- Patsiendiohutusjuhtumite lihtsat ja vähese ajakuluga dokumenteerimist toetades seda vajalike loendite ja/või klassifikaatoritega,
- Kõikidel TTO-del liidestuda loodava POHAK rakendusega võimalikult lihtsalt ja minimaalsete kuludega,
- Väikestel TTO-del sisestada andmeid läbi TTO iseteeninduse portaali.

Loodava patsiendiohutuse andmekogu valmimisel on kohustus sellega liituda või esitada sinna andmeid läbi iseteeninduse portaali kõigil tervishoiuteenuse tegevusluba omavatel juriidilistel isikutel sõltumata tüübist ning rahastamise allikatest.

POHAK rakenduses hakkab olema kaks keskkonda ametniku keskkond ja iseteeninduse keskkond. Ametniku keskkond on mõeldud Tervise ja Heaolu Infosüsteemide Keskuse (edaspidid TEHIK) ja Terviseameti töötajatele, et hallata süsteemi POHAK ja sealolevaid patsiendiohutuse juhtumeid. Iseteenindus keskkond on mõeldud TTO-dele, kellel puudub oma süsteem, mis on liidestunud POHAK rakendusega. Selle kaudu saavad TTO töötajad logida iseteeninduskeskkonda sisse, kasutades id-kaarti või mobiil-ID-d ning sisestada ja vaadata selle kaudu patsiendiohutuse juhtumeid.

[6] Info pärineb TEHIK-u wikist, mis ei ole avalik. Rohkem infot https://wiki.sm.ee/

2.1 Arhitektuur ja autentimise lahendus

Selles peatükis tutvustatakse rakenduse POHAK arhitektuuri ning tuuakse välja selle olulisemad aspektid seoses sisselogimisega. Samuti antakse ülevaade sisselogimise voost, ning sellega seonduvatest arhitektuurilistest otsustest.

2.1.1 Mikroteenused

Rakendus POHAK põhineb mikroteenuste arhitektuuril. Seega koosneb rakenduse loogiline osa erinevatest väikestest teenustest ehk mikroteenustest, mis vastutavad igaüks oma kindla ülesande eest.

Mikroteenused on viimastel aastakümnetel väga palju populaarsust kogunud arhitektuuri paradigma ning on leidnud laialdast kasutust ettevõtte rakendustes [2]. He Zhang *et. al*. on oma kokkuvõtlikus töös mikroteenuste kohta toonud välja selle paradigma põhilised plussid ja miinused ning selle alusel antakse ka järgnevalt lühike ülevaade mikroteenustest [1].

Mikroteenuste põhiline idee seisneb selles, et selle asemel, et arendada ja hallata ühte suurt rakendust, on kergem korraga arendada ja hallata mitmeid väiksemaid teenuseid, mis küll suhtlevad üksteisega, kuid ei sõltu täielikult üksteisest. See toob endaga üldjuhul kaasa mitmed järgnevad plussid:

1. **Lihtsam rakenduse või rakenduse osade skaleerimine** ehk mikroteenustel põhinevat rakendust on kergem horisontaalselt skaleerida, kuna on võimalik tekitada juurde instantse just sellest rakenduse osast, mis on kõige suurema koormuse all.
2. **Kergem arendada** ehk mitu tiimi saab samaaegselt töötada mitme erineva teenuse arendamisega ilma üksteist segamata ja valides just need tööriistad, mis on neile sobivaimad.
3. **Kergem hallata** ehk mitut väiksemat projekti on kergem hallata ja hoomata kui ühte suurt koodibaasi.
4. **Kergem tarnida ja uuendada** ehk kergem on teostada pidevat integratsiooni ja pidevat valmidust (ingl *Continues Integration*, *Continues Delivery*), kuna iga uue värskenduse jaoks ei pea testima kogu suurt rakendust, vaid ainult kindlat osa sellest.
5. **Kõrgem robustsus** ehk mikroteenustel põhinev arhitektuur on enamasti robustsem ja kõrgema veataluvusega, kuna ühe teenuse või teenuse instantsi ebaõnnestumisel saavad teised teenused jätkata enda tööd.

Samuti on He Zhangi töös välja toodud ka mikroteenuste miinuseid ning juhitud tähelepanu sellele, et praktikas võib mikroteenuste arendamine olla tülikam ja kulukam, kui see oleks monoliitse rakenduse puhul. See on eriti tõsi, kui rakenduse skoop on piisavalt väike. Põhilisteks väljatoodud miinusteks on:

1. Lisakeerukus ehk mitmete teenuste omavahelise koostöö ja suhtluse implementeerimine võib osutuda keerukamaks ja kulukamaks kui monoliitse arhitektuuri puhul.
2. Kogu rakenduse testimine ja paigaldamine on keerukas, kuna on tarvis jooksutada kõiki teenuseid ja tihti ka mitmeid abiteenuseid nagu erinevaid andmebaase ning suhtlusmaaklereid.

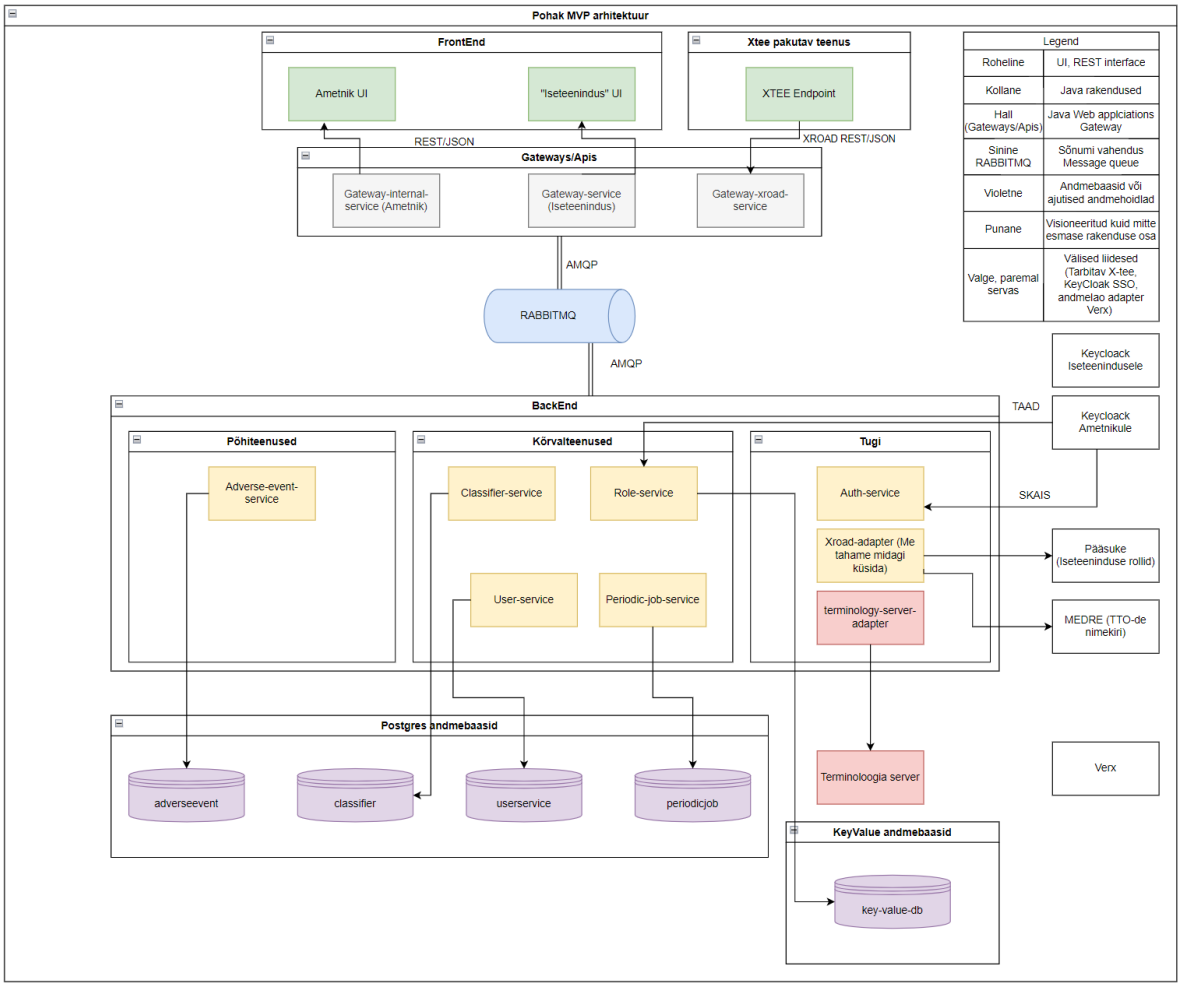
Rakenduse POHAK puhul on mikroteenustel põhinev arhitektuur üheks TEHIK-u poolt tulevaks mittefunktsionaalseks nõudeks ja seda just sellest tuleneva kergema skaleerimise ja kõrgema robustsuse ja seega turvalisuse tõttu.

2.1.2 *Common-lib* teenus

Kuigi POHAK rakenduse eraldiseisev funktsionaalsus on jaotatud eri mikroteenuste vahel, on siiski tihti vaja ka jagada mingit funktsionaalsust või kindlaid klasse, et neid ei peaks igasse teenusesse ümber kopeerima. Selleks, et teatud osa koodist jagada, kasutatakse POHAK projektis *Git submodules* funktsionaalsust. Seda kasutades on võimalik importida ühte projekti alammooduleid, mida saab ülemprojektis kasutada, kuid mille versioone haldab Git eraldi. See lubab teha kergelt muudatusi nii *common—lib* projektis kui ka teenuses, mille kallal hetkel töötatakse. Samuti saavad kõik tiimi liikmed alati kähku ühises koodis tehtud muudatustele ligi, ilma mingit teeki uuendamata, vaid tõmmates Giti kaudu alla lihtsalt kõige uuemad muudatused.

2.1.3 Üldine arhitektuur

Nagu joonisel 4 näha, koosneb POHAK rakendus kahest eraldi kasutajaliidesest koos lüüsi rakendustega (ingl *gateway*), mis teenindavad kahte eri keskkonda, ning mingist hulgast tagumistest teenustest, mis haldavad rakenduse loogikat. Lisaks sellele on ka mitmeid väliseid teenuseid, mida kasutatakse nagu näiteks KeyCloak server.



Joonis 4. POHAK rakenduse arhitektuur

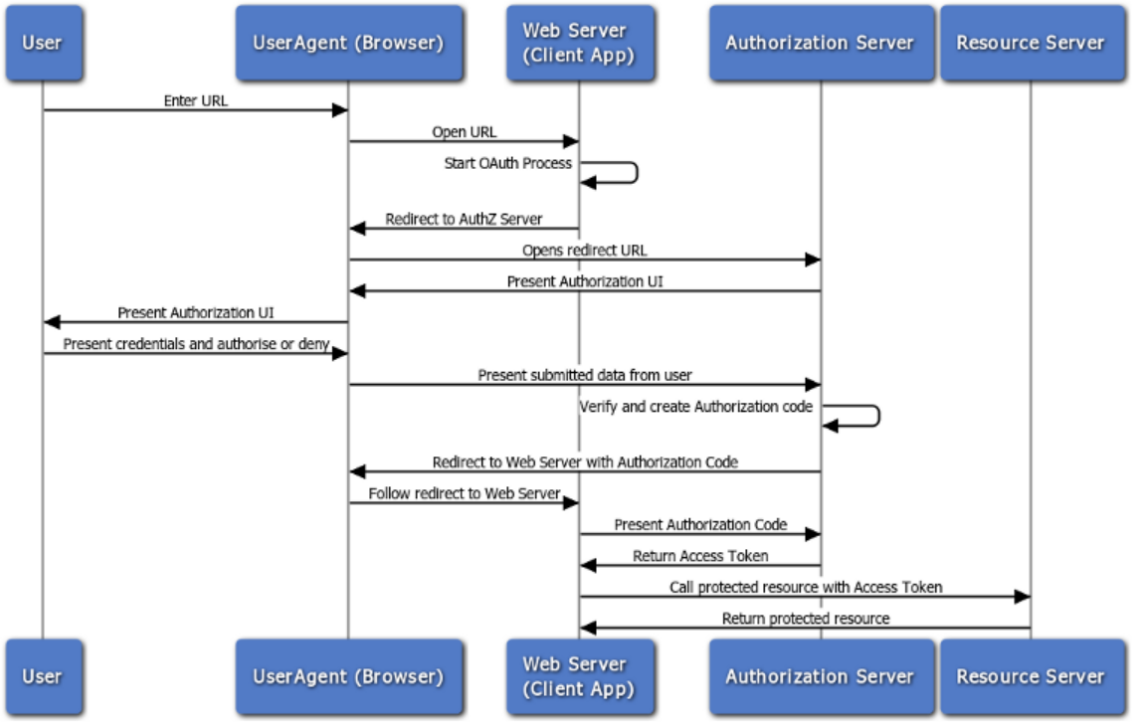
Kui vaadata arhitektuuri kihtidena, siis koosneb kõige kasutajapoolsem kiht kahest eraldi kasutajaliidesest. Sealt järgmine kiht koosneb n-ö lüüsirakendustest, mis haldavad kasutajaliidesest tulevaid HTTP-päringuid. Järgmiseks kihiks on sõnumi maakler, mis haldab nii kasutajaliidese ja tagumiste teenuste kui ka tagumiste teenuste omavahelist suhtlust. Kõige tagumine kiht koosnebki eri teenustest, mis vastutavad igaüks oma loogilise ülesande eest. Lisaks sellele on igal teenusel, mis peab salvestama mingeid andmeid, ka enda andmebaas.

2.1.4 Sisselogimise voog ja arhitektuur

Kuna POHAK rakenduses on kaks eri keskkonda, on ka mõlemal keskkonnal oma sisselogimine. Ametniku keskkonda sisselogimine toimub TEHIKu *Active Directory* kaudu. Iseteeninduskeskkonda sisselogimine on aga keerukam ja hõlmab endast mitmete päringute teostamist, et teha kindlaks kasutaja rollid. Just selles osas kasutatakse Pääsukest ning sellele osale keskendub ka antud töö.

Laiemas plaanis põhineb POHAK rakenduse iseteeninduse sisselogimise voog Oauth 2.0 protokollil, kus autentimisserveriks on TEHIKu KeyCloak ning ressursiserveriks ja klientrakenduseks on POHAK rakendus. Oauth on tänapäeval standardiks saanud protokoll, mida

kasutatakse pääsuhalduseks [21]. Oauth määrab ära, kuidas turvaliselt ressurssidele ligipääsu küsimist korraldada, pakkudes selleks mitu erinevat varianti, mida saab kasutada olenevalt rakendusest ja vajadusest. Veebirakenduste puhul on neist kõige levinum „*Authorization Code Grant Type*“, millel põhineb ka kirjeldatav sisselogimine. Ülevaade antud voost on näha joonisel 5. Nagu jooniselt näha siis suunab klient rakendus kasutaja ümber autentimisserverisse autentimiseks. Eduka autentimise tulemusena tagastatakse kasutajale autentimis kood (ingl *authorization code*). Seda kasutades, saab klient rakendus pärida selle koodi järgi kasutaja jaoks vajalikud märgised (ingl token). Selle jaoks peab suutma klientrakendus turvaliselt hoida autentimisserveri poolt tema joaks genereeritud võtit, mida kasutatakse serverite vaheliseks suhtluseks ja mis saadetakse iga märgiste päringuga kaasa. Kui rakendusel ei ole võimalik turvaliselt infot hoida (näiteks mobiilirakenduste puhul) siis ei tohiks kasutada antud voogu.

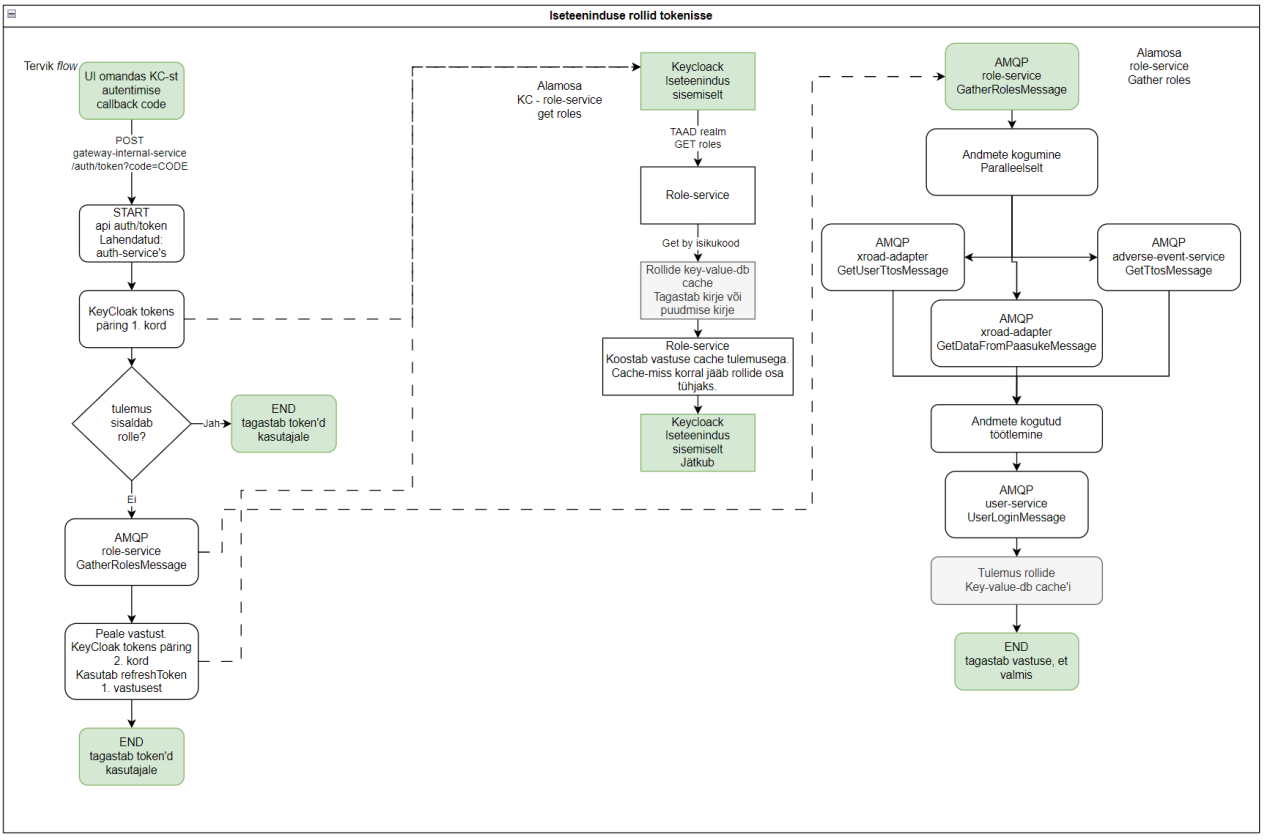


Joonis 5. Oauhti *Authorization Code flow* [22]

Iseteeninduse rollid on kujul „roll_registrikood“, kus roll on kas „TTO_sisestaja“ või „TTO_peakasutaja“ ja registrikood on antud TTO registrikood. Rollid tulevad kahest välisest teenusest. Esiteks tehakse päring tervishoiukorralduse infosüsteemi MEDRE, mis hoiustab tervishoiu töötajate andmeid. Teiseks on tarvis teha päring Pääsukesse selleks, et saada teada, ega kasutajal ei ole lisaks veel mingeid volitusi. Kolmandaks on vaja teha päring veel ohutusjuhtumi teenuse poole (joonisel 4 *adverse-evente service*), et pärida hetkel aktiivsed TTO-d ning valideerida rolle nende vastu. Oluline on veel märkida, et selleks, et saada mingi TTO peakasutaja rolli, on tarvis Pääsukeses anda selleks eraldi volitus.

Iseteeninduse autentimise voog koosneb kolmest osast. Esiteks tehakse Kasutajaliidesest sisselogimise päring, millele vastatakse ainulogimise (ingl *single sign-on*) URL lingiga. Seejärel tehakse teiseks selle lingi järgi päring KeyCloak serverisse, mis omakorda autendib kasutaja TARA kaudu ning tagastab genereeritud koodi, millega on võimalik märgiseid pärida. Viimaks teeb kasutajaliides kolmanda päringu */auth/tokens* otspunkti pihta. Selle päringu terviklik voog on näha joonisel 6 ja selle tulemusena tagastatakse kasutajaliidesele kolm JWT märgist: pääsu märgis (ingl *access token*), värskendus märgis (ingl *refresh token*) ja id märgis (ingl *id token*). Pääsu märgis sisaldab endas ka kasutaja rolle.

Selleks, et rikastada KeyCloaki poolt väljastatud märgiseid rollidega, on tarvis KeyCloakil pärida rollid rolli teenuselt (joonisel 4 *Role-service*). Esmane plaan selleks oligi teha päring rolli teenuse pihta, mis oleks omakorda teinud kaks päringut X-tee liidese teenuse (joonisel 4 *Xroad-adapter*) pihta, pannud kokku rollide info, ning tagastanud selle KeyCloakile. Turvanõuete tõttu ei tohi aga KeyCloaki päring võtta rohkem aega kui viis sekundit. Seda ei saanud siiski algse arhitektuuri puhul garanteerida kuna selle aja sees oli vaja sooritada 3 X-tee päringut (Pääsukese pihta on tarvis sooritada 2 päringut) ja üks andmebaasi päring. Seega pidi arhitektuuri muutma ning tooma sisse kohaliku puhvri, et rolle kiiremini kätte saada.



Joonis 6. Iseteeninduse sisselogimise voog

Täiustatud arhitektuur ja voog on näha joonisel 6 ning hõlmab endas järgmisi samme:

1. Sooritatakse KeyCloak päring, kus küsitakse rolle varem omandatud koodi alusel. KeyCloak pärib rolle rolli teenuselt, mis omakorda proovib leida kasutaja rolle puhvrist. Kui see ebaõnnestub tagastatakse tühi rollide list, ning Keycloak tagastab autentimis teenusele märgised, kus sees puudub rollide info.
2. Autentimis teenus kontrollib, kas märgises leiduvad rollid. Kui ei, siis sooritab autentimis teenus päringu „*GatherRoles*“ rolli teenuse pihta.
3. Rolli teenus kasutab omakorda X-tee adapteri teenust, et koguda kokku rollid. Seejärel saadetakse kasutaja teenusele (joonisel 4 *user-service*) sisselogimis sõnumi, ning rollid salvestatakse puhvrisse.
4. Autentimis teenus sooritab teise päringu Keycloaki suunal, kasutades seekord värskendus märgist.
5. KeyCloak küsib teist korda rolle rolli teenuselt, mis peaks nüüd leidma rollid puhvrist ning tagastama KeyCloakile.
6. Märgised, mis sisaldavad rolle tagastatakse Kasutajaliidesele.

Sellise voo puhul on võimalik garanteerida, et KeyCloak saab rollid kätte piisavalt kähku kuna tema päringuks on juba rollid eelnevalt puhvrisse salvestatud.

2.2 Tehnilised nõuded

Kuna POHAK rakendus on TEHIK-u tellimus, siis kehtivad sellele ka kõik TEHIK-u mittefunktsionaalsed nõuded. See hõlmab endast ka vastavust Digiriigi ristfunktsionaalsetele nõuetele, mis on mõeldud olema aluseks kõigile avaliku sektori IT arendustele ja on koostatud Digiriigi Arhitektuurinõukogu poolt. Lisaks sellele peab rakendus läbima ka OWASP ASVS (vt selgitust peatükis 2.2.4) turvatestid. Selles peatükis tuuakse välja olulisemad mittefunktsionaalsed nõuded, mikroteenuste ja autentimise vaatest toetudes TEHIK-u dokumentatsioonile, mis on avalikult kättesaadav [23].

2.2.1 Arhitektuurilised nõuded

Järgnevalt on välja toodud olulisemad arhitektuurilised nõuded POHAK projektile:

1. Lahenduse arhitektuuris kasutatakse domeenist juhitud disaini (ingl *domain driven design*) ja mikroteenuste põhimõtteid.
2. Rakenduse tehnilised hajuskomponendid ei jaga sama andmebaasi, mälu ega failisüsteemi.
3. Rakenduse, andmebaasi ja kolmanda osapoole komponendid peavad olema sellised, mille turbe uuenduste eluea lõpp pole teadaolevalt vähem kui 2 aasta pärast.
4. Rakenduse komponentide konfiguratsiooni peab olema võimalik ette anda käivitamisel.
5. Rakendus ei tohi luua uut identiteedisüsteemi. Tuleb tugineda olemasolevatele riiklikele (ID-kaart) või põhiliste op-süsteemide süsteemidele.
6. Eelistada tuleb tsentraalseid autentimislahendusi (nt Tellija SSO lahendus).
7. Infosüsteemide vaheline andmevahetus toimub üle X-tee.

Arhitektuuriliste nõuete täitmist jälgib enamasti rakenduse arhitekt ning seega kooskõlastati tehnoloogiate valikut ja arhitektuurilisi otsuseid projekti arhitektiga.

2.2.2 Nõuded lähtekoodile

Järgnevalt on välja toodud olulisemad mittefunktsionaalsed nõuded lähtekoodile:

1. Rakenduse lähtekood ja kommentaarid peavad olema inglise keeles. Rakenduse ärilised muutujad aga eesti keeles, kui neile pole mõlemapoolset loogilist vastet.
2. Rakenduse kood peab olema piisavalt hästi dokumenteeritud, et erialast haridust omav tarkvaraarendaja on võimeline süsteemile jätkuarendusi teostama. Kui muutuja nimetus vajab kommentaari, siis pigem muuta muutuja nimetust kommenteerimise asemel.
3. Koodis kasutatavaid konstante ei tohi selle kasutamise kohta väärtusena *hardcode*'da – need tuleb defineerida muutujatena ja kasutada läbi nende.
4. Rakenduse kood peab olema kirjutatud vastavalt Google'i stiili juhendile[7].
5. Arendamisel kasutatakse DRY (ingl *Don't Reapeat Yourself*) ja SOLID (vt selgitust peatükis 2.2.3) printsiipe.
6. Üleantavas koodis ei tohi olla komponente, mille CVSS punktid on 7 ja kõrgemad ning CVE on rakendatav.

Lähtekoodi nõuete eest vastutab põhiliselt arendaja ja nende tagamiseks kasutatakse nii SonarQube tarkvara (vt selgitust peatükis 2.2.5) kui ka manuaalset koodi ülevaatust.

2.2.3 DRY ja SOLID põhimõtted

Üks POHAK rakenduse mittefunktsionaalsetest nõuetest on, et kood peaks järgima DRY ja SOLID printsiipe. DRY ja SOLID on kaks tarkvaratehnikas tuntud ja viimasel ajal populaarsust kogunud printsiipi, mis proovivad anda juhiseid, et parandada tarkvara arhitektuuri ja koodi kvaliteeti [24]. Järgnevalt antakse lühike ülevaade eelnimetatud kahest printsiibist.

SOLID on akronüüm viiele objekt-orienteeritud programmeerimisekeele printsiibile, mida mainis esimest korda Robert C. Martin [25]. Kuigi mõningaid printsiipe tunti juba varem, on need kokku võtnud Robert C. Martin oma töös „Design Principles and Design Patterns", mille põhjal antakse neist kiire ülevaade.

SOLID koosneb R. C. Martini järgi järgnevast viiest printsiibist [26]:

1. „*Single-responsibility principle*" ehk igal komponendil või klassil peaks olema üks kindel vastutus ja seega ainult üks põhjus muutuda.
2. „*Open-closed principle*" ehk kõik tarkvara komponendid peaks olema avatud laiendusele (kergesti laiendatavad) aga kinnised muudatustele.

[7] Google'i stiili juhised on saadval järgmisel addressil: https://google.github.io/styleguide/

3. „*Liskov substitution principle*“ ehk iga alamklass peaks katma vähemalt sama funktsionaalsust, mis tema ülemklass ja ülemklass peaks seega olema vahetatav kõigi oma alamklassidega.
4. „*Interface segregation principle*“ ehk klasside liidesed peaksid olema funktsionaalsusest lähtudes lahutatud, nii et ükski klass ei peaks täitma liidese meetodeid, mida tal tegelikult ei ole vaja täita.
5. „*Dependency Inversion principle*“ ehk tarkvara komponendid ei tohiks sõltuda otseselt mingist muust komponendist, vaid pigem abstraktsioonist. See lubab muuta komponente, ilma sõltuvusi lõhkumata.

DRY printsiip on teine laialdaselt mainitud printsiip, mida tutvustati esmalt Andrew Hunti ja David Thomasi raamatus „The Pragmatic Programmer“ [27]. Selle printsiibi põhiline idee on lihtne: kood ei tohiks sisaldada liigset kordust. Printsiibi järgimisega kaasneb aga ka enamasti see, et kood on paremini struktureeritud ja igal komponendil tekib oma kindel vastutus. Lisaks sellel on ka koodibaas lühem ja kergemini hallatavam.

2.2.4 Turvanõuded

Lisaks eelnevalt mainitud mitte funktsionaalsetele nõuetele on oluline nõue rakendusel läbida ka OWASP ASVS turvatestid.

OWASP (ingl *The Open Worldwide Application Security Project*) on veebi kommuun, mis on juhitud mittetulundusühingu OWASP Foundationi poolt. OWASP töötab välja tasuta kättesaadavaid artikleid, metodoloogiaid, dokumente ja tööriistu tarkvara ja veebiteenuste turvalisuse parandamiseks.

Üks OWASP-i standarditest on ASVS (ingl *Application Security Verification Standard*), mis on protokoll veebirakenduste turvatestimiseks [28]. Selle eesmärgiks on anda baas turvatestideks, mis veenduvad veebirakenduse tehnilises turvalisuses ja tõrketaluvuses erinevate tuntud rünnete nagu SQL süstide ja murdskriptimise (ingl *cross-site scripting*) vastu. Seega saab seda kasutada nii turvalisuse mõõtevahendina kui ka dokumendina, millest juhinduda tarkvara disainimisel ja arendamisel.

OWASP ASVS standardil on 3 järgnevat turvataset:

1. Esimene tase on madala kindlusega ja on ainus tase, mis on läbistustestimis meetodil (ingl *penetration testing*) testitav. See tähendab, et testimiseks ei ole vaja lähtekoodi ega dokumentatsiooni.
2. Teine tase on mõeldud rakendustele, mis sisaldavad tundlikuid andmeid, mis vajavad kaitsmist. See on soovituslik tase enamus rakenduste jaoks
3. Kolmas tase on kõige kriitilisemateks rakendusteks, mis teostavad kõrge väärtusega transaktsioone või vajavad kõrgeimat usalduse taset.

POHAK rakendus peab vastama vähemalt teisele tasemele, ja selleks teostatakse eraldi turvatestimine peale rakenduse valmimist.

2.2.5 Testimine ja SonarQube

POHAK rakenduse üheks oluliseks nõudeks on ka koodi automaat-testidega kaetavus vähemalt 75% ulatuses. See tähendab, et vähemalt 75% koodist peab saama kaetud automaatsete ühik- või integratsioonitestidega. Lisaks sellele teostatakse veel eraldi läbivtestimine (ingl *e2e testing*), koormustestimine ning robustsus testimine[8].

Selleks, et tagada mittefunktsionaalsete nõuete täitmist, kasutatakse SonarQube tarkvara, mis on vabavaraline ning avatud lähtekoodiga tarkvara (eksisteerib ka tasuline versioon), et analüüsida koodi kvaliteeti ning testidega kaetavust.

SonarQube-i suureks eeliseks on see, et seda on lihtne integreerida enda pideva integratsiooni konveieriga (ingl *CI/CD pipeline*) ning see analüüsib koodi mitmete eri omaduste poolest. SonarQube analüüsib aspekte nagu:

1. Testidega kaetavus ehk kui suur osa koodist on kaetud automaatsete testidega.
2. Koodi duplitseerimine ehk kui suur osa koodist on kuskil duplitseeritud.
3. Halvad koodi tavad (ingl *code smells*) ehk kas kood sisaldab halbu tavasid.
4. Turvanõrkused ehk kas kood sisaldab turvanõrkusi.
5. Vead ehk kas koodis leidub vigu.

Lisaks SonarQube-le toimub ka pidev asutusesisene testimine kõigi uute muudatuste osas ning koodi ülevaatamine.

[8] Lisa infot testimise ja erinevate testimis meetodite kohta saab õpikust “Introduction to software testing” https://assets.cambridge.org/97805218/80381/frontmatter/9780521880381_frontmatter.pdf ISBN: 978-0-521-88038-1

3 Protsess

See peatükk kirjeldab tarkvara arenduses kasutatavaid tehnoloogiaid, nende valikut ning üleüldist arenduse protsessi ning arhitektuurilisi valikuid. Lisaks sellele antakse ülevaade kokkulepetest ja lubadest, mis on vajalikud Pääsukese kasutamisel, samuti kirjeldatakse Pääsukesega liidestumise protsessi.

3.1 Kasutatud tehnoloogiad

See peatükk kirjeldab erinevaid tehnoloogiaid, mida on selle töö raames POHAK projektis kasutatud, ning põhjendab nende valikut. Turvalisuse ja avatuse eesmärgil eelistatakse POHAK projektis võimalusel avatud lähtekoodiga tehnoloogiaid.

3.1.1 Java

Java on kompileeruv programmeerimiskeel, mille avalikustas 1995. aastal ettevõte Sun Microsystems [29]. See on objekt-orienteeritud programmeerimiskeel, mille kasutusalad on väga laialdased, hõlmates endas ettevõtterakendusi, mobiilirakendusi, mänge, veebiteenuseid, mängukonsoole ja meditsiiniriistvara [30]. Java põhineb paljuski C ja C++ keelel ja sellel on sarnane süntaks. Javat tuntakse tema kiiruse, turvalisuse ja usaldusväärsuse poolest ning see on viimased 20 aastat olnud üks populaarsemaid programmeerimiskeeli ning on seda ka praegu [31].

3.1.2 Spring Boot

Spring Boot on Spring raamistikul põhinev vabavaraline ja avatud lähtekoodiga Java raamistik, mida kasutatakse tihti ettevõtterakenduste, mikroteenuste ja veebirakenduste arendamiseks [32]. Spring Boot teeb kvaliteetse tarkvara arendamise Spring raamistikus kergemaks ja kiiremaks, pakkudes järgmisi funktsioone:

1. Automaatne konfiguratsioon ehk Spring Boot konfigureerib kolmanda poole tarkvara paketid ja ühendused kolmanda poole teenustega automaatselt.
2. Eeldustel põhinev konfiguratsioon ehk Spring Boot teeb konfigureerimisel eeldusi, võttes arvesse kõige populaarsemaid kasutusviise ning valdkonna parimaid tavasid. Alati on võimalik vajadusel konfiguratsiooni muuta.
3. Isesisev rakendus ehk Spring Boot loob iseseisva rakenduse, mida on võimalik kergelt jooksutada. Selleks hõlmab Spring Boot rakendus ise endas veebiserverit ja loob sellega automaatselt ühenduse.

Tänu nendele omadustele on Spring Boot leidnud kasutust enim just veebiteenuste ja mikroteenuste arendamisel, kus on vaja mitmeid rakendusi luua ja konfigureerida. Sellest lähtuvalt kasutavad ka kõik POHAK mikroteenused Javat ja Spring Booti, et oleks kerge ja kiire teenuseid üles seada ja oleks võimalik konfiguratsiooni teenuste vahel kergelt kopeerida.

3.1.3 Gradle

Gradle on vabavaraline ja avatud lähtekoodiga automaatne ehitustööriist, mis on võimeline koodi kompileerima, testima, sõltuvusi haldama, ning palju muud tänu tuhandetele pistikmoodulitele (ingl *plugin*). Gradle on üks populaarsemaid ehitustööriistu ja sellel on laialdane tugi nii eri programmeerimiskeeltes kui ka arenduskeskkondades [33].

Gradle'i põhiliseks eeliseks teiste populaarsete ehitustööriistade nagu Maven ja Ant ees on just selle kiirus. See tuleneb suures jaos sellest, et Gradle ehitab tarkvara järk-järgult, ning jälgib, mis osasid rakendusest on muudetud, seejärel uuesti ehitades, ehitatakse ainult vajalikud osad [33].

3.1.4 Redis

Redis on populaarne avatud lähtekoodiga mälusisene võti-väärtus andmebaas [34]. Kuigi Redisel on mitmeid kasutusalasid, siis kasutatakse seda enamasti just tema kiiruse pärast rakenduse sisese puhvrina. Redis salvestab endas võti-väärtus paare, kus võtmeteks on sõned ja väärtuseks võib olla mitmeid erinevaid andmestruktuure nagu näiteks sõne, list või hulk. Redis on väga kiire, kuna salvestab infot põhimälus ja just sellel põhjusel kasutatakse ka seda POHAK projektis puhvrina.

3.1.5 RabbitMQ

RabbitMQ on avatud lähtekoodiga sõnumimaakler, mis toetab nii AMQP (ingl *Advanced Message Queuing Protocol)* kui ka MQTT (ingl *Message Queuing Telemetry Transport*) protokolli [35]. AMQP on veebirakenduste puhul väga levinud sõnumi protokoll, mis ei piira kuidagi seda, millises formaadis sõnumi sisu on. MQTT protokolli kasutatakse pigem IOT (ingl *Internet of Things*) lahendusteks. RabbitMQ-d tuntakse kui paindliku tööriista, sest ühildub erisuguste programmeerimiskeelte ja arenduskeskkondadega. RabbitMQ abil on võimalik seadistada nii asünkroonseid sõnumijärjekordi kui ka järjekordade abil teostada RPC (ingl *Remote Procedure Call*) väljakutseid.

Hajusa arhitektuuriga rakenduste puhul on enamasti kaks laialt levinud viisi kommunikeerida eri teenuste vahel, millest annab hea ülevaate Menasce töö „MOM vs RPC: communication models for distributed applications“ [36].

Üheks viisiks on kasutada sõnumi järjekordi ja selleks on enamasti vaja eraldi sõnumimaakleri serverit, mis neid järjekordi haldaks. Üks populaarseid vabavaralisi variante selleks ongi RabbitMQ. Teenustel on võimalik saata sõnumeid eri järjekordadesse ja registreerida eri järjekordadele, et sealt sõnumeid vastu võtta. Selline kommunikatsioon on enamasti asünkroonne ja tagab teenuste minimaalse omavahelise sõltumise. Näiteks ei pea üks teenus

üldse teadma, milline teenus või teenused tema sõnumi järjekorrast vastu võtavad või kas üldse keegi võtab. See annab kogu rakendusele ka suurema robustsuse, kuna mõne teenuse ebaõnnestumine ei mõjuta kuidagi teisi teenuseid.

Teine populaarne meetod on RPC, see on lihtne ja klassikaline lähenemine, kus kutsutakse välja mingi meetod, mis asub teises serveris. Selline suhtlus on enamasti sünkroonne ning sellise suhtluse puhul on ka kerge arendada teenuseid, kuna teise teenusega ühendus peidetakse tavaliselt arendaja eest ära. Samas nagu Menasce välja toob, jätab RPC suhtlus teenused omavahel rohkem seotuks. Üks tänapäeval väga populaarseid RPC teeke on gRPC, mis on välja arendatud Google'i poolt [37].

Otsus, kas kasutada RPC-d või sõnumite järjekordadega kommunikatsiooni mikroteenuste vahel, sõltub enamasti sellest, kas soovitakse sünkroonset või asünkroonset suhtlust. POHAK rakenduses kasutatakse RabbitMQ sõnumimaaklerit, et hoida teenuseid üksteisest võimalikult vähe sõltuvana ja tõsta seeläbi rakenduse robustsust ja veataluvust. Lisaks saab RabbitMQ abil teostada ka järjekordade abil RPC mustrit.

3.1.6 Feign

Feign on deklaratiivne HTTP päringute Java teek, mis on loodud Netflixi poolt [38]. Feign toetub annotatsioonidele ja selle abil on kerge teha HTTP päringuid Javas, kasutades juba tuttavaid ja populaarseid komponente nagu Apache HTTP klient ja JAXB (ingl *Java Architecture for XML Binding*). POHAK projektis kasutatakse Feign teeki, et lihtsustada HTTP päringute tegemist X-tee REST teenuste pihta.

3.1.7 Lombok

Lombok on Java teek, mis lihtsustab Java klasside kirjutamist, pakkudes annotatsioone, selleks et automaatselt implementeerida mingeid osasid klassist [39]. Lomboki abil saab automaatselt genereerida konstruktoreid, *getter* ja *setter* meetodeid ja teisi üldlevinud Java klasside meetodeid. See vähendab koodi, mida peab kirjutama, ja hoiab klasse puhtamana.

3.1.8 Docker

Docker on avatud lähtekoodiga tarkvara ja platvorm, mis võimaldab luua, publitseerida ja hallata konteinereid [40]. Konteinerid on eraldatud virtuaalsed keskkonnad oma operatsioonisüsteemi ja muu tarkvaraga. Konteinerid on tänapäeval laialdaselt kasutatud, et lihtsustada ja automatiseerida rakenduste ja teenuste publitseerimist, haldamist, ehitamist või liigutamist. Docker on üks populaarsemaid konteineriseerimise tarkvarasid tänapäeval ning seda kasutatakse POHAK projektis, et hallata mikroteenuseid ja teisi vajalikke kõrvalteenuseid [41]. Lokaalseks arendamiseks on tekitatud eraldi *dev-ops* teenus, mille käivitamisel jooksutatakse kõik vajalikud konteinerid.

3.1.9 TestContainers

TestContainers on avatud lähtekoodiga raamistik, mis kasutab konteinerite tehnoloogiat selleks, et jooksutada integratsiooniteste erinevates programmeerimiskeeltes ja raamistikes [42]. TestContainers teeb integratsioonitestide kirjutamise ja jooksutamise palju lihtsamaks, kuna kasutab konteinereid, et jooksutada võltsteenuseid (ingl *mock services*). Seeläbi puudub vajadus seadistada eraldi andmebaase ja muid teenuseid nagu sõnumimaaklereid ainult testimise jaoks. Lisaks sellele, kuna TestContainers kasutab konteinerite tehnoloogiat, siis toimub võltsteenuste ülesseadmine kiirelt.

Tulenevalt kõrgest testidega kaetavuse nõudest kasutatakse POHAK projektis TestContainers raamistikku, et jooksutada integratsiooniteste, mis kataks ära suurt osa funktsionaalsusest.

3.1.10 Mockito

Mockito on avatud lähtekoodiga Java teek, mis on mõeldud automaattestide kirjutamiseks [43]. Mockito põhiline eesmärk on lihtsustada testide kirjutamist, pakkudes mitmeid eri meetodeid ja annotatsioone, et lihtsamini imiteerida (ingl *mock*) Java klasse. Klasside imiteerimine või teisisõnu võltsimine on levinud tehnika ühiktestide kirjutamisel, mille mõte on imiteerida liideseid, et oleks võimalik testida ainult ühte kindlat komponenti korraga.

3.1.11 KeyCloak

KeyCloak on avatud lähtekoodiga ning vabavaraline autentimisserver [44]. KeyCloak võimaldab ainulogimisega pöördust, mis lubab kasutada sama autentimisserverit erinevate rakenduste jaoks. KeyCloak on samuti paindlik ja lubab erinevaid autentimisviise, näiteks kasutades Google kontot või OAuth 2 OpenID Connect protokolli. POHAK projekt kasutab KeyCloak serverit tulenevalt mittefunktsionaalsest nõudest kasutada juba olemas olevat tsentraalset autentimislahendust.

3.2 Pääsukesega liidestumise protsess

See peatükk kirjeldab lähemalt Pääsukesega liidestumise protsessi ja selleks vajalikke kokkuleppeid RIA-ga, POHAK rakenduse liidestumise näitel.

Selleks, et liituda Pääsukese teenusega, on vajalik see kokku leppida RIA-ga. Selleks on Pääsukese kodulehel olemas vastav liitumisleping[9]. Sealt edasi käib Pääsukesega liidestumine sisuliselt kahes etapis: liitumine testkeskkonnaga ja liitumine toodangukeskkonnaga. Mõlema keskkonnaga liitumiseks on tarvis täita eraldi leping, kus tuleb kirjeldada, mis on

[9] Liitumisleping on saadaval RIA lehel: https://www.ria.ee/sites/default/files/documents/2024-03/Paasuke-liitumisleping-2024.pdf

Pääsukesega liitumise eesmärk, ning kinnitada, et on tutvutud tehnilise dokumentatsiooniga. Testkeskkonda on võimalik peale rollide konfigureerimist tekitada testkasutajad, et arendada kõik vajalikud liidese otspunktid. Seejärel, kui kõik on testitud saab vastavad rollid liigutada toodangukeskkonda.

3.2.1 Rollide konfiguratsioon

Lisaks lepingutele on tarvis ära kirjeldada, milliseid rolle on soov Pääsukeses hoida. Selleks on vaja lähemalt tutvuda Pääsukese tehnilise dokumentatsiooniga, millest antakse järgnevalt lühike ülevaade, et mõista POHAK rollide konfiguratsiooni Pääsukeses[10] [45].

Nagu eelnevalt mainitud on üle X-tee Pääsukese kasutamiseks kaks põhilist varianti. Üks variant on hoida rolle enda infosüsteemis ja pakkuda ise X-tee otspunkti, mille kaudu saab Pääsuke pärida rolle. Teine variant on hoida rolle Pääsukeses ja tarbida Pääsukese poolt pakutavat X-tee otspunkti, mille kaudu pärida ise rolle Pääsukesest. Rakenduse POHAK raames kasutatakse teist varianti ehk isikud saavad anda volitusi Pääsukese enda kasutajaliideses, mis asub eesti.ee portaalis. Seejärel saab rakendus POHAK pärida sealt portaalist vastava isiku rolle.

Pääsukese rollide kirjeldamisel on oluline üheselt mõista järgmiseid konfiguratsioonis kasutatavaid mõisteid:

- *Representee* - füüsiline või juriidiline isik, kes on kedagi teist volitanud ehk volitaja;
- *Delegate* - füüsiline või juriidiline isik, keda on keegi volitanud ehk volitatu;
- *Role* - mingi grupp õiguseid, mida volitaja saab anda volitatule;
- *Role namespace* - esimene osa rollist kuni koolonini ehk antud rolli nimeruum;
- *Mandate* - volitus, mis on antud volitatule volitaja poolt.

Esiteks on vaja määrata soovitud rollidele mingi nimeruum. Iga roll Pääsukeses peab kuuluma kindla nimeruumi alla. Tavaliselt omistatakse nimeruum ühele asutusele, kuid see ei ole kindel reegel.

Lisaks sellele on igal rollil hulganisti parameetreid, mis määravad selle rolli täpse käitumise. Siinjuures peab tähele panema, et Pääsuke ei salvesta mitte mingil moel konkreetseid õiguseid, mis iga rolliga kaasnevad, vaid neid peab klient hoidma enda süsteemis. Seega ei ole need ka osa rolli konfiguratsiooni parameetritest. Tabelis 1 on välja toodud olulisemad rolliparameetrid lähtuvalt sellest tööst koos seletustega ja POHAK rakenduse väärtustega[11].

[10] Ajakohane ja terviklik tehniline dokumentatsioon on saadaval Pääsukese Githubi lehel: https://github.com/e-gov/PH

[11] See ei ole täielik nimekiri: rollide konfiguratsiooni info on saadaval Pääsukese Githubi lehel: https://github.com/e-gov/PH/blob/main/spec/Introduction_to_Paasuke_and_configuring_roles.v0.4.pdf

Tabel 1 Pääsukese rolli parameetrid

Parameeter	Kohustuslik	Tüüp	Kirjeldus	POHAK väärtus
Code	jah	sõne	Unikaalne rolli identifikaator, mis koosneb nimeruumist ja rolli nimest	POHAK:Sisestaja
Title	jah	tõlke objekt	Rolli pealkiri erinevates keeltes	{ "et": "Patsiendiohutusjuhtumi lisamine", "en": "Patsiendiohutusjuhtumi lisamine (eng)", "ru": "Patsiendiohutusjuhtumi lisamine (rus)" }
Description	ei	tõlke objekt	Rolli kirjeldus erinevates keeltes	{ "et": "Annab volitatud isikul õiguse lisada volitaja nimel patsiendiohutusjuhtumeid POHAK infosüsteemi.", "en": … "ru": … }
Representee Type	jah	*enum* [NATURAL_PERSON, LEGAL_PERSON]	Mis tüüpi võib selle rolli volitaja olla	["LEGAL_PERSON"]
Delegate Type	jah	*enum* [NATURAL_PERSON, LEGAL_PERSON]	Mis tüüpi võib selle rolli volitatu olla	["NATURAL_PERSON"]
AddableBy	ei	list sõnedest	Et seda rolli anda, peab olema volitajal üks järgnevatest rollidest	["BR_REPRIGHT:SOLEREP"]
addingMustBeSigned	ei	*boolean*	Selleks, et seda rolli anda, peab andma e-allkirja	*true*
withdrawableBy	ei	list sõnedest	Selleks, et seda rolli tagasi võtta, peab volitajal olema üks järgnevatest rollidest.	["BR_REPRIGHT:SOLEREP"]

withdrawal-MustBeSigned	ei	*boolean*	Selleks, et seda rolli tagasi võtta, peab andma e-allkirja	*true*
waivableBy	ei	list sõnedest	Selleks, et seda rolli tühistada, peab volitatul olema üks järgnevatest rollidest.	["NATURAL_PERSONS:SELFREP"]
waivinMustBe-Signed	ei	*boolean*	Selleks, et seda rolli tühistada, peab andma e-allkirja	*true*
subDelegable	jah	*enum* [YES, NO, ASK, LEGAL_PERSON_YES__NATURAL_PERSON_ASK, LEGAL_PERSON_YES__NATURAL_PERSON_NO]	Kas rolli saab edasi anda	*false*

Kuna Pääsukesega on liidestunud ka äriregister, on Pääsukeses olemas ka äriregistri info ja seega on võimalik pärida juriidiliste isikute esindusõigusega isikuid ning samuti määrata, kes saab anda rolle nende järgi. Nii määratakse POHAK rakenduses näiteks, et rolle saavad anda ettevõtte esindusõigusega liikmed ehk liikmed, kellel on roll „BR_REPRIGHT:SOLEREP“. Samuti saab isik ise enda kohta antud volitust tühistada. Tabelis 1 on välja toodud ainult TTO_Sisestaja roll, lisaks sellele on sarnane konfiguratsioon ka rollil TTO_Peakasutaja, vaata (vt lisa 1).

3.2.2 Rollide pärimine

Pääsukeses käib rollide pärimine üle X-tee REST liidese. Seega tuleb teha REST päring Pääsukese X-tee teenuse pihta. Nagu kõigi X-tee päringutega, siis tuleb panna kaasa mitu vajalikku päise välja (ingl *request header*) nagu kliendi info ja päringu id, mis on lähemalt kirjeldatud X-tee REST dokumentatsioonis[12]. Pääsukese teenusel on rollide pärimiseks kaks erinevat otspunkti, mis on järgmised:

1. */delegates/{delegate}/representees*, mis võtab parameetriks volitatu isikukoodi ning tagastab kõik isikud, kes on talle volitusi andnud.
2. */representees/{representee}/delegates/{delegate}/mandates*, mis võtab parameetriteks nii volitaja kui ka volitatu isikukoodid (juriidilise isiku puhul registrikoodi) ning tagastab kõik volitused, mis volitaja on volitatule andnud.

Seega on võimalik näiteks pärida esmalt kõik volitajad, kes on antud isikule volitusi andnud ja siis pärida mingi kindla volitaja poolt antud volitused. Lisaks sellele on võimalik päringut piirata nii nimeruumide kui ka rollide järgi.

POHAK rakenduses teostatakse kasutaja rollide saamiseks kaks järgnevat päringut:

1. *GET /delegates/{delegate}/representees* rollifiltriga ["BR_REPRIGHT:SOLEREP", "TERVISEAMET_POHAK:Sisestaja"], et saada teada, millistes asutustes on kasutaja POHAK Sisestaja rollis.

[12] X-tee REST dokumentatsioon: https://www.x-tee.ee/docs/live/xroad/pr-rest_x-road_message_protocol_for_rest.html

2. GET /delegates/{delegate}/representees rollifiltriga ["TERVI-SEAMET_POHAK:Peakasutaja"], et saada teada, millistes asutustes on kasutaja POHAK Peakasutaja.

Niimoodi kasutaja rollide pärimine väldib liigsete päringute sooritamist, kuna asutusi, mis võivad kasutajat volitada on teadmatu arv ja sellisel juhul tuleks teha iga asutuse kohta üks lisa päring. Siinkohal on see eriti oluline, kuna kasutaja ootab sellel ajal sisselogimist ja iga päringu ajaline kestvus on teadmata.

3.2.3 X-tee server

Selleks, et Pääsukesest üle X-tee rolle pärida, on asutusel vaja ka liituda X-teega ning seadistada X-tee turvaserver. X-teega liitumise juhend on leitav RIA kodulehelt, kuid lühidalt koosneb X-teega liitumine kolmest järgnevast sammust [46].

1. Tuleb allkirjastada X-teega liitumise leping, mida saavad teha Eesti äriregistris olevate ettevõtete juhid ja juhatuse liikmed.
2. Tuleb kasutusele võtta X-tee turvaserver. Selleks tuleb ise seadistada X-tee turvaserver või tehnilise võimekuse puudumisel turvaserverit rentida.
3. Tuleb taotleda ligipääsu andmetele, mida soovitakse tarbida. Kui on selge, mis teenuse andmeid soovitakse, saab selle kohta esitada taotluse otse X-tee iseteeninduskeskkonnast[13].

3.3 Mikroteenuste arendamine

See peatükk kirjeldab iseteeninduse autentimiseks vajalike mikroteenuste arenduse protsessi ja arenduse käigus tehtud otsuseid.

Selleks, et iseteeninduse keskkonda oleks võimalik sisse logida, oli esiteks vaja arendada X-tee liidese teenus, mis oskaks teha Päringuid üle X-tee nii Pääsukese kui MEDRE infosüsteemi otspunktide pihta. Järgmiseks oli vaja arendada rolli teenus, mis oskab koguda rolle ning neid puhvrisse salvestada. Samuti peaks rolli teenusel olema otspunkt, mille kaudu saab KeyCloak rolle küsida. Viimaks oli vaja teha vajalikud muudatused autentimisteenusesse ja lüüsi teenusesse, et kogu sisselogimisvoog toimima saada.

3.3.1 Suhtlus teenuste vahel

Suhtlus erinevate mikroteenuste vahel toimub läbi RabbitMQ sõnumimaakleri. RabbitMQ ülesseadmine käib rakenduste enda koodi kaudu kasutades „spring-boot-starter-amqp“ teeki. *Common* teenuses on kirjeldatud kõik sõnumid Java klassidena ja samuti on kirjeldatud iga sõnumi vastuse klass. Lisaks sellele pakub *Common* teenus mitmeid meetodeid, et saata sõnumeid ja samuti, et tekitada kindlate sõnumite jaoks järjekorrad. Iga mikroteenus

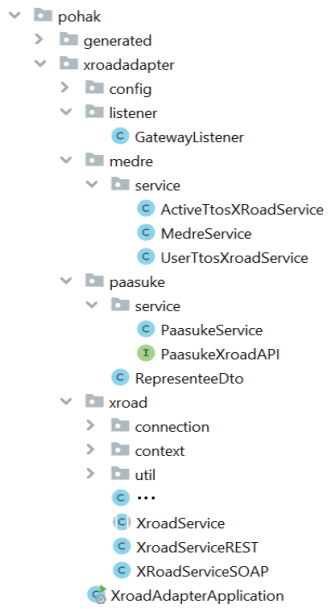
[13] X-tee iseteenindus keskkond: https://x-tee.ee/home

seab kõigi sõnumite jaoks, mida ta vastu võtab, üles ühe järjekorra, kasutades *Common* teenuse meetodit *initQues*. Järjekordade nimed koosnevad vastuvõtvast teenusest ja konkreetse sõnumi nimest, mis võimaldab meetodile *InitQues* anda ette, millise teenuse järjekordi tekitada. Seega on igast teenusest võimalik läbi *Common* teenuse koodi alati ligi pääseda teiste teenuste sõnumitele ning kasutades *Commoni* meetodeid neid vastavasse järjekorda saata (vt lisa 2).

3.3.2 X-tee adapter

X-tee adapteri teenuse põhiline ülesanne on võtta vastu järgnevaid sõnumeid:

1. *GetTtosFromXroadMessage*, mis pärib MEDRE-st kõik hetkel aktiivsed TTO-d.
2. *GetUserTtosMessage*, mis pärib MEDRE-st TTO-d, kus antud kasutaja hetkel töötab.
3. *GetDataFromPaasukeMessage*, mis pärib Pääsukesest antud kasutaja rollid.



Joonis 7. X-tee adapteri faili struktuur

Nagu joonisel 7 on näha, siis klassis *GatewayListener* paiknevad meetodid, mis kuulavad järjekordades olevaid sõnumeid. Sealt edasi kutsutakse välja vastava teenuse klassi meetod. X-tee päringute tegemiseks on samuti eraldi teenuse klass *XroadService*. Kuna MEDRE päringud kasutavad SOAP protokolli ja Pääsukese päringud REST protokolli, siis on *XroadService* klass abstraktne ja mõlema protokolli jaoks on veel eraldi alamklass (vt lisa 3 ja 4). Lisaks sellele kasutab Feign päringute kirjeldamiseks liideseid (ingl *interface*). Selleks, et *XroadServiceRest* klass püsiks piisavalt üldisena, on sellel ka geneeriline (ingl *generic*) parameeter *apiInterface*, mis sisaldab endas antud teenust kirjeldavat liidest. Pääsukese puhul on selleks *PaasukeXroadAPI* (vt lisa 5).

Kuna SOAP protokoll on XML põhine, siis on erinevatele SOAP vastustele vastavad klassid automaatselt genereeritud, kasutades .wsdl (ingl *Web Service Description Language*) faili. Wsdl-failid ongi selleks, et kirjeldada SOAP-i põhist veebiteenust. Klassid on genereeritud kasutades „com.github.edeandrea.xjc-generation“ teeki ning joonisel 8 välja toodud konfiguratsiooni build.gradle failis. SOAP päringute tegemiseks on kasutatud Spring raamistiku *WebServiceTemplate* klassi, mis on Spring raamistiku poolt pakutav klass SOAP päringute tegemiseks.

```
xjcGeneration {
    schemas {
        tam {
            schemaFile = 'tam14.wsdl'
            javaPackageName = 'ee.sm.ta.pohak.generated.tam'
            additionalXjcOptions = ['encoding': 'UTF-8']
        }
    }
}
```

Joonis 8. SOAP klasside genereerimine wsdl-faili järgi

Üks oluline osa, mille peale rollide pärimisel mõelda tuli, oli see, et see liigselt aega ei võtaks, kuna kasutaja peab samal ajal ootama, et saada sisselogitud. Seega on oluline, et X-tee päringud toimuksid paralleelselt. Javas on paralleelsuse saavutamiseks *Thread API*, mis lubab kasutada mitut lõime, et täita käske samal ajal. Nagu Oracle dokumentatsioonis on aga välja toodud, siis tuleb siinkohal tähele panna, et Java lõimed kulutavad palju ressursse, kuna need kasutavad ka eraldi füüsilisi lõimesid, mille koordineerimine võtab palju ressursse [47]. Samuti on serveri füüsiliste lõimede arv piiratud, seega kui veebirakenduse pihta tuleb mitmeid päringuid, siis tuleb hästi läbi mõelda, et ei loodaks liigselt lõimi. Seega on Java lõimed väga head, kui on vaja teha mitmeid arvutusi samal ajal, kuna JVM (ingl *Java Virtual Machine*) saab delegeerida töö mitmele füüsilisele lõimele. Samas ei ole Java lõimed kõige paremad, kui on vaja vastata sadadele päringutele, kuna need ei skaleeru hästi. Tänapäeval on üks levinud tehnikaid kasutada ühist lõime kogumikku (ingl *thread pool*), kus on kindel arv lõimi, mis on valmis töötama [47]. See väldib liigsete lõimede kasutamist. Samas kui päringuid tuleb palju, siis ei taha me ka osade päringutega ootama jääda, kuna kõik lõimed kogumikust on otsas.

Üks uus lahendus sellele probleemile on kasutada Java 21-ga tulevat uut funktsionaalsust ehk virtuaalseid lõimi (ingl *Java virtuaal threads*). Virtuaalsed lõimed pakuti esmalt välja JEP (ingl *Java Development Kit Enhancement Proposal*) 425 raames, kuid nendest antakse põhjalik ülevaade JEP 444 dokumendis [48]. Virtuaalsed lõimed on palju n-ö kergekaalulisemad lõimed, mis ei ole tingimata ühendatud füüsiliste lõimedega. Virtuaalsed lõimed on mõeldud just situatsioonidesse, kus on soov kasutada väga palju lõimi, et tegeleda suure arvu päringutega. Tänapäeval kasutavad mitmed serverid näiteks arhitektuuri, kus kasutatakse uut lõime iga sissetuleva päringu jaoks, kuna enamus päringud sisaldavad omakorda päringuid, kus taga peab ootama. Sellisel juhul ei ole vaja proovida tekitada kümneid või

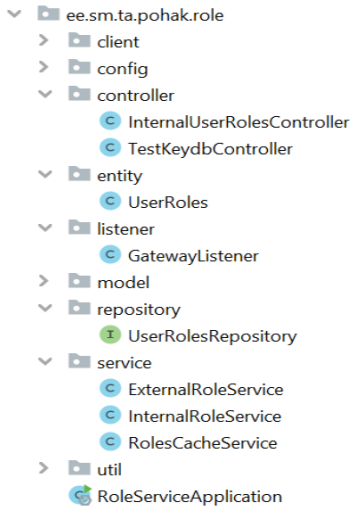
sadu süsteemseid lõimesid. Hea on ka see, et virtuaalsed lõimed kasutavad sama *Thread API*-t mis Java süsteemsed lõimed.

POHAK rakendus kasutab Java virtuaalseid lõimi, kasutades *ExecutorService API-t*, mis on kõrgematasemeline API lihtsamaks lõimede haldamiseks ja on ka tänapäeval üks levinuimaid viise Javas lõimedega programmeerimiseks [47]. Lõimedega programmeerimise osas peab samas ettevaatlik olema ja lõimede eluea peale mõtlema, et ei tekiks lõimi, mis jäävad jooksma. Näiteks tuleks mõelda, mis juhtub, kui üks lõim annab vea, kas selle peale väljutakse meetodist kohe, mis saab sellisel juhul teisest lõimest? Üks võimalus Javas lõimedega programmeerimiseks on *Structured Concurrency API*, mida on kirjeldatud põhjalikult JEP 428 raames. See moodul keskendub sellele, kuidas kergemini ja ohutumalt lõimedega programmeerida [49]. Siiski on see alles eelvaate funktsioon (ingl *preview feature*) ja seega POHAK projektis seda ei kasutata.

3.3.3 Rolli teenus

Rolli teenus haldab nii iseteeninduse kui ka ametniku keskkonna rolle, kuid selle töö raames on implementeeritud osad, mis tegelevad iseteeninduse rollidega ja seega keskendub järgnev vaid rolli teenuse iseteeninduse osale.

Rolli teenuse põhiliseks ülesandeks on koguda kokku kasutaja rollid ning salvestada need puhvrisse, mis toimub *GatherRoles* sõnumi raames. Samuti peab rolli teenus pakkuma HTTP otspunkti selleks, et KeyCloak saaks pärida kasutaja rolle.



Joonis 9. Rolli teenuse struktuur

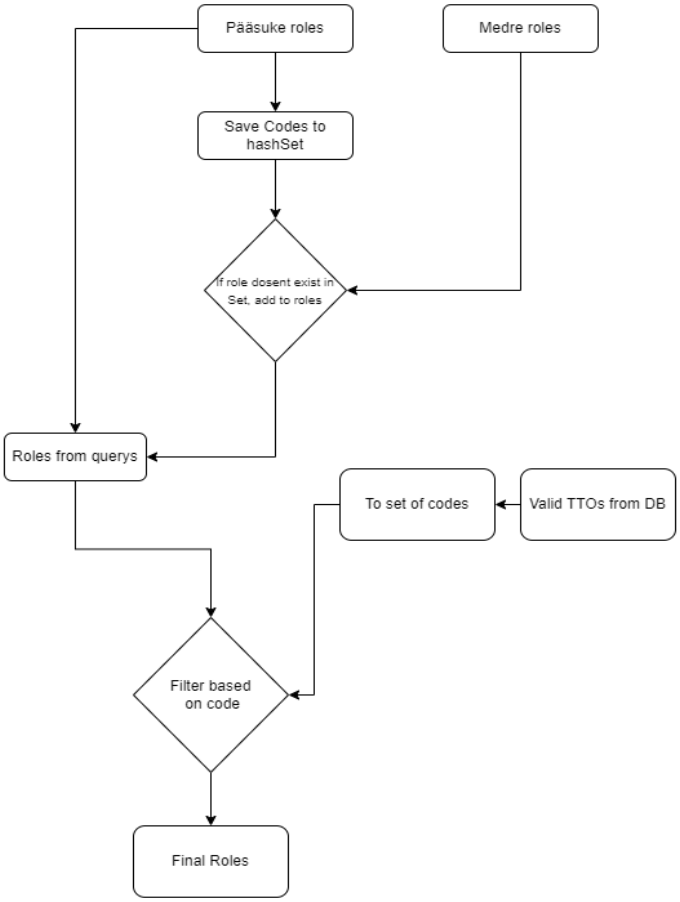
Rolli teenuse struktuur on näha joonisel 9. Põhiliselt koosneb see *Listener* klassist, mis kuulab järjekorras olevaid sõnumeid, ning *Service* klassidest, mis tegelevad vastavalt, puhverdamisega ning iseteeninduse ja ametniku rollidega.

Redisesse rollide salvestamine toimub läbi *RolesCacheService* klassi, mis defineerib lihtsad CRUD meetodid rollide haldamiseks. Selleks on kasutatud *spring-boot-starter-data-redis* teeki, mis võimaldab ilma suurema vaevata mingi olemi klassi (ingl *entity class*) järgi automaatselt luua vajalikud CRUD meetodid, andmebaasiga suhtlemiseks. Samuti nagu Spring Boot-i puhul tavaks luuakse ühendus redis andmebaasiga automaatselt, kasutades vajalikke ühendusparameetreid, mis on defineeritud *properties* failis. Konkreetne olemi klass on näha joonisel 10. Algselt ei sisaldanud see *boolean* parameetreid päringute kohta, kuid lähemal arutamisel selgus, et see info on siiski vajalik. Selle järgi on võimalik märgiste uuendamisel või uuesti sisselogimisel sooritad päringud uuesti kui üks päringutest on varem läbi kukkunud. Samuti on kasutatud eraldi ttl (ingl *time to live*) välja, mis seadistatakse alati puhvri teenuse klassi meetodites. Ttl välja abil on kerge pärida objekti puhvris oleku aega ning seda uuendada kui selline funktsionaalsus peaks tulevikus vajalik olema.

```
@AllArgsConstructor
@Builder
@Data
@NoArgsConstructor
@RedisHash
public class UserRoles implements Serializable {

    @Id
    private String personalCode;
    private Integer userId;
    private String firstName;
    private String lastName;
    /**
     * Roles in the form '12345_TTO_SISESTAJA' or '12345_TTO_...'
     */
    private List<String> roles;
    private boolean medreQueryOk;
    private boolean paasukeQueryOk;

    @TimeToLive
    private long ttl;

}
```

Joonis 10. Kasutaja rollide olemi klass

Rollide kogumise meetod, kasutab sarnaselt X-tee adapterile Java virtuaalseid lõimi, et sooritada mitmeid päringuid paralleelselt. Joonisel 11 on näha, millise algoritmi järgi töödeldakse eri päringutest kogutud rolle, et need puhvrisse salvestada.



Joonis 11. Rollide töötlemise voog

Siinkohal on oluline tähele panna, et Pääsukese päringust tulnud rollid sisaldavad nii peakastuaja kui ka sisestaja rolle, seevastu infosüsteemist MEDRE tulnud päringu rollid sisaldavad ainult sisestaja rolle. Seetõttu peab rolle töötlema viisil, kus tuleb sama TTO koodiga rolli puhul jäta alles vaid peakasutaja roll (vt lisa 6).

3.3.4 Autentimisteenus

Selleks, et sisselogimise iseteeninduse voog oleks lõplik, oli vaja autentimise teenuses teha muudatusi kahele järgnevale Rabbit otspunktile:

1. *GetExternalAuthTokensMessage*, mis saab parameetriks KeyCloaki poolt väljastatud koodi, ning tagastab märgised.
2. *ExternalRefreshTokenMessage*, mis saab parameetriks märgised, ning tagastab uuendatud märgised.

Mõlemas otspunktis kutsutakse välja joonisel 12 näha olev meetod *getTokensWithRoles*, mis teeb Keycloaki pihta päringu, kontrollib kas vastuses sisalduvas pääsu märgises on rollid ning nende puudumisel saadab *GatherRoles* sõnumi. Peale seda küsib teenus KeyCloakist uusi märgiseid kasutades esimesest päringust omandatud värskendus märgist.

```
public TokenResponseDto getTokenWithRoles(String authCode, RequestMetaData requestMetaData, boolean isRefreshGrant) throws ParseException {
    TokenResponseDto initialTokens;

    log.info("getting tokens for: {}", authCode);
    if(isRefreshGrant){
        initialTokens = refreshTokens(authCode);
    }
    else{
        initialTokens = authCodeToTokens(authCode, requestMetaData);
    }

    String[] roles = initialTokens.getRoles();
    if(roles != null) return initialTokens;

    log.info("No roles were received, gathering roles...");
    String fullName = initialTokens.getFullName();
    String firstName = fullName.substring(0,fullName.indexOf(" "));
    String lastName = fullName.substring(fullName.indexOf(" ")+1);

    GatherRolesReply reply;
    try{
         reply = queueUtil.sendAndRecive(new GatherRolesMessage(authCode, firstName, lastName), GatherRolesReply.class);
    }
    catch (Exception ex){
        log.debug(ex.getMessage());
        throw new ResponseStatusException(HttpStatus.valueOf(401),ex.getMessage());
    }
    TokenResponseDto tokens = refreshTokens(initialTokens.getRefreshToken());
    if(!reply.getPayload().getMedreQueryOk()) tokens.setMedreMissing(true);
    if(!reply.getPayload().getPaasukeQueryOk()) tokens.setPaasukeMissing(true);
    if(tokens.getRoles() == null || tokens.getRoles().length == 0) throw  new ResponseStatusException(HttpStatus.valueOf(403), "no-roles");
    return tokens;
}
```

Joonis 12. autentimis teenuse *getTokenWithRoles* meetod

Selleks, et sooritada Keycloaki pihta päringuid ja töödelda JWT märgiseid kasutatakse autentimisteenuses com.nimbusds Java teeki. See on avatud lähtekoodiga teek, kus on turvaliselt implementeeritud JWT märgistega ja Oauth vooga seotud algoritmid ja meetodid. See tagab algoritmide ja räsifunktsioonide turvalisuse.

3.3.5 Automaattestid

Kõigile teenustele oli ka vaja kirjutada automaatsed ühik- ja integratsioonitestid. Nii rolli kui ka X-tee adapteri teenuses on kasutatud sarnast testide ülesseadmise struktuuri, kus on integratsiooni testideks kasutatud ülem klassi *GenericIntegrationTestSetup* ning iga integratsiooni testi klass on selle klassi alamklass.

Üldises klassis seadistatakse teek TestContainers abil vajalikud konteinerid, et imiteerida sõnumimaaklerit ja X-tee serverit. Nagu joonisel 13 on näha, kasutakse *@Testcontainers* ja *@Container* annotatsioone, et lasta teegil Testcontainers konteinerite eluiga ise hallata. Samuti on oluline seadistada vajalikud ühendusparameetrid, et Springboot ühenduks testide ajal hoopis testimiseks mõeldud konteineritega. Selleks kasutatakse *DynamicPropertySource* annotatsiooni, et dünaamiliselt muuta Springi muutujaid.

```
@Testcontainers
@Configuration
@DirtiesContext
public class GenericIntegrationTestSetup implements AutoCloseable {

    @Container
    static MockServerContainer mockServerContainer =
            new MockServerContainer(DockerImageName.parse("mockserver/mockserver:5.15.0"));
    @Container
    static RabbitMQContainer mockRabbitContainer =
            new RabbitMQContainer(DockerImageName.parse("rabbitmq:latest"));

    static MockServerClient mockServerClient;

    static final String[] QUEUE_NAMES = { RabbitUtil.getRouteKeyByCommand(GetDataFromPaasukeMessage.class) };

    @Autowired
    protected RabbitTemplate rabbitTemplate;


    @DynamicPropertySource
    static void overrideProperties(DynamicPropertyRegistry registry) {
        mockServerClient = new MockServerClient(
                mockServerContainer.getHost(),
                mockServerContainer.getServerPort()
        );
        registry.add( name: "spring.rabbitmq.port", mockRabbitContainer::getFirstMappedPort);
        registry.add( name: "spring.rabbitmq.host", mockRabbitContainer::getHost);
        registry.add( name: "xroad.url", mockServerContainer::getEndpoint);
    }
```

Joonis 13. TestContainers teegi abil konteinerite üles seadmine

Lisaks konteinerite üles seadmisele, seadistatakse geneerilises klassis ka vajalikud abimeetodid, näiteks Rabbit sõnumite saatmiseks.

Testid ise järgivad tavalist testide voogu, kus esmalt seatakse teatud tingimused, seejärel sooritatakse tegevus ja viimaks kontrollitakse, kas tulemus vastab ootusele. Selleks, et tingimusi seada on võimalik võltsservereid (ingl *mock serve*) seadistada vastama kindlatele päringutele või sõnumitele mingi kindla vastusega. Veel on oluline märkida, et enne igat uut testi tuleks igaks juhuks tühjendada kõik sõnumijärjekorrad ja samuti taastada kõik võltsserveritele seatud eeldused. Seda saab JUnit teegis kergelt teha kasutades *BeforeEach* annotatsiooni.

4 Tulemused

See peatükk käsitleb sisselogimiseks vajalikke valminud mikroteenuseid ning hindab nende vastavust püstitatud nõutele. Samuti antakse ülevaade Pääsukese kasutamisest pääsuhaldussüsteemina ning tähelepanekuid Pääsukese kasutamiseks sarnastes projektides tulevikus. Viimaks antakse põgus ülevaade potentsiaalsetest tuleviku uuendustest, mida võiks POHAK autentimis süsteemi puhul kaaluda.

4.1 Valminud rakendus

Arenduse tulemusena on valminud funktsionaalseid nõudeid täitev POHAK rakendus õigeaegselt. Käesoleva töö raames on arendatud X-tee adapteri teenus ning rolliteenus ja tehtud vajalikud muudatused autentimis teenuses selleks, et terviklikult implementeerida iseteeninduse keskkonda sisselogimine. Tehtud muudatused on üle vaadatud projekti arhitekti poolt ning jõudnud test keskkonda, läbides kõik automaattestid koos läbivustestidega. Järgnevalt antakse ülevaade POHAK rakenduse testimise tulemustest, sisselogimise perspektiivis.

SonarQube'i genereeritud ülevaatest on näha, et kirjeldatud mikroteenused vastavad projekti mittefunktsionaalsetele nõuetele. Automaattestidega kaetavus on X-tee adapateri- ja rolliteenusel vastavalt 77.3% ja 78.5%. Samuti ei tuvastanud Sonarqube kummaski teenuses turvavigu ja mõlema teenuse hinne koodi hooldatavusele on A. Täpsem Sonarqube analüüs on lisades (vt lisa 7).

Samuti on läbiviidud koormustestid, kuigi sisselogimise koha pealt ei ole need olulised, kuna X-tee liideste ülekoormamise ohu tõttu on koormustestides sisselogimine võltsitud (ingl *mocked*). Kokkuvõtlikult võib öelda, et POHAK rakenduse sisselogimine vastab projektis seatud nõuetele.

4.2 Pääsukese kasutus

Projekti nõuete ja arhitektuuri loomisel otsustati kasutada rakenduse POHAK iseteeninduskeskkonna sisselogimise implementeerimiseks RIA poolt pakutavat teenust Pääsuke. Selle otsuseni viis asjaolu, et oli tarvis otsustada isiku roll võttes arvesse seda, kas isik on TTO töötaja või on talle antud volitus TTO juhtkonna poolt. Sellise funktsionaalsuse realiseerimine nõuab aga mitut eri päringut erinevatesse andmebaasidesse ja lisaks eraldi kõrgkäideltavat ja turvalist süsteemi, et hallata antud volitusi. Selle asemel, et seda süsteemi ise implementeerida on võimalik liidestada enda süsteem Pääsukesega.

Pääsukese kasutamine antud projektis võimaldas seega hoida kokku nii aega kui ka raha, mis oleks vastasel juhul kulunud volituse süsteemi implementeerimiseks. Antud projekti raames oli liidestumise protsess kerge ja ei nõudnud suurt lisaplaneerimist või ajakulukaid lisategevusi. Samuti ei olnud tarvis teha päringuid äriregistri poole, kuna äriregister on juba Pääsukesega ühendatud. Seega kasvab Pääsukese kasutegur projektis vastavalt sellele, kui

paljude allikregistrite infot on tarvis, et rolle otsustada. Lisaks sellele annab Pääsuke lisa võimaluse igal isikul kergelt hallata ja vaadelda volitusi otse eest.ee keskkonna kaudu.

Pääsukese kasutamisel tuleks samas arvestada ka mitme olulise aspektiga. Esiteks peab arvestama sellega, et Pääsuke ei ole lõplik lahendus. See tähendab, et Pääsuke ei autendi kasutajaid ja ei salvesta infot kasutajate ega rollide õiguste kohta. Seega on klientrakendusel tarvis ise hallata kasutajaid ja rollidega seotud õiguseid. Samuti on Pääsukeses kõik rollid klientsüsteemidele ja vastavate õigustega kasutajatele (isik ise ja kõik teda esindavad isikud) avalikud. Seega ei ole sobilik Pääsukest kasutada kui mingi roll ei tohiks olla avalik.

Arenduse käigus tuli ka mõelda selle peale, kuidas rolle pärida ja mitu päringut peab Pääsukese pihta tegema, et soovitud vastust saada. Tuleb tähele panna, et ei ole võimalik saada kätte ühe isiku kõiki volitusi ühe päringuga. Enamasti peab selleks pärima kasutaja volitajaid ja siis otse volitajate volitusi antud kasutajale. See võib teatud juhtudel aga olla liiga ajakulukas tegevus. Teine võimalus, mida kasutatakse ka POHAK rakenduses on teha üks päring iga erineva rolli kohta. Suure arvu rollide korral peab aga ka selle lähenemisega tegema mitmeid päringuid. Siiski on sellisel juhul võimalus liidestuda integratsiooni mudeli järgi, kus hoitakse kõiki volitusi enda andmebaasis ja implementeeritakse otspunktid, mille kaudu saab Pääsuke volitustele ligi.

4.3 Tagasiside ja edasiarenduse võimalused

Tagasiside POHAK rakenduse kohta on siiani olnud positiivne ning rakendus on täitnud talle seatud funktsionaalseid ja mittefunktsionaalseid nõudeid. Siiski kuna tegu on esmase versiooniga rakendusest on mitmeid asju, mida annaks parandada ja samuti mitmeid funktsioone, mis olid juba algselt kõrvale jäetud tuleviku arendusteks. Selle peatüki raames käsitletakse edasiarenduse variante ainult sisselogimise perspektiivist.

Rakenduses POHAK on tarvis rolle puhverdada, et tagada KeyCloaki päringule kiire vastus ning kasutajale kiirem sisselogimine. See toob endaga kaasa ka teatud küsimusi. Näiteks, mis saab kui kasutajale järsku mingi volitus juurde antakse? Hetkel peab kasutaja selleks, et uued rollid rakenduses kajastuksid rakendusest välja logima, mille tulemusena saadetakse rolli teenusele sõnum rollid puhvrist kustutada. Tulevikus võib aga parem lahendus olla mingi aja tagant teha päringud uuesti ja teavitada kasutajat või vaikimisi lisada roll kasutajale kui talle on mõni uus volitus antud. Lisaks tuleb hoolikalt läbi mõelda, kui kaua kirjed puhvris püsivad. Hetkel püsivad kõik kirjed puhvris 12h. Tulevikus võib aga turvalisuse või mugavuse ajendil vaja olla natuke keerukamat süsteemi, kus puhvris olemise aeg ei ole staatiline, vaid seda uuendatakse iga kord kui kirjet puhvrist välja küsida.

Teine asi, mida võib tulevikus vaja minna on rolli teenuse sisene puhver, et salvestada aktiivseid TTOsid. Praegu saadetakse ohutusjuhtumi teenusele päring iga kord kui rolle kogutakse, see omakorda tähendab andmebaasi päringut. Kuna TTOd aga väga tihti ei muutu võib efektiivsuse huvides kaaluda TTOde puhverdamist rolli enda teenuse sees. Selleks saaks kasutada näiteks Java Guava cache teeki[14].

[14] https://github.com/google/guava/wiki/CachesExplained

Kokkuvõte

Bakalaureusetöö eesmärk oli luua POHAK projektile sisselogimise lahendus, mis vastaks kõigile seatud nõuetele. Samuti oli töö eesmärgiks kasutada sisselogimise voo raames RIA poolt loodud uut keskset pääsuhalduse infosüsteemi Pääsuke ning tuua konkreetne näide Pääsukese kasutamisest POHAK projekti raames, mis annaks ülevaate kogu liidestumise protsessist.

Töös selgitati erinevaid pääsuhalduse mudeleid ja nende puudusi, andes ülevaate Pääsukese vajadusest ning arhitektuurist. Seejärel kirjeldati POHAK projekti ja sellega seonduvaid nõudeid, eelkõige sisselogimise perspektiivist. Töö protsessi osas anti ülevaade valitud tehnoloogiatest, arhitektuurist ja Pääsukese rollist kogu sisselogimise voos. Viimaks kirjeldati valminud lahendust, selle vastavust nõuetele ja edasiarenduse võimalusi. Samuti anti ülevaade Pääsukese kasutamise mõjust POHAK projekti raames, ning soovitusi ja tähelepanekuid sarnastele projektidele, kes võiks kaaluda Pääsukese kasutamist enda rakenduses.

POHAK rakendus on läbinud suurema osa testidest. Seniste testide tulemused ja tagasiside nii POHAK rakendusele üldiselt kui ka rakenduse sisselogimisele on positiivne. POHAK rakendus on seega üks näide mikroteenustel põhinevast kõrgete standarditega projektist, kus on edukalt kasutatud Pääsukese poolt pakutavat pääsuhaldus lahendust, et kokku hoida nii aega kui ka rahalist ressurssi. Töö lisades on Pääsukese liidese implementatsiooni kood, samuti soovitakse POHAK rakenduse kood avaldada Eesti koodivaramus[15].

[15] Eesti koodivaramu: https://koodivaramu.eesti.ee/users/sign_in

Viidatud kirjandus

[1] He Zhang, Zijia Jia, Chenxing Zhong, Cheng Zhang, Zhihao Shan, Jinfeng Shen, Muhammad Ali Babar Shanshan Li, "Understanding and addressing quality attributes of microservices architecture: A Systematic literature review," *Information and Software Technology*, vol. 131, 2021.

[2] L. De Lauretis, "From Monolithic Architecture to Microservices Architecture," in *IEEE International Symposium on Software Reliability Engineering Workshops*, Berlin, 2019, pp. 93-96.

[3] Ruoming Pang and Ramon Caceres and Mike Burrows and Zhifeng Chen and Pratik Dave and Nathan Germer and Alexander Golynski and Kevin Graney and Nina Kang and Lea Kissner and Jeffrey L. Korn and Abhishek Parmar and Christina D. Richards and Mengzhi Wang, "Zanzibar: Google's Consistent, Global Authorization System," in *2019 Annual Technical Conference (*

[4] Ed and Weil, Timothy R. Coyne, "ABAC and RBAC: Scalable, Flexible, and Auditable Access Management," *IT Professional*, vol. 15, no. 3, pp. 14-16, 2013.

[5] Shaomin and Zhang, Haiyan and Wang, Baoyi Zhang, "Study on Centralized Authorization Model Supporting Multiple Access Control Models," *2009 Fifth International Conference on Information Assurance and Security*, vol. 2, pp. 769-772, 2009.

[6] A.K.Y.S., Auer, D., Hofer, D. and Küng, J. Mohamed, "A systematic literature review for authorization and access control: definitions, strategies and models," *International Journal of Web Information Systems*, vol. 18, no. 2/3, pp. 126-180, 2022.

[7] A. Jøsang, *Security and Trust Management*, 13th ed., G., Mitchell, C. Livraga, Ed. Oslo, Norway: Springer Cham, 2017.

[8] R.S. and Coyne, E.J. and Feinstein, H.L. and Youman, C.E. Sandhu, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38-47, 1996.

[9] David Ferraiolo, Rick Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, Karen Scarfone Vincent C. Hu, *National Institute of Standards and Technology Special Publication 800-162*., 2014.

[10] Harry Haury and Michael H. Davis Alan H. Karp, "From ABAC to ZBAC," *Information Warfare*, vol. 9, no. 2, pp. 38-46, 2010. [Online]. https://www.jstor.org/stable/26486786 (07.12.23)

[11] Damian Schenkelman, "Deep Dive Into Google Zanzibar and its Concepts for Authorization Scenarios," , Denver, 2022.

[12] Carrie Gates, "Access Control Requirements for Web 2.0 Security and Privacy," in *IEEE Web 2.0 privacy and security workshop (W2SP'07)*, California, Jan. 2007.

[13] Tahmina and Sandhu, Ravi and Park, Jaehong Ahmed, *Classifying and Comparing Attribute-Based and Relationship-Based Access Control*. New York, USA: Association for Computing Machinery, 2017.

[14] "Keskne volituste haldamise infosüsteem Pääsuke," *Riigi Infosüsteemi Amet*, 2024. [Online]. https://www.ria.ee/riigi-infosusteem/kesksed-platvormid-avalike-e-teenuste-pakkumiseks/paasuke

[15] "Keskne volituste, rollide ja pääsuõiguste haldamise süsteem Hetkeolukorra kaardistus ja analüüs," Proud Engineers OÜ, 2021. [Online]. https://www.ria.ee/media/878/download

[16] "Keskne volituste, rollide ja pääsuõiguste haldamise süsteem Tulevikulahenduse arhitektuur," Proud Engineers OÜ, 2021. [Online]. https://www.ria.ee/media/893/download

[17] "Keskne volituste, rollide ja pääsuõiguste haldamise süsteem Tulevikulahenduse kirjeldus," Proud Engineers OÜ, 2021. [Online]. https://www.ria.ee/media/896/download

[18] "Andmevahetuskiht X-tee," *Riigi Infosüsteemi Amet*, 2024. [Online]. https://www.ria.ee/riigi-infosusteem/andmevahetuse-platvormid/andmevahetuskiht-x-tee

[19] "X-Road Architecture," *X-Road Documentation*, 2023. [Online]. https://docs.x-road.global/Architecture/arc-g_x-road_arhitecture.html#version-history

[20] Riigikogu, "Tervishoiuteenuse osutaja kohustusliku vastutuskindlustuse seadus," *Riigiteataja*, 2022. [Online]. https://www.riigiteataja.ee/akt/129042022001

[21] "Oauth 2.0". [Online]. https://oauth.net/2/

[22] Oracle, "API Gateway OAuth 2.0 Authentication Flows". [Online]. https://docs.oracle.com/cd/E50612_01/doc.11122/oauth_guide/content/oauth_flows.html

[23] TEHIK, "TEHIK Mittefunktsionaalsed nouded arendustele". [Online]. https://tehik.ee/sites/default/files/2023-11/TEHIK%20Mittefunktsionaalsed%20nouded%20arenduste.pdf

[24] Arslan Ahmad, "Essential Software Design Principles You Should Know Before the Interview," *Design Gurus*, May 2023. [Online]. https://www.designgurus.io/blog/essential-software-design-principles-you-should-know-before-the-interview

[25] Wesley Resz, "Uncle Bob Martin on Clean Software, Craftsperson, Origins of SOLID, DDD, & Software Ethics," *InfoQ*, August 2018. [Online]. https://www.infoq.com/podcasts/uncle-bob-solid-ddd/

[26] Robert C. Martin, "Design Principles and Design Patterns," 2000. [Online]. https://staff.cs.utu.fi/~jounsmed/doos_06/material/DesignPrinciplesAndPatterns.pdf

[27] David Thomas Andrew Hunt, *The Pragmatic Programmer*.: Addison-Wesley Professional, 1999.

[28] Daniel Cuthbert, Jim Manico, Josh C Grossman, Elar Lang Andrew van der Stock, "Application Security Verification Standard 4.0.3," *OWASP*, October 2021. [Online]. https://github.com/OWASP/ASVS/tree/v4.0.3#latest-stable-version---403

[29] "What is Java technology and why do I need it?". [Online]. https://www.java.com/en/download/help/whatis_java.html

[30] IBM, "What is Java," *IBM Think*. [Online]. https://www.ibm.com/topics/java

[31] TIOBE, "TIOBE Index for March 2024," March 2024. [Online]. https://www.tiobe.com/tiobe-index/

[32] "Spring Boot". [Online]. https://spring.io/projects/spring-boot

[33] "Gradle Build tool". [Online]. https://gradle.org/

[34] "Redis". [Online]. https://redis.io/docs/about/

[35] "RabbitMQ". [Online]. https://www.rabbitmq.com/

[36] D. A. Menasce, "MOM vs. RPC: communication models for distributed applications," *IEEE Internet Computing*, vol. 9, no. 2, pp. 90-93, March 2005. [Online]. https://ieeexplore.ieee.org/abstract/document/1405980

[37] "gRPC". [Online]. https://grpc.io/

[38] "OpenFeign". [Online]. https://github.com/OpenFeign/feign

[39] "Project Lombok". [Online]. https://projectlombok.org/

[40] "Docker". [Online]. https://www.docker.com/

[41] Tyler Charboneau, "Key Insights from Stack Overflow's 2022 Developer Survey," *Docker blog*, June 2022. [Online]. https://www.docker.com/blog/key-insights-from-stack-overflows-2022-developer-survey/

[42] "testcontainers". [Online]. https://testcontainers.com/

[43] "Mocikto". [Online]. https://site.mockito.org/

[44] "KeyCLoack". [Online]. https://www.keycloak.org/

[45] RIA, "Description of X-Road services provided by Pääsuke (Oraakliliides)," February 2024. [Online]. https://github.com/e-gov/PH/blob/main/spec/x-road_services_provided_by_paasuke.v0.5.3.pdf

[46] RIA, "X-tee kasutusele võtmine". [Online]. https://abi.ria.ee/xtee/x-tee-kasutusele-votmine

[47] "Virtual Threads," *Java Core Libraries Developer Guide*, 2023. [Online]. https://docs.oracle.com/en/java/javase/21/core/virtual-threads.html#GUID-2BCFC2DD-7D84-4B0C-9222-97F9C7C6C521

[48] Alan Bateman Ron Pressler, "JEP 444: Virtual Threads," *OpenJDK*, March 2023. [Online]. https://openjdk.org/jeps/444

[49] Ron Pressler Alan Bateman, "JEP 428: Structured Concurrency (Incubator)," *OpenJDK*, November 2021. [Online]. https://openjdk.org/jeps/428

[50] "What is Java Spring Boot?," *IBM Think*. [Online]. https://www.ibm.com/topics/java-spring-boot

Lisad

1. Pääsukese rollide konfiguratsiooni näide

```
[
  {
    "code": "TERVISEAMET_POHAK:Sisestaja",
    "title": {
      "et": "Patsiendiohutusjuhtumi lisaja",
      "en": "Patsiendiohutusjuhtumi lisaja (eng)",
      "ru": "Patsiendiohutusjuhtumi lisaja (rus)"
    },
    "description": {
      "et": "Annab volitatud isikul õiguse lisada volitaja nimel patsiendiohutusjuhtumeid POHAK infosüsteemi.",
      "en": "Annab volitatud isikul õiguse lisada volitaja nimel patsiendiohutusjuhtumeid POHAK infosüsteemi. (eng)",
      "ru": "Annab volitatud isikul õiguse lisada volitaja nimel patsiendiohutusjuhtumeid POHAK infosüsteemi. (rus)"
    },
    "representeeType": [
      "LEGAL_PERSON"
    ],
    "delegateType": [
      "NATURAL_PERSON"
    ],
    "addableBy": [
      "BR_REPRIGHT:SOLEREP"
    ],
    "addingMustBeSigned": true,
    "subDelegable": "NO",
    "subDelegateType": [],
    "subDelegableBy": [],
    "subDelegatingMustBeSigned": false,
    "withdrawableBy": [
      "BR_REPRIGHT:SOLEREP"
    ],
    "withdrawalMustBeSigned": true,
    "waivableBy": [
      "NATURAL_PERSONS:SELFREP"
    ],
    "waivingMustBeSigned": true,
    "addableOnlyIfRepresenteeHasRoleIn": [],
    "validityPeriodFromNotInFuture": false,
    "validityPeriodThroughMustBeUndefined": false,
    "delegateMustEqualToRepresenteeOnAdd": false,
    "hidden": false
  },
  {
    "code": "TERVISEAMET_POHAK:Peakasutaja",
    "title": {
      "et": "TTO peakasutaja POHAK infosüsteemis",
      "en": "TTO peakasutaja POHAK infosüsteemis (eng)",
      "ru": "TTO peakasutaja POHAK infosüsteemis (rus)"
    },
    "description": {
      "et": "Annab volitatud isikul õiguse vaadata ja muuta kõiki volitaja patsiendiohutusjuhtumeid POHAK infosüsteemis, sh lisada.",
      "en": "Annab volitatud isikul õiguse vaadata ja muuta kõiki volitaja patsiendiohutusjuhtumeid POHAK infosüsteemis, sh lisada. (eng)",
      "ru": "Annab volitatud isikul õiguse vaadata ja muuta kõiki volitaja patsiendiohutusjuhtumeid POHAK infosüsteemis, sh lisada. (rus)"
    },
    "representeeType": [
      "LEGAL_PERSON"
    ],
    "delegateType": [
      "NATURAL_PERSON"
    ],
    "addableBy": [
      "BR_REPRIGHT:SOLEREP"
    ],
    "addingMustBeSigned": true,
```

```
    "subDelegable": "NO",
    "subDelegateType": [],
    "subDelegableBy": [],
    "subDelegatingMustBeSigned": false,
    "withdrawableBy": [
      "BR_REPRIGHT:SOLEREP"
    ],
    "withdrawalMustBeSigned": true,
    "waivableBy": [
      "NATURAL_PERSONS:SELFREP"
    ],
    "waivingMustBeSigned": true,
    "addableOnlyIfRepresenteeHasRoleIn": [],
    "validityPeriodFromNotInFuture": false,
    "validityPeriodThroughMustBeUndefined": false,
    "delegateMustEqualToRepresenteeOnAdd": false,
    "hidden": false
  }
]
```

2. InitQues meetod

```java
/**
 * Establish queues for specific messages. Optionally, post-initialization logic
 * can be used for each queue (for example, for exchange bindings)
 *
 * @param beanFactory initialized bean factory
 * @param routePrefix route name api / domain
 * @param routeDomain route domain to be initialized or NULL to initialize all
 *                    queues
 */
public static void initQueues(@NonNull ConfigurableBeanFactory beanFactory,
        @NonNull final String routePrefix,
        @NonNull final RabbitRouteDomain routeDomain) {

    Map<RabbitRouteAction, Collection<Class<?>>> routeCommandClasses = new EnumMap<>(RabbitRouteAction.class);
    Map<RabbitRouteAction, Collection<Class<?>>> acc = new EnumMap<>(RabbitRouteAction.class);

    if (routeDomain.equals(RabbitRouteDomain.GATEWAY)) {
        MESSAGES.values()
                .forEach(actionMap -> actionMap
                        .forEach((rabbitRouteAction, classes) -> acc.merge(rabbitRouteAction, classes,
                                (c1, c2) -> Stream.of(c1, c2)
                                        .flatMap(Collection::stream).toList())));
        routeCommandClasses = acc;
    } else if (MESSAGES.containsKey(routeDomain)) {
        routeCommandClasses = MESSAGES.get(routeDomain);
    }

    if (!CollectionUtils.isEmpty(routeCommandClasses)) {
        DirectExchange exchange = new DirectExchange(EXCHANGE_NAME);
        beanFactory.registerSingleton( beanName: "directExchange", exchange);

        Optional<BiConsumer<String, Queue>> commandQueuePostInitializer = Optional.of(
                (routingKey, queue) -> {
                    Binding binding = BindingBuilder.bind(queue).to(exchange).with(routingKey);
                    beanFactory.registerSingleton( beanName: routingKey + "Binding", binding);
                });

        routeCommandClasses.forEach((routeAction, commandClasses) -> {
            for (Class<?> commandClass : commandClasses) {
                String routingKey = "%s.%s.%s".formatted(routePrefix, routeAction.name(),
                        commandClass.getSimpleName());
                Queue queue = new Queue(routingKey,  durable: false);
                beanFactory.registerSingleton( beanName: routingKey + "QueueBean", queue);
                commandQueuePostInitializer.ifPresent(initializer -> initializer.accept(routingKey, queue));
            }
        });
    }
}
```

3. PaasukeService klass

```
public class PaasukeService extends XroadServiceREST<PaasukeXroadAPI> {
    private static final ExecutorService executor = Executors.newVirtualThreadPerTaskExecutor();
    @Autowired
    private PaasukeRoleProps roleMapping;
    public PaasukeService() { super( serviceMemberCode: "70006317",  serviceSubsystemCode: "volitused",  serviceCode: "oraakel", PaasukeXroadAPI.class); }

    @Override
    public void init() {
        super.init();
        log.info("Running with following role config. Solerep role: '{}'. Administrator role: '{}'. User role: '{}'.",
                roleMapping.getSolerep(), roleMapping.getAdministrator(), roleMapping.getUser());
    }
    public List<RoleDto> getUserMandates(RequestMetaData requestMetaData, String userId){

        if(userId == null){
            throw new NullPointerException("userId missing");
        }
        Future<List<RepresenteeDto>> sisestajadFuture =  executor.submit(() ->
                apiInterface.SisestajaRoles(userId, roleMapping.getSolerep(), roleMapping.getUser()));
        Future<List<RepresenteeDto>> peakasutajadFuture =  executor.submit(() ->
                apiInterface.PeakasutajaRoles(userId, roleMapping.getAdministrator()));

        try {
            List<RepresenteeDto> sisestajad = sisestajadFuture.get();
            List<RepresenteeDto> peakasutajad = peakasutajadFuture.get();

            List<RoleDto> roles = new ArrayList<>();

            sisestajad.forEach((RepresenteeDto rep) -> {
                if(peakasutajad.contains(rep)){
                    roles.add(new RoleDto(Enums.SelfserviceRole.TTO_PEAKASUTAJA, rep.getIdentifier()));
                    peakasutajad.remove(rep);
                }
                else{
                    roles.add(new RoleDto(Enums.SelfserviceRole.TTO_SISESTAJA, rep.getIdentifier()));
                }
            });
            peakasutajad.forEach((RepresenteeDto rep) -> {
                roles.add(new RoleDto(Enums.SelfserviceRole.TTO_PEAKASUTAJA, rep.getIdentifier()));
            });

            return roles;
        } catch(InterruptedException ex){
            log.error("Role gathering process was interrupted", ex);
            Thread.currentThread().interrupt();
            throw new XRoadApiException(500, ex.getMessage());
        } catch(ExecutionException ex){
            throw new XRoadApiException(500, ex.getCause().getMessage());
        }finally {
            if(!sisestajadFuture.isDone()){
                sisestajadFuture.cancel( mayInterruptIfRunning: true);
            }
            if(!peakasutajadFuture.isDone()){
                peakasutajadFuture.cancel( mayInterruptIfRunning: true);
            }
        }
    }
}
```

4. XroadServiceREST klass

```java
public class XroadServiceREST<T> extends XroadService {

    @Autowired
    private ObjectMapper objectMapper;
    @Autowired
    private HttpComponentsMessageSender messageSender;
    @Autowired
    private XroadHttpConfig httpConfig;
    protected XRoadContext context;
    protected T apiInterface;
    protected final Class<T> apiClass;
    protected String serviceCode;

    @PostConstruct
    public void init() { initXroadService(); }

    @Override
    protected String overrideServiceCode(String serviceCodeIn) {
        if (hasText(serviceCodeIn)) {
            this.serviceCode = serviceCodeIn;
        }
        return serviceCode;
    }

    @Override
    void initXroadService() {
        super.initXroadService();
        final JacksonDecoder decoder = new JacksonDecoder(objectMapper);
        final JacksonEncoder encoder = new JacksonEncoder(objectMapper);
        Request.Options queryOptions = new Request.Options(httpConfig.getConnectionTimeout(),
                TimeUnit.SECONDS, httpConfig.getReadTimeout(), TimeUnit.SECONDS,
                followRedirects: false);

        String url = MessageFormat.format( pattern: "{0}/r1/{1}/GOV/{2}/{3}/{4}", xroadUrl, xroadInstance,
                serviceMemberCode, serviceSubsystemCode, serviceCode);

        Feign.Builder builder = Feign.builder()
                .retryer(Retryer.NEVER_RETRY)
                .errorDecoder(errorDecoder())
                .client(new ApacheHttpClient(messageSender.getHttpClient()))
                .options(queryOptions)
                .encoder(encoder)
                .decoder(decoder)
                .requestInterceptor(requestInterceptor())
                .logger(new Slf4jLogger(apiClass))
                .logLevel(Logger.Level.HEADERS);
        this.apiInterface = builder.target(apiClass, url);
    }

    protected RequestInterceptor requestInterceptor(){
        return (RequestTemplate template) -> {
            String client = MessageFormat.format( pattern: "{0}/{1}/{2}/{3}", xroadInstance,
                    xroadClientMemberClass, xroadClientMemberCode, xroadClientSubsystemCode);
            template.header( name: "X-Road-Client", client);
            template.header( name: "X-Road-Id", UUID.randomUUID().toString());
        };
    }

```

```
    protected ErrorDecoder errorDecoder(){
        return (methodKey, response) -> {
            var errorHeader = response.headers().get("X-Road-Error");
            int status = response.status();
            String body = null;
            try {
                body = new String(response.body().asInputStream().readAllBytes(), StandardCharsets.UTF_8);
            } catch (IOException e) {
                // ignore
            }

            if(errorHeader != null){
                log.error("Error connecting to Xroad service with status code {}: X-Road-Error header: {}. Body: {}",
                        status, errorHeader, body);
                return new RuntimeException("Error connecting to Xroad service");
            } else {
                log.error("Error from Xroad service with status code {}: {}", status, body);
                return new RuntimeException("Error from Xroad service");
            }
        };
    }

    public XroadServiceREST(String serviceMemberCode, String serviceSubsystemCode, String serviceCode, Class<T> apiClass) {
        super(serviceMemberCode, serviceSubsystemCode);
        this.serviceCode = serviceCode;
        this.apiClass = apiClass;
    }
}
```

5. PaasukeXroadAPI liides

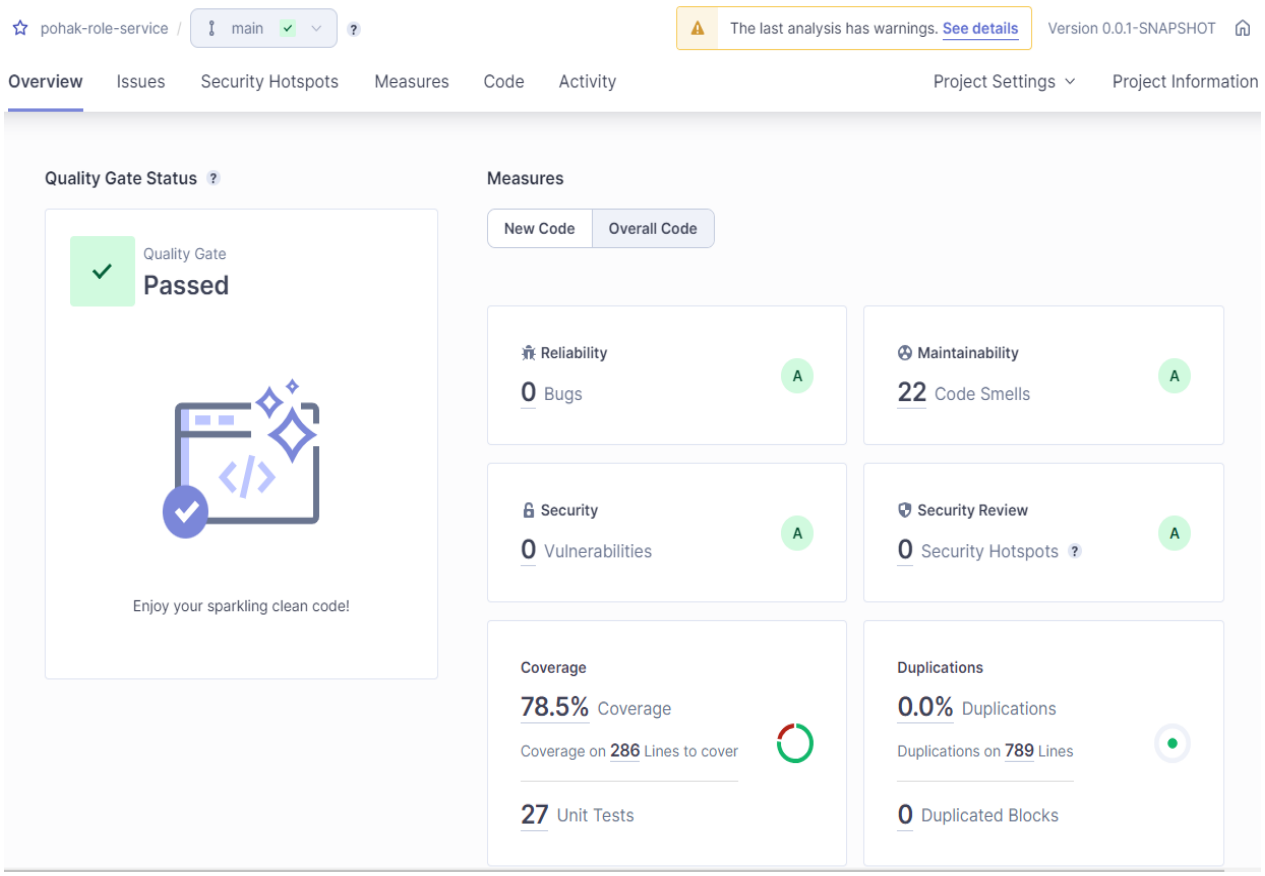
```
11  public interface PaasukeXroadAPI {
12
13      @RequestLine("GET /delegates/{identifier}/representees?role={soleRepKey}&role={userRoleKey}")  1 usage
14      @Headers({
15              "Content-Type: application/json",
16              "Accept: application/json",
17              "X-Road-UserId: {identifier}"
18      })
19      List<RepresenteeDto> SisestajaRoles (@Param("identifier") String identifier,
20                                           @Param("soleRepKey") String soleRepKey,
21                                           @Param("userRoleKey") String userRoleKey);
22
23
24
25      @RequestLine("GET /delegates/{identifier}/representees?role={administratorRoleKey}")
26      @Headers({
27              "Content-Type: application/json",
28              "Accept: application/json",
29              "X-Road-UserId: {identifier}"
30      })
31      List<RepresenteeDto> PeakasutajaRoles (@Param("identifier") String identifier,
32                                             @Param("administratorRoleKey") String administratorRoleKey);
33
34  }
```

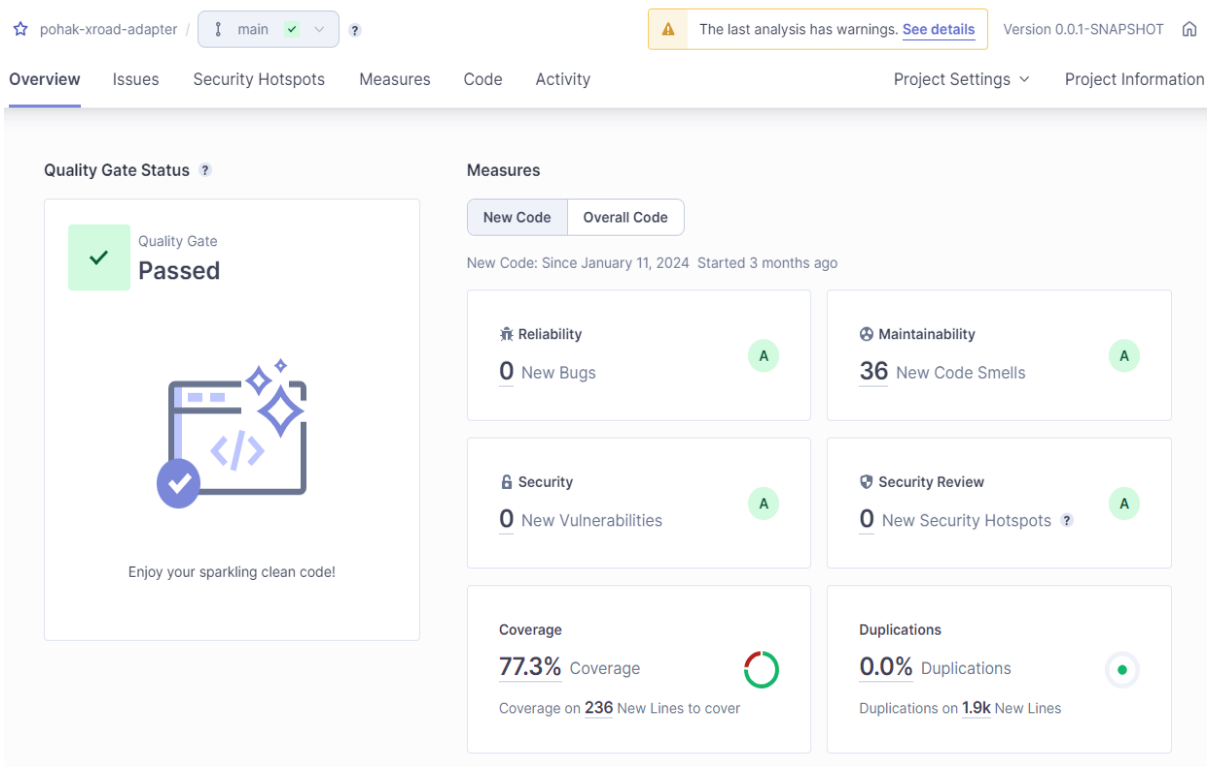
6. GatherRoles meetod

```
65  public RolesResponseDto gatherAndCacheRoles(String personalCode, String firstnName, String lastName, RequestMetaData requestMetaData){
66
67      if(personalCode == null) throw new NullPointerException("userId missing");
68
69      Optional<UserRoles> cachedRoles = cacheService.getFromCache(personalCode);
70      if(cachedRoles.isPresent() && cachedRoles.get().isPaasukeQueryOk() && cachedRoles.get().isMedreQueryOk()){
71              log.info("Roles already exist in cache!");
72              return new RolesResponseDto( medreQueryOk: true,  paasukeQueryOk: true);
73      }
74
75      UserRoles userRoles = new UserRoles();
76      userRoles.setPersonalCode(personalCode);
77      userRoles.setFirstName(firstnName);
78      userRoles.setLastName(lastName);
79      RolesResponseDto response = new RolesResponseDto( medreQueryOk: false,  paasukeQueryOk: false);
80
81      var paasukeMessage = new GetDataFromPaasukeMessage( personalCode: "EE"+personalCode);
82      paasukeMessage.setRequestMetaData(requestMetaData);
83      Future<GetDataFromPaasukeReply> paasukeFuture =  executor.submit(() -> queueUtil.send(paasukeMessage, GetDataFromPaasukeReply.class));
84
85      var medreMessage = new GetUserTtosMessage(personalCode);
86      medreMessage.setRequestMetaData(requestMetaData);
87      Future<GetUserTtosMessageReply> medreFuture =  executor.submit(() -> queueUtil.send(medreMessage, GetUserTtosMessageReply.class));
88
89      var ttosMessage = new GetTtosMessage(new GetTtosRequestDto( returnActiveTtos: true,  searchTerm: null,  limit: null));
90      ttosMessage.setRequestMetaData(requestMetaData);
91      Future<GetTtosReply> ttoFuture =  executor.submit(() -> queueUtil.send(ttosMessage, GetTtosReply.class));
92
93
94      try {
95          Optional<List<RoleDto>> paasukeRoles = getResult(paasukeFuture);
96          Optional<List<RoleDto>> medreRoles = getResult(medreFuture);
97          List<RoleDto> roles;
98
99          if(paasukeRoles.isEmpty() && medreRoles.isEmpty()){
100             throw new RuntimeException("xroad-roles-unavailable");
101         }
102
103         if(paasukeRoles.isPresent()){
104             response.setPaasukeQueryOk(true);
105             userRoles.setPaasukeQueryOk(true);
106
107             Set<String> paasukeCodes = new HashSet<>();
108             roles = paasukeRoles.get().stream().peek(role ->
109             {
110                 role.setCode(role.getCode().substring(2)); //remove EE from the start
111                 paasukeCodes.add(role.getCode());
112             }).collect(Collectors.toList());
113
```

```
            medreRoles.ifPresent(roleDtos -> {
                response.setMedreQueryOk(true);
                userRoles.setMedreQueryOk(true);
                roleDtos.forEach(role -> {
                    if (!paasukeCodes.contains(role.getCode())) {
                        roles.add(role);
                    }
                });
            });
        } else {
            response.setMedreQueryOk(true);
            userRoles.setMedreQueryOk(true);
            roles = medreRoles.get();
        }

        Set<String> ttos = getResult(ttoFuture).orElseThrow(() -> new RuntimeException("ttos-unavailable")) TtoSearchResponseDto
                .getResultsList().stream() Stream<TtoDto>
                .map(TtoDto::getCode).collect(Collectors.toSet());

        List<String> rolesToCache = roles.stream()
                .filter(role -> ttos.contains(role.getCode()))
                .map(role -> role.getCode() + "_" + role.getRole()).toList();

        log.trace("Roles gathered: ");
        rolesToCache.forEach(log::trace);

        var userLoginMessage = new UserLoginMessage(personalCode,  name: firstnName + ' ' + lastName);
        userLoginMessage.setRequestMetaData(requestMetaData);
        Optional<UsersDto> user = getResult(executor.submit(() -> queueUtil.send(userLoginMessage, UserLoginReply.class)));

        Integer userId = user.map(UsersDto::getId).orElseThrow(() -> new RuntimeException("user-service-unavailable"));
        userRoles.setUserId(userId);
        userRoles.setRoles(rolesToCache);
        cacheService.putToCache(userRoles);

        return response;
    } catch (ExecutionException ex) {
        log.error(ex.getCause().getMessage());
        throw new RuntimeException(ex.getCause().getMessage());
    } catch (InterruptedException ex) {
        log.error(ex.getCause().getMessage());
        Thread.currentThread().interrupt();
        throw new RuntimeException(ex.getCause().getMessage());
    } finally {
        if (!paasukeFuture.isDone()) {
            paasukeFuture.cancel( mayInterruptIfRunning: true);
        }
        if (!medreFuture.isDone()) {
            medreFuture.cancel( mayInterruptIfRunning: true);
        }
        if (!ttoFuture.isDone()) {
            ttoFuture.cancel( mayInterruptIfRunning: true);
        }
    }
}
```

7. SonarQube analüüs X-tee adapter- ja rolliteenusele.




pohak-xroad-adapter / main
The last analysis has warnings. See details
Version 0.0.1-SNAPSHOT
Overview
Issues
Security Hotspots
Measures
Code
Activity
Project Settings
Project Information
Quality Gate Status
Quality Gate
Passed
Enjoy your sparkling clean code!
Measures
New Code
Overall Code
New Code: Since January 11, 2024 Started 3 months ago
Reliability
0 New Bugs
A
Maintainability
36 New Code Smells
A
Security
0 New Vulnerabilities
A
Security Review
0 New Security Hotspots
A
Coverage
77.3% Coverage
Coverage on 236 New Lines to cover
Duplications
0.0% Duplications
Duplications on 1.9k New Lines

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

1. Mina, Holden Karl Hain, annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose „Keskse volituste haldamise infosüsteemiga Pääsuke liidestumine Patsiendiohutuse andmekogu näitel“, mille juhendajad on Margus Jäger ja Eero Vainikko, reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 4.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

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

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

Holden Karl Hain

13.05.2024