TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Kristjan Mäeots 185076IADB

REST API mikroteenuse disain ja arendamine Micronaut raamistikus ettevõtte näitel

Bakalaureusetöö

Juhendaja: German Mumma
MSc
Äriinfotehnoloogia

Tallinn 2021

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Kristjan Mäeots

16.05.2021

Annotatsioon

Käesoleva lõputöö peamiseks eesmärgiks oli luua ettevõtte mobiilirakendustele mikroteenus, mis lahendaks ettevõtte probleemid seoses andmete küsimise ja muutmisega vanast ja ettevõtte poolt mittehallatavast *backend* teenusest. Teiseks probleemiks oli mikroteenuste mälukasutuse parandamine.

Töötulemusena valmis *REST API* mikroteenus *Micronaut* raamistikul, mis tagastab mobiilirakendustele tema kasutajate reklaami kuvamise eelistusi ja pakub ka võimalust neid eelistusi muuta. Antud mikroteenus lahendab ettevõtte jaoks autoriseerimise puudulikkuse ning tagastab mobiilirakendustele sobiva andmemudeli. *Micronaut* raamistik kasutusele võtmine vähendas mikroteenuse mälukasutust. Autori kaugem soov on võtta *Micronaut* raamistik ettevõtte kõigis mikroteenustes kasutusele.

Mikroteenus on juurutatud ettevõtte serverisse ning sellele pääsevad ligi vaid projekti kuuluvad *IOS* ja *Android* mobiilirakendused. Lõputöös teenusele ligipääsu andmine ei ole konfidentsiaalsuse tõttu võimalik.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 36 leheküljel, 7 peatükki, 25 joonist.

Abstract

REST API Microservice Design and Development Using Micronaut Framework on the Example of a Company

The primary goal of this thesis was to develop a microservice for a company's mobile apps that would solve problems about requesting data from an old and unoptimised backen service that the author's company does not have access to develop. The additional problem was to reduce microservice footprint by reducing its memory usage with more efficient framework than currently used *Spring Boot*.

As a result of this thesis *REST API* microservice on *Micronaut* framework was developed. This microservice serves a purpose for mobile apps to show its users their marketing permissions and also offers a possibility to change those permissions. This microservice solves problems about authorizing and not suitable data model for mobile apps. Using Micronaut as its framework reduced its memory usage. Longer approach is to take *Micronaut* into use in the author's company that he works for, to reduce memory usage for all the microservices.

Microservice is deployed to a server of a company and only *IOS* and *Android* mobile apps have access to it. Further details about microservice are not added to this thesis because of the confidental reasons.

This thesis is written in Estonian and contains 36 pages of text, 7 chapters, 25 figures.

Lühendite ja mõistete sõnastik

AOT	*Ahead-Of-Time compilation*; kompileerimine enne programmi käivitumist.
Android	Tarkvarakomplekt elektroonikaseadmetele, mis hõlmab operatsioonisüsteemi, vahetarkvara ja peamisi rakendusi.
AOP	*Aspect-Oriented Programming*; aspektile orienteeritud programmeerimine.
Apache	Populaarseim veebiserveri tarkvara.
API	*Application Programming Interface*; rakendusliides.
Backend	Tagarakendus, andmete juurdepääsu kiht. Osa rakendusest, mida kasutaja ei näe.
Boilerblate	Korduvkasutatud koodijupp, mida on koodis palju kasutatud ja saaks vältida.
Boolean	Arvutiteaduses andmetüüp, millel on kaks võimalikku väärtust.
Byte	Digitaalse informatsiooni ühik.
Cache	Reserveeritud andmete koht, mis kogub ajutisi andmeid, et kiirendada rakenduste laadimist.
Container	Konteiner. Keskkond, kuhu saab rakenduse koodi pakkida koos kõigi sinna kuuluvate teekide ja sõltuvustega, mis annab võimaluse tal isoleeritud keskkonnas töötada.
Controller	Meetod, mille ülesanne on töödelda teenusesse tulevaid päringuid.
Convention over configuration	Vaikimisi standardkonfiguratsioon liigse kodeerimiseta.
CPU	Protsessor.
Dashboard	Andmete visualiseerimise töövahend.
Data binding	Tehnikavõte, mis seob pakkuja ja tarbija andmed kokku ja sünkroniseerib need.
DI	*Dependency Injection*; tehnika, milles objektile antakse teised objektid, millest ta sõltub.
DTO	*Data Transfer Object*; disainimuster, mida kasutatakse andmete edastamiseks tarkvararakenduste alamsüsteemide vahel.
Elasticsearch	Igat tüüpi andmete otsimise ja analüüsimise mootor.

Endpoint	Sidekanali üks lõpp. API ja teiste süsteemide vahelise suhtluse puutepunkt.
Erind	Arvutiprogrammi töös tekkinud eriolukord.
Facade	*Facade pattern*; disainimuster, kus klass peidab keerulise äriloogika.
Github	Koodihaldamise platvorm.
Gradle	Avatud lähtekoodiga projekti ehitamise automatiseerimise tööriist.
Grafana	Avatud lähtekoodiga visualiseerimise ja analüüsi tarkvara.
Groovy	*Java* platvormi programmeerimiskeel.
Header	*API* päringu metaandmete kandja.
HTTP	*Hypertext Transfer Protocol*.
ID	Objekti identifikaator.
InfluxDB	Aegridade andmebaas, mis on mõeldud suurte kirjutamis- ja päringukoormuste käsitlemiseks.
Invoke Dynamic	Abivahend, mille eesmärk on suurendada dünaamilist tüüpi keelte tuge *JVM*'is.
IOS	Apple mobiilne operatsioonisüsteem.
JAR	*Java ARchive*; failiformaat.
Java	Programmeerimiskeel.
Java JDK	*Java Development Kit*; *Java* rakenduste loomise peamine platvormikomponent.
Java reflection API	Koodi käivitusaegne uurimise ja muutmise *Java* rakendusliides.
JTA	*Java Transaction Management*; tehinguhaldus.
JIT	*Just-In-Time*; kompileerija, mis on käivitusaegse keskkonna komponent.
JSON	*JavaScript Object Notation;* kerge vorming andmete salvestamiseks ja transportimiseks.
JVM	*Java Virtual Machine*; abstraktne arvutusmasin või virtuaalmasina liides, mis juhib *Java* koodi.
Kibana	*Elasticsearch*'ist saadud andmete visualiseerimise rakendus.
Kotlin	Avatud lähtekoodiga staatiline programmeerimiskeel.
Kubernetes	Avatud lähtekoodiga platvorm konteineriseeritud töökoormuste ja teenuste haldamiseks.
Laziness	Käivitumise strateegia, kus mingit funktsionaalsust käivitatakse vaid vajadusel.

Library	Arvutiprogrammide poolt kasutatavate ressursside kogu tarkvara arendamiseks.
MAF	*Oracle Mobile Application Framework*; hübriid-mobiilne arendusraamistik, mis võimaldab arendusmeeskondadel kiiresti arendada.
Mapper	Klass koodis, kus luuakse ühe objekti andmetest teine objekt.
Micronaut	*JVM* raamistik mikroteenuste kirjutamiseks kasutades Java't, Kotlin'it või Groovy't.
MIT	Töö autori ettevõttes projekti kuuluv vana ja optimiseerimata andmesüsteem.
Classloader	*Java Runtime Environment* osa, mis dünaamiliselt laeb Java klassid *JVM*'sse.
Node.js	Võrgurakenduste kirjutamise platvorm.
OCI	*Object Computing Inc*.; konsulteerimise ettevõte.
Pod	Väikseim juurutatav objekt *Kubernetes*'es. Tähistab klastri käimasoleva protsessi ühte eksemplari.
Proxy	Programmeerimismuster, mis kontrollib ja haldab ligipääsu objektidele, mida ta kaitseb.
RAM	Arvuti muutmälu.
Reflection	*Java* rakendusliides, mida kasutatakse, et uurida ja muuta meetodite, klasside ja liideste käitumist koodi käivitumisajal.
RESTful	REST API teise sõnaga.
REST API	*Representational State Transfer API*; tarkvara arhitektuuriline stiil rakendusliidesele, mis kasutab *HTTP* päringuid andmetele ligipääsemiseks ja nende kasutamiseks.
Serverless	Võimalus arendajatel keskenduda ainult äriloogika kirjutamisele. Serveri infrastruktuuriga arendajad tegelema ei pea.
Session token	Krüpteeritud unikaalne sõne, mis identifitseerib spetsiaalse sessiooni olemust.
Skeleton teenused	Tühi projekt kõige väiksema võimaliku funktsionaalsusega.
Skript	Hulk käske, mida jooksutab mingi kindel programm.
Spring Boot	*Java* platvormi raamistik.
SOAP	*Single Object Access Protocol; XML* baasil infovahetuse protokoll arvutite vahel.
String	Objekt, mis koosneb tähemärkide järjestusest.
Swagger	Tööriist *API*'de kirjeldamiseks.
UI	*User Interface*; kasutajaliides.

Unit test	Väiksemate osade testimine rakenduses.
Vert.x	JVM tööriistakomplekt.
XML	*Extensible Markup Language;* märgistuskeel dokumendi struktuuri loomiseks.
WSDL	*Web Services Description Language*; *XML*-baasil liidese kirjeldamise keel, mis kirjeldab veebiteenuse funktsionaalsusi.

Sisukord

Jooniste loetelu

1 Sissejuhatus

Käesolev lõputöö kirjeldab mikroteenuse loomise protsessi *Micronaut* raamistikul ettevõtte näitel. Lõputöö praktiline osa on luua, hiljem projekti kuuluvate mobiilirakenduste poolt kasutusesse minev, mikroteenus, mis lahendaks ettevõtte poolt antud probleemid seoses autoriseerimise ja andmemudeliga. Loodavat teenust hakatakse kasutama mobiilirakenduste poolt *Android* ja *IOS* platvormidel. Nendel rakendustel on kahepeale kokku üle saja viiekümne tuhande igapäevase kasutaja.

Lõputöö eesmärk on, lisaks tööülesandena antud probleemide lahendamisele, saada kogemusi *Micronaut* raamistikuga, et selle najal juurutada see ettevõttes. Tiim, kus autor töötab, omab palju mikroteenuseid *Spring Boot* raamistikul ja töö autor usub, et *Micronaut*'i kasutuselevõtmine aitaks teenuste mälukasutust optimeerida. Eesmärk on katsetada uut raamistikku, õppida see selgeks ja selle põhjal teenus arendada ning võrrelda tema mälukasutuse jalajälge teiste ettevõttes olevate *Spring Boot* raamistikel olevate mikroteenustega. Eesmärk on ettevõtte teenuste mälukasutust vähendada.

Töö teoreetilises osas analüüsitakse ettevõtte poolt esitatud autoriseerimise ja andmemudeliga seonduvaid probleeme. Analüüsitakse *Spring Boot* raamistiku mälukasutust mikroteenustel ning võrreldakse seda *Micronaut*'iga. Luuakse ülevaade *Micronaut* raamistikust. Samuti seletatakse lahti teenuse kirjutamisel kasutatud arhitektuuri ja disaini aspekte. Analüüsitakse ka teenuse jälgimisvahendite kasutamist.

Lõputöö tulemuste osas esitatakse ekraanikuvad, kuidas loodud mikroteenus mobiilirakenduses lõpplahendusena kasutusel on. Samuti tuuakse võrdlus valminud *Micronaut* teenuse ja ettevõttes olemasoleva samaväärse *Spring Boot* teenuste mälukasutuste vahe, et tõestada mälukasutuse paranemist.

2 Taust

Autori loodava mikroteenuse eesmärk on hakata vahendama andmeid projekti kuuluvate mobiilirakenduste ja vana optimeerimata ning ebaturvalise infosüsteemi *MIT* vahel. Teenuse ülesandeks on tagastada kasutajale tema turunduseelistused seoses mobiilirakendusega ning pakkuda ka võimalust neid muuta.

IOS ja *Android* platvormi mobiilirakendustele on vajadus lisada funktsionaalsus, et pärida ja muuta rakenduse kasutaja turunduseelistusi. Kasutajal on seitse erinevat turunduseelistuse seadistust, mis määravad, milliseid reklaame võib mobiilirakendus talle näidata.

Teiseks eesmärgiks on luua teenus, mille mälukasutus oleks võimalikult optimiseeritud.

2.1 Probleemi kirjeldus

Teenuse loomise vajadusega seonduvaid probleeme on mitmeid. Suurimad probleemid on seoses andmevahetusega *MIT*'iga ning seda tehes autoriseerimise puudumine. Samuti on mure mikroteenuste liigne mälukasutus *Spring Boot* raamistikul [1], mida ettevõttes suur osa teenuseid kasutavad.

MIT, kust kogu mobiilirakenduste äriloogilise osa info päritakse, on vana, ebaturvaline ja optimeerimata. Sealt info pärimise jaoks on antud üks üldine kasutaja, mis on kõikidel projektidel sama. See tähendab, et iga kasutaja saab küsida ükskõik kelle teise isiku andmeid. Eesmärk on luua teenus, mis suudaks eristada kasutajaid teiste andmetest.

Teine probleem on andmemudeli mittesobivus. Tagastatavad andmed ei ole sellisel kujul nagu mobiilirakendused soovivad. *MIT*'ist otse infot pärides peavad mõlemad mobiiliplatvormid kirjutama lisakoodi, et saadud andmed rakendustega sobitada. See tähendaks palju *boilerplate* koodi ning seda saab vältida mikroteenusest valmis andmemudeli küsimisega.

Mikroteenuse mälukasutuse probleem *Spring* raamistikul. Mikroteenuste arhitektuur tähendab, et kõik projekti teenused on eraldi osadena koos ühel serveril. Sellest tulenevalt

on mikroteenuste loomisel tähtis nende mälukasutus. Seda mitte jälgides võib lihtsasti probleemiks saada, et serveri mälu ja protsessorit kasutatakse rohkem kui vaja. Selle tulemusena tuleb tellida mälu juurde, mis võtab lisaraha ja -energiat [2]. Ettevõttele, kus autor töötab, on oluline, millise keskkondliku ja sotsiaalse jalajälje ta jätab. Seda tõestab ka fakt, et ettevõte on saanud nomineerituks Ettevõtlusauhindade galal Aasta Vastuvõtliku Ettevõttena. See on ka üheks põhjuseks, mis ajendas autorit mõtlema võimalusest säästa serverite kasutatavat mälu mikroteenuste poolt.

2.2 Lähtetingimused

Ettevõtte jaoks loodav mikroteenus peab olema vahendajaks *MIT*'i ja mobiilirakenduste vahel. Tema ülesandeks on tagastada kasutaja reklaami kuvamise eelistusi ja vajadusel neid ka muuta. Loodav mikroteenus peab sisaldama täielikku kasutajapõhist autoriseerimist. Teenus peab tagastama soovitud andmemudeli, vältimaks mobiilirakendustes duplikaatkoodi. Mikroteenust peab olema võimalik jälgida ettevõtte kasutatavate riistvara kaudu.

Projekti arhitektuur, disain, raamistik, tehnoloogia ja muu teenuse kirjutamisega seonduvad valikud on vabad ning autori enda valida.

2.3 Eesmärk

Luua juba toimiva ja suure kasutajaskonnaga (üle saja viiekümne tuhande kasutaja) mobiilirakendusele loodava uue vaate jaoks vajalik *backend* (tagarakendus) teenus, mis oleks vahendajaks vajaliku info küsimisele ja muutmisele.

Eesmärk on luua teenus, mis suudaks autoriseerida, et kasutaja, kes infot küsib, küsiks ainult temale endale kuuluvat infot. Samuti on eesmärk kirjutada ümber tagastatav andmemudel nii, et see sobiks korraga nii *IOS*'ile kui ka *Android*'ile.

Micronaut'i kasutades on eesmärk tuua teenuse mälukasutus võimalikult madalale.

Kogu selle protsessi käigus on eesmärgiks kasutada parimaid programmeerimise tavasid.

2.4 Metoodika

Käesoleva lõputöö metoodika koosneb järgnevatest osadest:

1. Tööülesande analüüs. Mis on teenuse eesmärk ning mis probleemi see lahendab?
2. Valitava tehnoloogia ja arhitektuuri analüüs.
3. Teha nende seast valik ning põhjendada kõiki valikuid üksikasjalikult.
4. *REST API* (Representational State Transfer Application Programming Interface) mikroteenuse kirjutamine *Micronaut* raamistikul: töö praktiline osa.
5. Tehtud töö analüüsimine: kas teenus täidab seatud tingimusi.
6. Tulemuste kokkuvõte ja tööprotsessi järeldused.

3 Probleemi analüüs

Käesolevas peatükis esitatakse ettevõtte ja tema kliendi poolt esitatud probleemid, mida loodav mikroteenus peab lahendama. Samuti põhjendatakse teenuse kirjutamiseks *Micronaut* raamistiku kasutusele võtmist.

3.1 Ettevõtte põhjused uue mikroteenuse vajalikkusele

Projekti tellija poolt esitati nõudmine lisada mobiilirakendustele võimalus muuta kasutaja reklaami kuvamise eelistusi. Kogu sellele vastav info on talletatud varem mainitud *MIT* teenusesse. Kuna mobiilirakendused otse ei saa *MIT*'iga suhelda autoriseerimise ja *SOAP* (*Single Object Access Protocol*) päringute lisakeerukuse tõttu, siis jäigi ainukeseks variandiks luua uus mikroteenus info vahendajaks.

MIT, kust päritakse andmeid mobiilirakendustele, ei ole ettevõtte, kus autor töötab, hallata. Seal muudatuse sisseviimine on pikk ja aeganõudev protsess. Selle põhjuseks on *MIT*'i tiimi suurus ja laiahaardelisus. Lihtsamate äriloogikaga või andmemudeliga seotud muudatustega on seda autori loodavas mikroteenuses lihtne ja kiire teha. See lühendab ja lihtsustab arendusprotsessi märkimisväärselt, sest mikroteenuste arendaja on mobiilirakenduste tiimile igapäevaselt kättesaadav ressurss, kuid *MIT*'i tiimiliikmed ei ole.

3.1.1 Autoriseerimine ja turvalisus

MIT'i teenusest info küsimine on võimalik autoriseerimise poole pealt vaid üldkasutajat kasutades. See tähendab, et iga kasutaja saab ükskõik kelle teise isiku infot pärida. Ühtegi kontrolli ei tehta isikuliselt. Seetõttu on kasutajate privaatsus puudulik ning seda ei saa lubada.

Selleks on vaja lisada mikroteenusele autoriseerimisteenus, mille ülesandeks oleks teha kindlaks, et iga kasutaja küsiks *backend'ilt* ainult enda andmeid.

3.1.2 Tagastatava andmemudeli muudatused

MIT'i poolt tagastatava andmemudeli ja selle formaadiga seoses on kaks probleemi, mille saab lahendada mikroteenusega.

Esimeseks probleemiks on, et *MIT* on ehitatud *SOAP* protokollile. See tähendab, et teenus kasutab *XML* (*Extensible Markup Language*) formaati infovahetuseks [3] ja *WSDL* (*Web Services Description Language*) faile päringute funktsionaalsuse kirjeldamiseks [4]. Projekti kuuluvad mobiilirakendused ei ole hetkel implementeerinud funktsionaalsust tegema *SOAP* päringuid ega neid testima. Selle võimalikuks tegemine oleks lisatöö, mida saab vältida.

Funktsionaalse ja võimsa mobiilirakenduse kirjutamiseks peavad tema *backend* teenused olema optimeeritud mobiilide tarvitamismahu jaoks. *RESTful* (*Representational State Transfer* arhitektuuriga) veebiteenused, mis kasutavad *JSON* (*JavaScript Object Notation*) formaati on laialdaselt hinnatud parimaks arhitektuuriliseks valikuks mobiilirakenduste ja *backend* teenuste omavahelisel integreerimisel. On näiteid, kus kasutatakse mobiilirakenduste *backend* teenustes *SOAP* teenuseid, kuid seda tehakse vaid lihtsate teenuste puhul ja *MAF*'il (*Oracle Mobile Application Framework*) [5].

Teine probleem on andmemudeli mittesobivus. Mudelis on vaja teha muudatusi väljanimedes ning andmete struktuuris. Need muudatused võimaldavad vältida mobiilirakenduste kummalgi platvormil topelt koodi. Selleks tehakse vajalikud muudatused ära mikroteenuses ning mobiilirakendused saavad info õigel kujul. Seekaudu vähendatakse *boilerplate* koodi tekkimist.

3.1.3 Teenuse jälgimine

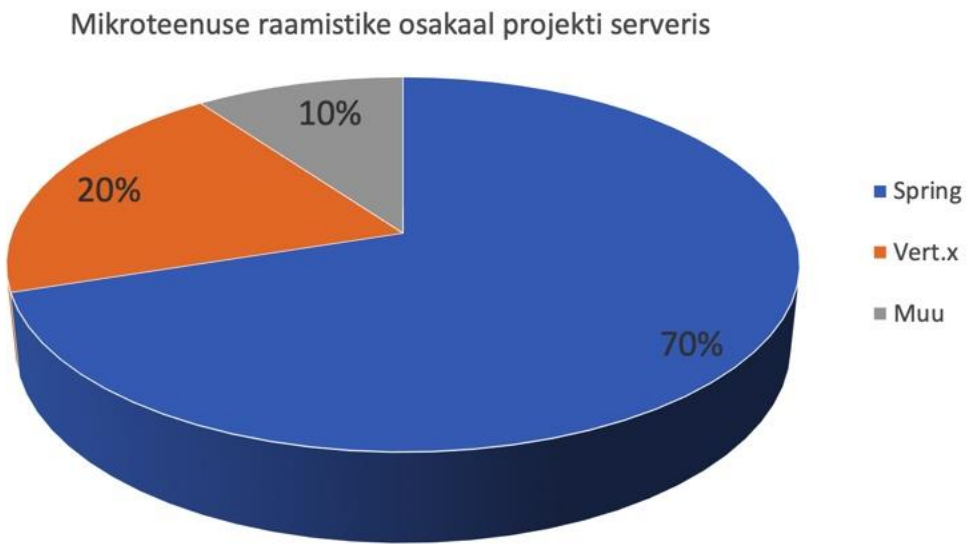
Seoses asjaoluga, et *MIT* teenus pole ettevõtte hallata, ei ole selle jälgimine samuti võimalik, kuna ligipääs nendele andmetele puudub. Ettevõttel on mikroteenuste seiramiseks võetud mitmeid lahendusi kasutusele, mis teevad selle protsessi lihtsaks.

Analüüsi osas tuuakse välja skeemid mikroteenuse ühendamisest kõikide ettevõttes kasutusel olevate jälgimisvahenditega. Selgitatakse ka välja, kuidas mõjutab nende ühendamist Micronaut raamistikule vahetamine.

3.2 Probleemid, mis ajendasid tegema teenust Micronaut raamistikul

Ettevõttes laialdaselt kasutusel olevate *Spring* raamistikul põhinevate mikroteenuste liigne mälukasutus on probleem, mis vajab lahendust [1].

Töö autor luges kokku projektis kasutatavate raamistike arvu ning tegi sellest diagrammi. Ettevõttes moodustavad mikroteenustest *Spring* raamistikul olevad teenused seitsekümmend protsenti.



Joonis 1. Erinevate raamistike kasutuste osakaal projektis.

3.2.1 Mikroteenuste liigne mälukasutus

Ettevõte on koondanud suure hulga erinevaid mikroteenuseid ühele serverile. Selleks, et vältida serveri asjatut üle koormamist ning mälumahu juurde ostmise vajadust, tuleb hakata tegema mikroteenuseid nii, et need võimalikult vähe ressursse kasutaksid. Selleks valis teenuse arendamiseks autor välja *Micronaut* raamistiku, mis on noor, kuid tuntud oma säästliku mälukasutuse poolest [6]. *Micronaut* disainiti tegemaks paljusid protsesse kompileerimise ajal [7]. Tänu sellele ei kasutata sellel raamistikul käivitusaegseid *reflections*'e, mis on peamine põhjus, miks *Micronaut* suudab mälukasutusega kokku hoida [7].

Seitsekümmend protsenti ettevõtte mikroteenuseid on *Spring Boot* raamistikule ehitatud. Autor arvab, et *Micronaut'i* kasutuselevõtt aitab säästa teenuste mälukasutust, sest

analüüsid näitavad, et *Micronaut*'i mälukasutus lihtsate mikroteenustega on kordades väiksem kui *Spring*'il [8].

3.2.2 Kasulikkus firmale

Ettevõttel on palju mikroteenuseid ning need kõik on koondatud kokku ühele serverile. Sellises olukorras tuleb mälukasutuse võit kasuks. Mälukasutuse jalajälje vähendamine lubab serverite mälu vähendada ning seekaudu selle arvelt raha kokku hoida [9].

Ettevõte, kus autor töötab, on suur toetaja rohelisele eluviisile ja väikesele keskkondlikule jalajäljele. Iga väiksemgi mälukasutuse jalajälje vähendamine aitab kaasa rohelisemale arendusele ja süsiniku jalajälje vähendamisele [2].

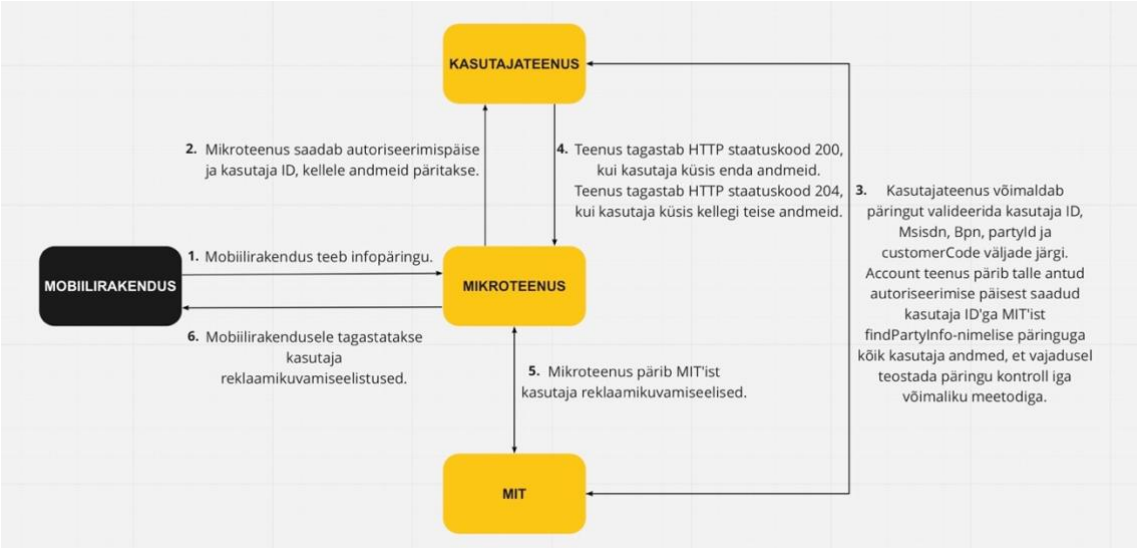
3.3 Seotud projektid

Loodaval mikroteenusel saab olema kaks temast sõltuvat mobiilirakendust. Üks on *Android* ja teine *IOS* platvormil. Mobiilirakendustele on ettevõtte kliendi poolt soov lisada funktsionaalsus, millega saaks kasutajad valida, kas ja millist tüüpi reklaame võib rakendus neile kuvada. Vajalikku infot hakkavad rakendused pärima ja muutma autori loodavalt mikroteenuselt.

Arendusprotsessi mobiilirakenduste arendajate kaasamine on ülioluline ning tagasiside tuleb küsida konstantselt, et mikroteenus teeniks oma eesmärki ja hiljem toodangus ühtegi probleemi ja arusaamatust ei esineks.

Mikroteenus saab vajalikud andmed kasutaja reklaami kuvamise eelistuste kohta ettevõtte kliendi vanast *SOAP* protokolli põhjal ehitatud andmebaasist nimega *MIT*.

Mikroteenus suhtleb lisaks mobiilirakendustele ka kasutajateenusega. Kasutajateenuse kaudu saab mikroteenus autoriseerida mobiilirakenduste poolt tulevaid päringuid. Kontrollitakse, kas kasutaja pärib enda andmeid või kellegi teise. Kasutajateenus kasutab samuti *MIT* teenust vajalike kasutajaandmete pärimiseks.



Joonis 2. Joonis, mis kirjeldab mikroteenusega seonduvaid teenuseid ja nende toimimist.

4 Lahenduse analüüs

Käesolevas peatükis analüüsitakse ettevõtte poolt seatud probleeme ja mikroteenuste mälukasutuse probleemi lahendamist *Micronaut* raamistiku abil detailsemas plaanis. Seletatakse lahti nõuded lahendusele. Kirjeldatakse projekti arhitektuurset poolt, tehnilisi valikuid ja *Micronaut* raamistikule ülemineku eeldusi. Räägitakse ka *Micronaut*'ile üleminekust ja teenuse jälgimise protsessist.

4.1 Nõuded lahendusele

Autorile esitati mitmeid nõudeid loodava mikroteenuse osas. Teenus peab lahendama turvalisuse, andmemudeli ja teenuse jälgimise probleemid.

Autor ise võttis lisaprobleemiks veel leida alternatiivne raamistik *Spring*'ile, millel oleks temast väiksem mälukasutuse jalajälg, et vähendada serverite mälukasutust.

Micronaut raamistikule ülemineku eeldused ettevõtte poolt – uue raamistiku kasutusele võtt peab olema põhjendatult parem juba kasutusel olevast *Spring* raamistikust. Samuti peab tegemist olema raamistikuga, mis on pidevas uuendamises ka tulevikus. See väldib probleemi, et tulevikus turvaauke ei tekiks.

Mikroteenusele lisatav autoriseerimislahendus peab tagama, et mobiilirakendusest sinna päringut tulev päring oleks autoriseeritud. Põhinõue on valideerida, et päringu teinud kasutaja küsiks enda *ID*'le kuuluvat infot.

Mikroteenusest tagastatav andmemudel peab olema *JSON* formaadis ning sisaldama kõiki ülesandes kirjeldatud väljanimede- ja struktuurimuudatusi. Mobiilirakendused ei tohi midagi teada *SOAP* päringutest ega *XML*'ist.

4.2 Micronaut raamistiku analüüs

Käesolev peatükk analüüsib *Micronaut* raamistikku. Tehakse ülevaade *Micronaut*'ist ja tema omadustest. Selgitatakse, miks valiti mikroteenuse mälu probleemide lahendamiseks just see raamistik.

4.2.1 Ülevaade

Micronaut on vaba ja avatud tarkvara arendamise raamistik, mida kasutakse, et ehitada modulaarseid, kergelt testitavaid ja *serverless* (arendaja ei pea tegelema serveri infrastruktuuriga) rakendusi [11].

Micronaut'i arendustööd algasid 2017 aastal, kui *OCI* (*Object Computing Inc.*) tahtis luua/ehitada intelligentset kompileerijat (kompilatsiooni tegema), mis eelarvutas raamistiku infrastruktuuri, et saada sama produktiivsed kasumid/tulemused, mis *Spring*, aga väiksema käivitusajaga [11].

Esimest korda lasi *OCI Micronaut*'i välja 2018 aasta mais ja kahe aasta jooksul on *Micronaut* on suurelt jaolt arenenud tehnoloogiaks, mis on ümber defineerinud ootused sellele, kuidas seda tüüpi raamistikud muudavad tähtsaid omadusi nagu näiteks teenuse käivitamise aeg ja mälukasutus mikroteenustel ja *serverless* rakendustel [11].

Micronaut välistab vajaduse kasutada keerukaid käivitusaegseid *Java* tehnoloogiaid, et saada selliseid funktsioone nagu näiteks *AOP* (*Aspect-Oriented Programming*), JTA (Java *Transaction Management)*, *caching*. *Micronaut* on näidanud, et mikroteenuste tulevikuks on intelligentsed kompilaatorid ja väiksemad, kiiremad käivitusajad [11].

Algusest peale on *Micronaut JVM*'i (*Java Virtual Machine*) kogukonnas nautinud tohutut entusiasmi ja toetust. Näiteks on 2020 aasta 24 juuli seisuga *Micronaut*'il *GitHub*'is 3,8 tuhat tähte ja üle 200 kaasautori. *Micronaut*'i kasutatakse *Alibaba*, *Target*, *Minecraft*, *Boeing* ja paljude teiste organisatsioonide tootmises. *Micronaut* on litsentseeritud liberaalse *Apache 2* (populaarseim veebiserveri tarkvara) litsentsi alusel [11].

OCI võttis 2020 aastal oma juhtimise alla uue mittekasumlikel eesmärkidel loodud *Micronaut Foundation* organisatsiooni. Selle peamisteks eesmärkideks on jätkata tasuta ja avalikult kasutatavat *Micronaut* raamistiku ja kindlustada, et selle arendus on

jätkusuutlik ja innovaatiline. Sinna organisatsiooni kuuluvad nimekad inimesed ja oodatakse, et *Micronaut*'i tulevik saab olema veelgi huvitavam ja innovatiivsem [11].

Micronaut esindab olulist sammu edasi mikroteenuste raamistike hulgas *JVM* jaoks. Ta toetab kõiki võimalusi, mida *Java* arendajad teavad ja armastavad, näiteks *DI* (*Dependency Injection*) ja *AOP*, ilma tegemata kompromisse teenuse käivitumise aegade, jõudluse ja mälu tarvitamise osas [12].

Micronaut konkureerib traditsiooniliste raamistike vastu AOT (*ahead-of-time*) kompilatsiooniga [12]. *AOT* kompilatsioon on üks viisidest *Java* rakenduste võimekuse ja *JVM* käivitamisaegade parandamiseks [13].

4.2.2 Kompileerimisaegsed protsessid

Üks suuremaid erinevus *Micronaut*'il võrreldes teiste raamistikega on, et paljud olulised protsessid tehakse ära kompileerimise ajal. Tänu sellele minimaliseeritakse *reflection*'ite (peegeldamiste) kasutamist käivitusaja hetkel. *Java reflection* on vahend, mis lubab *Java* rakendusel koodi jooksutamise käigus ennast uurida ja analüüsida, et vajadusel manipuleerida enda käitumist [14]. Peegelduste liigsel kasutamisel kahjustavad nad teenuse jõudlust [15]. Peegeldused rikuvad abstraktsioone ja võivad platvormi täiendustega teenuse käitumist muuta [15].

Java reflection API (koodi käivitumisaegse uurimise ja muutmise liides) on olnud kasutuses juba mõnda aega. Seda on kasutatud teekide ja raamistike poolt läbi *Java* ökosüsteemi [12].

Suurem osa moodsaid raamistikke kasutavad reflektsioonipõhiseid lähenemisi *endpoint*'ide (puutepunkt API ja temaga suhtlevate teenuste vahel) kutsumiseks, *proxy*'de loomiseks, andmete sidumiseks(*data binding*), konfiguratsiooni lugemiseks ja muud. See on märkimisväärne, kui palju dünaamilisi, *reflection* loogikat eksisteerib tüüpilises modernses *Java* rakenduses, arvestades, et keel on staatiliselt kirjutatud [12].

Kahjuks *reflection API* kasutamisel on miinused nii mälukasutuses kui käivitusaja jõudluses [12]. Nende põhjusteks on:

- Kuna *Java*'s ei ole *reflection*il *cache*'t, seega *reflective* info lugemine on kallis. Iga *library* (teek) ja raamistik toodab unikaalset *reflection* vahemälu. Tüüpilises

modernses Java rakenduses on igal raamistikul eraldi *cache*, mis teeb ülimalt keeruliseks mälukasutuse optimeerimise reflektsioonide kasutamisel [12].

- *Reflective* kutsed on palju keerulisemad *JIT*'ile (*Just-In-Time*) optimeerida. *Invoke Dynamic* (abivahend, mille eesmärk on suurendada dünaamilist tüüpi keelte tuge *JVM*'il) ja teised abivahendid üritavad seda leevendada, kuid *reflective* kutsed ei saa kunagi nii efektiivseks [12].

- Enamus raamistikke ei ole teadlikud mitmekesisest keelte ökosüsteemist *JVM* peal. Keeled nagu *Groovy* ja *Kotlin* toodavad rohkem meetodeid *byte* koodi tasemel kui *Java*, et toetada keelelisasid. See vahe tähendab, et *reflection cached* on *Groovy* ja *Kotlin*'i baasil klassidel suuremad, kuna *byte* kood sisaldab rohkem meetodeid, väljasid ja muud [12].

AOT tunnuselemendid:

- Kompileerimisaegne tugi *Swagger*'ile (*OpenApi*). *Swagger* on tööriist *API*'de kirjeldamiseks. Suurem osa raamistike integreerivad *Swagger*'it käivitusajal, mis kasutab *reflection*'i baasil lähenemisi, mis on nii võimsuse kui mälukasutuse perspektiivist halb lahendus [12].

- Kompileerimisaegne valideerimine. Kõik kaasaegsed *Java* raamistikud omavad mitmeid päritud reegleid, mida arendaja ei saa rikkuda. Kui *Java* teeb neid kontrolle kompileerimise ajal, siis enamus kaasaegseid raamistike seda ei tee. Nad sõltuvad käivitusaja *erinditest* (arvutiprogrammi töös tekkinud eriolukordi), et teatada arendajale, et tema kood on vale [12].

- Annotatsioonide *mappimine* kompileerimise ajal. Erinevalt traditsioonilistest *Java* raamistikest nagu *Spring*, mis teevad analüüse raamistiku tasemel annotatsioonidest käivitusajal, teeb *Micronaut* raamistik teeb samu analüüse, aga kompileerimise ajal. See võimaldab teenuse käivitamist kiirendada ja mälukasutust vähendada [12].

4.2.3 Üleminek Spring Boot raamistikult Micronaut'ile

Micronaut raamistik on tugevalt inspireeritud *Spring boot*'ist. Raamistik on üles ehitatud järgmina samu arendusvõtteid, et arendajatel oleks lihtne raamistikku vahetada. See

hõlmab annotatsioone, *DI* ja *convention over configuration* (vaikimisi standardkonfiguratsioon liigse kodeerimiseta) [16].

4.2.4 Mõõdikute analüüs Micronaut'il võrreldes teiste teenustega

Käesolevas peatükis uuritakse raamistike mälukasutuste ja kiiruste analüüse. Vaatluse peamine rõhk on *Spring*'i tulemustel, kuna see tahetakse *Micronaut*'i vastu vahetada.

Analüüsid on tehtud Graeme Rocher'i poolt, kes on üks *Micronaut*'i loojatest. Autor põhjendab võrdluse tegemist sellega, et kuigi nad on varasemalt julgustanud arendajaid ise võrdlusi tegema, on need võrdlused põhjendamatult valet informatsiooni jaganud ning *Micronaut*'i meeskond ei ole kuidagi suutnud samu tulemusi saada. Seetõttu üritas Rocher teha võimalikult põhjaliku analüüsi kolme raamistiku, 2020 aasta aprillikuu seisuga, uusimate versioonide kohta: *Micronaut* 2.0, *Quarkus* 1.3.1 ja *Spring Boot* 2.3 [17].

Raamistike testiti *iMac Pro* 2017 aasta mudelil. Sellel on 2.3 GHZ 8 tuumaline *Intel Zeon* protsessor ja 32GB *RAM*'i (muutmälu). Operatsioonisüsteemiks on *MacOS*. Kasutatakse *Java JDK* (*Java Development Kit*) 14 [8].

Lihtsa *unit test*'i (väikse osa rakendusest testimine) jooksutamiseks kulus keskmiselt aega *Micronaut'il* 374 millisekundit (edaspidi ms), *Quarkus*'el 965 ms, *Spring*'il 942 ms. *Micronaut* võidab siin märkimisväärselt [8].

Ühe *controller*'iga (teenusesse tulevate päringute töötleja) *REST API* rakenduse kompileerimisele keskmiselt kuluv aeg *Micronaut'il* 1.5 sekundit, *Quarkus*'el 1.43 sekundit ja *Spring Boot*'il 1.33 sekundit. *Spring Boot* võidab, sest ta on ainuke, kellel puuduvad kompileerimisaegsed protsessid [8].

Testide sooritamine võttis keskmiselt aega *Micronaut'il* 4.15 sekundit, *Quarkus*'el 5.8 sekundit ja *Spring Boot*'il 7.4 sekundit. *Micronaut* võidab, kuna kasutab ainult *Micronaut* süsteemi *classloader*'it (*Java Runtime Environment* osa, mis dünaamiliselt laeb Java klassid *JVM*'sse) testideks. Teised raamistikud kasutavad keerukamaid tööriistu [8].

Rakenduse käivitamisaeg keskmiselt *Micronaut'il* 440 ms, *Quarkus*'el 893 ms ja *Spring Boot*'il 960 ms. *Micronaut* võidab. Igal raamistikul on talle spetsiifilised lipud, mis kiirendavad arendamisstaatuses rakenduse käivitamist. Seega täpsem on mõõta kiirusi

tehes *JAR* fail ja käivitada rakendus päriselt. Nii ei ole spetsiaalseid *JVM* lippe kiiremaks käivituseks ja võrdlus on võrdsem [8].

Puhas *Java* rakenduse käivitamine ilma *JVM* lippudeta võttis keskmiselt aega *Micronaut'il* 507 ms, *Quarkus*'el 680 ms ja *Spring Boot*'il 1270 ms. Siin võidab jälle *Micronaut*, kuigi kõikidel raamistikel on *laziness*'i (käivitumise strateegia, kus mingit funktsionaalsust käivitatakse vaid vajadusel) [8].

Käesolev test on tähtsaim. Testitakse aega esimese vastuseni. Kasutatakse *Node.js* (võrgurakenduste kirjutamise platvorm) *skript*'i, mis mõõdab aega rakenduse käima panemisest kuni esimese vastuse tagastamiseni. *Micronaut'il* kulus aega 978 ms, *Quarkus*'el 900 ms ja *Spring Boot*'il 1880 ms. Siin kohal *Quarkus* on võitja, kuid vahe ei ole midagi märkimisväärset võrreldes *Micronaut'iga*. Vahe *Spring Boot*'iga võrreldes on suur, pea üks sekund [8].

Järgmisena tehakse koormustest. Serverile, kus rakendused jooksevad, on antud mälu 128 mb. *Micronaut* suutis vastata kaheksakümnele tuhandele päringule sekundis, *Quarkus* seitsmekümne viiele tuhandele ja *Spring Boot*'i kohta andmed puuduvad, kuna *skript*'il ei õnnestunud *keep alive* päringute andmeid salvestada [8].

Teenuse mälukasutuse mõõtmine peale koormuse tekitamist. *Micronaut* 'i mälukasutus oli 290 megabitti (edaspidi Mb), *Quarkus*'el 394 Mb ja *Spring Boot*'il 481 Mb. Selle testi võidab selgelt *Micronaut*. jääb 100 Mb võrra *Micronaut'ile* alla. *Spring Boot*'i mälukasutus ületab suurelt *Micronaut'i* [8].

Viimaseks testiks on koormustest 18 Mb mäluga serveril. *Micronaut* ja *Quarkus* saavad hakkama, kuid *Spring Boot*'iga tuli "*Run out of memory*" [8].

Nagu tulemustest näha, siis *Micronaut* jääb vähesel määral alla *Quarkus*'ele esimese vastuse andmise ajaga käivitusest ning *Spring*'ile kompileerimisajaga [8]. Ülejäänud testides ületab *Micronaut* teisi raamistike [8].

OpenJDK 14 on 2019 iMac Pro Xeon 8 Core. Winner in Red.

METRIC	MICRONAUT 2.0 M2	QUARKUS 1.3.1	SPRING 2.3 M3
Compile Time ./mvn clean compile	1.48s	1.45s	1.33s
Test Time ./mvn test	4.3s	5.8s	7.2s
Start Time Dev Mode	420ms	866ms (1)	920ms
Start Time Production java -jar myjar.jar	510ms	655ms	1.24s
Time to First Response	960ms	890ms	1.85s
Requests Per Second (2)	79k req/sec	75k req/sec	??? (3)
Request Per Second -Xmx18m	50k req/sec	46k req/sec	??? (3)
Memory Consumption After Load Test (-Xmx128m) (4)	290MB	390MB	480MB
Memory Consumption After Load Test (-Xmx18m) (4)	249MB	340MB	430MB

(1) Verifier Disabled
(2) Measured with: ab -k -c 20 -n 10000 http://localhost:8080/hello/John
(3) Spring WebFlux doesn't seem to support keep alive?
(4) Measured with: ps x -o rss,vsz,command | grep java

Joonis 3. Graeme Rocher'i raamistike võrdlus [17]

Sõltuvalt raamistike mälukasutuste ja jõudluste ülevaadetele on lõputöö autor veendunud, et ettevõttes ülekaalu omavate *Spring Boot* raamistikel asuvate mikroteenuste asendamine *Micronaut*'iga omab kindlalt märgilist arengut serverite koormuse vähendamises. Lahenduse peatükis tuuakse välja ka võrdlused uue *Micronaut* teenuse ja teise samaväärse *Spring Boot* teenuste vahel.

Autor valis teenuse kirjutamisel *Quarkus*'e asemel *Micronaut*'i, kuna *Micronaut*'i on ettevõttes varem üksikutel teenustel kasutatud ja tegu ei ole töötajatele nii võõra raamistikuga. Samuti ei ole mälukasutuse võit *Quarkus*'el piisavalt suur, et seda põhjendatult eelistada.

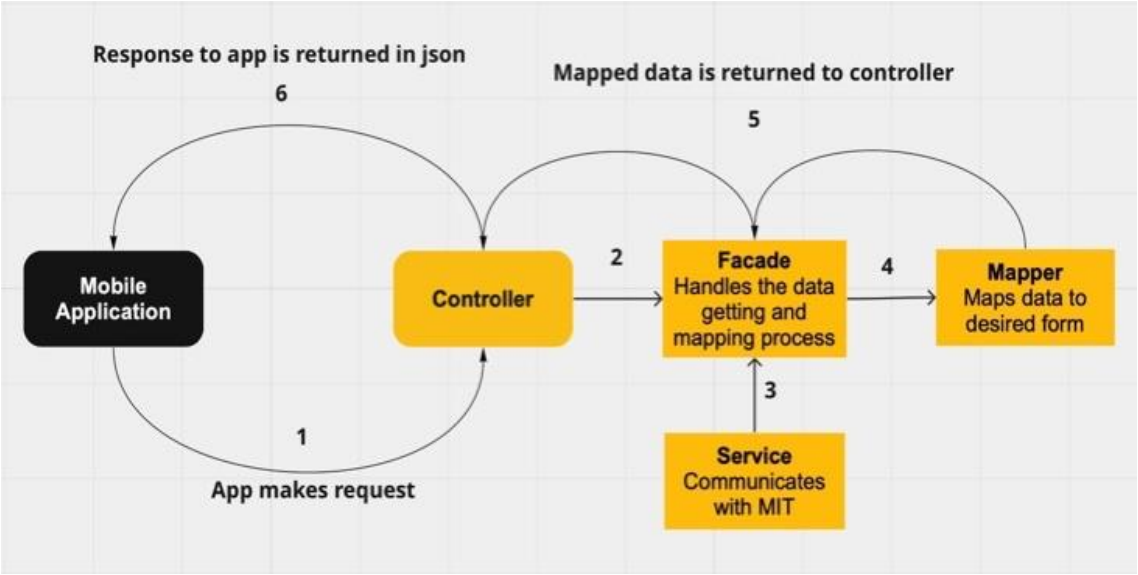
4.3 Koodi arhitektuuri analüüs ja valik

Käesolevas peatükis seletatakse lahti, milliseid valikuid tehti disaini ja arhitektuuri osas.

4.3.1 Arhitektuur

Teenuse arhitektuuriline pool on visualiseeritud all oleval joonisel. Teenusel on *controller* klass, mis tegeleb mobiilirakendustest tulevate päringutega. See klass suhtleb *facade* klassiga, mis tuleneb *Facade* disainimustrist. *Facade* disainimuster peidab kogu teenuse keerukuse enda taha [18]. Läbi tema suheldakse *MIT* teenusega, kust saadakse

vajalikud andmed ning *mapper* klassiga, kus töödeldakse valmis soovitud andmemudel. Töödeldud andmed saadetakse läbi *controller*'i tagasi mobiilirakendusele.



Joonis 4. Mikroteenuse päringu läbiviimise protsess

4.3.2 REST API disain

Teenus kirjutati *REST API* disaini põhjal. *REST API* disain on maailmas populaarne ning ehitatud kestma igavesti [19]. *REST API* oma tulevikukindluses ja populaarsuses osutus kindlaks valikuks teenuse kirjutamisel.

On olemas alternatiivseid valikuid *REST API*'le, kuid väikesele mikroteenusele ei ole nendel midagi rohkemat pakkuda [19].

4.4 Tehnoloogia valik

Käesolevas peatükis seletatakse lahti projekti tehnilised valikud.

4.4.1 Java

Teenus kirjutatakse *Java*'s versioonil 11. Java programmeerimiskeele valiku põhjus on, et projekti raamistikuks valitud *Micronaut* on *Java*, *Kotlin*'i ja *Groovy* jaoks ehitatud raamistik [20]. Samuti on *Java* ettevõtte projektis juba varem kõikides teenustes kasutusel, mistõttu on loogiline samm sellega jätkata.

Kasutatakse *Java* 11 versiooni, mis on hetkel viimane *LTS* (pikaaegse toega) versioon [21]. See on projekti kirjutamise ajahetkel ettevõttes soovitatud *Java* versioon.

4.4.2 Micronaut

Töö autor valis katsetatavaks raamistikuks *Micronaut*'i, sest selle loojad on esitanud tugevaid väiteid nende mälukasutuse säästlikkuse kohta [17]. *Micronaut*'i tiim on teinud muljetavaldavat tööd seoses *reflection*ite kasutamise minimaliseerimisega ja kompileerimisaegse *DI*'iga [22]. Samuti on *Micronaut* ehitatud järgima samu arendusvõtteid nagu *Spring*, mis teeb temale ülemineku lihtsamaks [16].

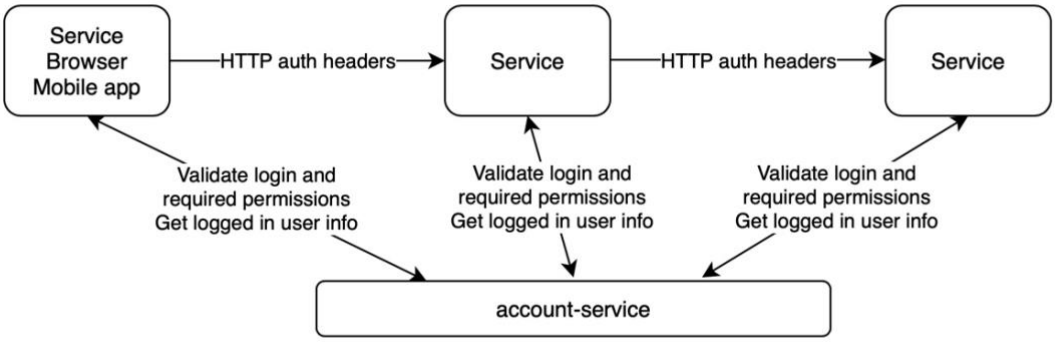
Vastavalt 4.2.4 peatükis tehtud raamistike võrdlustele tõestas Micronaut ka oma mälukasutuse säästlikust.

4.4.3 SOAP

Teenuses kasutatakse *SOAP* 'iga seotud lisamooduleid, (moodul on struktuur, mis on tervik paljudest väiksematest osadest [23]) kuna *MIT* põhineb *SOAP* protokollil. Selleks on vaja moodulit, mis suudaks etteantud *WSDL* faili põhjal luua *MIT*'i päringu jaoks vastava funktsionaalsusega klassid [4]. Genereeritud klassid on vajalikud, et teha päringuid *MIT*'i ja sealt tulevat vastust kasutada.

4.5 Autoriseerimine

Autoriseerimisprobleemi lahendamiseks implementeeritakse teenuses juba firmas mitmete teiste mikroteenuste poolt kasutuses olev *account-service*-nimelist (edaspidi kasutajateenus) autoriseerimisteenust. Kasutajateenus on avalik teenus, mis tagastab infot sisse logitud kasutajast ja üksustest, kuhu tal ligipääs on. Tegemist on lihtsa *REST API* teenusega, kust loodav mikroteenus iga päringu puhul hakkab valideerima, kas kasutaja, millega andmeid päriti, pärib ikka enda kasutajaandmeid ja kas tal on nendele juurdepääs. Kui juurdepääs on olemas, tagastab teenus *HTTP* (*Hypertext Transfer Protocol*) staatuskoodi 200, vastasel juhul 403 *Not authorized* ning päring jäetakse pooleli. Kasutajakontekst saadetakse teenuste vahel edasi *sessioon-token*'ina (sessiooni valideerimiseks spetsiaalne väärtus) *HTTP header*'ina (*API* päringu metaandmete kandja).



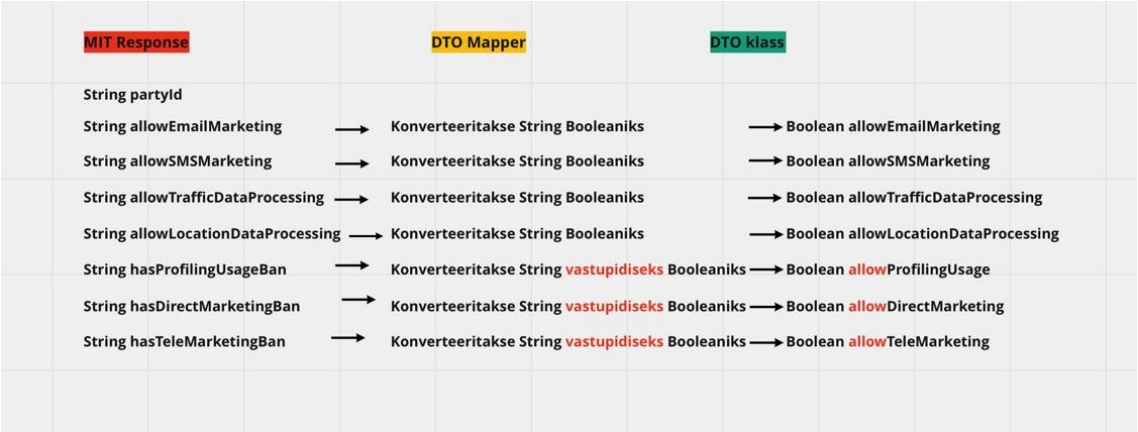
Joonis 5. Kasutajateenuse autoriseerimise protsessi kirjeldav joonis.

4.6 Andmemudeli parandamine

MIT on ehitatud *SOAP* protokolli põhjal. See tähendab, et kõik *MIT* 'i minevate päringute funktsionaalsus on ära kirjeldatud spetsiaalses *WSDL* failis [4]. *WSDL* faili järgi vajalike klasside genereerimiseks on tarvis teenusele lisada spetsiaalne moodul, mis seda teeks. Kui kõik vajalikud klassid päringu tegemiseks on genereeritud, saadakse info edukalt mikroteenusesse.

Käesolevas lõigus kirjeldatakse *MIT*'ist saadud andmete ümber muutmise protsessi nõutud vormingusse. Luuakse *mapper* klass, kuhu antakse saadud andmed ning kus on kogu loogika andmemudeli muudatusteks. Antud klassi väljundiks on *DTO* (*Data Transfer Object*) klass, milles on andmed muudetud kujul. Antud *DTO* klass tagastatakse mobiilirakendustele *JSON* formaadis.

Käesoleva lõigu all asuval joonisel on näidatud, kuidas *MIT* tagastab vastuses *Boolean*'id *String* objektidena (tähemärkide järjestusest koosneb objekt). *DTO mapper* klassis konverteeritakse *String*'id *Boolean* objektideks. Punasena on näidatud andmemudeli muudatused võrreldes esialgse mudeliga. Väljad "*hasProfilingUsageBan*", "*hasDirectMarketingBan*" ja "*hasTeleMarketingBan*" teisaldatakse vastupidiseks *Boolean*'iks, kuna uues andmemudelis on nende nimed vastupidise tähendusega.



Joonis 6. *DTO mapper*'i näide andmemudeli muudatuste kohta.

Info muutmise päringu puhul mobiilirakenduse poolt saadetakse mikroteenusele *JSON* formaadis andmeväljad, mida on vaja muuta. Päringu vorm pannakse kokku ja saadetakse *MIT*'i.

Info muutmise õnnestumise korral on mobiilirakendusel vaja küsida *backend*'ilt uuesti kasutaja reklaami kuvamise eelistusi, et teha kindlaks info uuenemine *MIT*'is. Vältides, et mobiilirakendused peaks info muutmisel topelt päringuid tegema, tagastab loodud mikroteenus ise peale info muutmist värske seisu mobiilirakendusele. See vähendab päringute hulka ja säästab serveri koormust ning kiirust.

4.7 Teenuse jälgimine

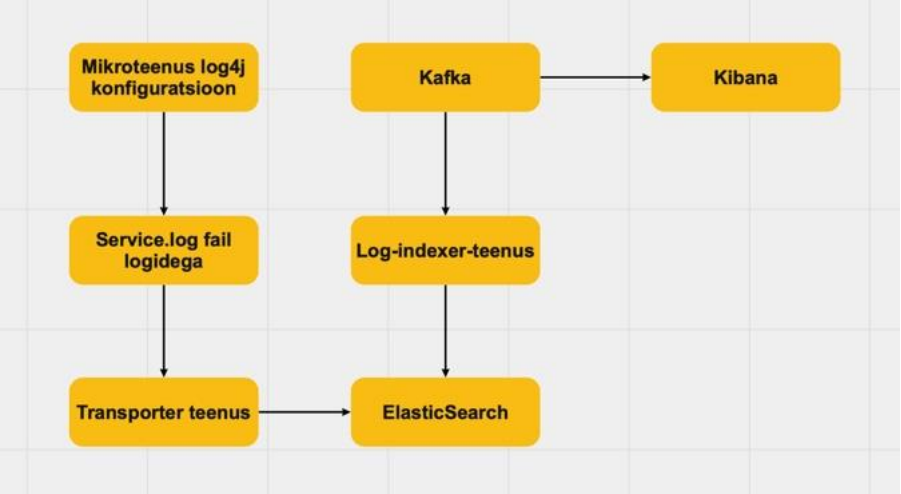
Selles peatükis kirjeldab autor ettevõttes kasutusel olevaid teenuste jälgimisvahendeid ja analüüsib, kas *Micronaut* raamistikul oleva teenuse ühendamine nendega nõuab lisasamme.

Ettevõte kasutab mikroteenuste jälgimiseks kolme erinevat tööriista: logide jälgimiseks *Kibana*'t, mõõdikute jälgimiseks *Grafana*'t ja rakenduskonteinerite jooksutaja ja manageerijana *Kubernetes*'t.

4.7.1 Ettevõtte monitoorimise tehnoloogia integreerimine uue raamistikuga ja võrdlus vanadega

Kibana, *Grafana* ja *Kubernetes*'e ühendamine teenusega ei hõlma lisapingutusi võrreldes teiste raamistikega, kuna nende seadistamine ei olene raamistikust. Käesolev peatükk kirjeldab, kuidas on jälgimisvahendid mikroteenusega ühenduses.

Mikroteenus läbi log4j teegi lisab logid faili *service.log*. Antud fail saadetakse läbi transport teenuse *Elastic Search*'i. *Elastic Search*'i info võetakse Kibana's lahti ja kuvatakse kasutajale [24].



Joonis 7. Mikroteenuse ja *Kibana* ühendamine

Grafana on kasutajaliides, mis võimaldab jälgida teenust [25]. Loodud mikroteenus on ühendatud läbi *Influx Client Library Influx* andmebaasiga, mis saadab mikroteenuse jälgitud infot *Grafana*'le [26].

Joonis 8. Mikroteenuse ja *Grafana* ühenduse joonis.

Kubernetes'el on seos *pods*'ide. *Pod* on väikseim juurutatav üksus arvutis. Ta sisaldab ühte või mitut *container*'it [27]. *Container* on keskkond, kuhu saab rakenduse koodi pakkida koos kõigi sinna kuuluvate teekide ja sõltuvustega, mis annab võimaluse tal isoleeritud keskkonnas töötada [28]. *Pods*'idele kehtivad *Kubernetes'e* poolt defineeritud piirangud ning võimalused [27].

Joonis 9. Mikroteenuse ja Kubernetes'e ühendus.

Seega on kõik jälgimisvahendid neutraalsed ning ei ole vahet, millisel raamistikul oleva mikroteenusega ühendada.

5 Lahenduse realiseerimine

Käesolevas peatükis võetakse lahti mikroteenuse kirjutamise protsess. Räägitakse, kuidas pidasid paika seatud hüpoteesid ning mis probleeme esines.

Hüpotees oli, et arhitektuurialase osa kirjutamine uuel raamistikul ei tekita ühtegi probleemi. Teenus kirjutati *REST API* arhitektuuril ning seda ei mõjutanud kuidagi raamistiku erinevused. *REST API* arhitektuuri implementeerimine *Micronaut* raamistikul õnnestus probleemideta, kuna *Micronaut* raamistik on inspireeritud *Spring Boot*'ist, mis tähendab, et ta on arendatud kasutama samu arendusvõtteid nagu näiteks *DI*, annotatsioonid ja *convention-over-configuration* [16]. Tänu sellele ei olnud REST API arhitektuuri implementeerimine võrreldes Spring'iga oluliselt erinev.

Mikroteenuse kirjutamisel kasutati *Java*'t versiooninumbril 11.

Mikroteenus kirjutati *Micronaut* raamistikul.

MIT'ist *SOAP* päringutega andmete pärimiseks vajalikud *WSDL* faili järgi klasside genereerimiseks kasutati järgmisi mooduleid: *jaxb-xjc, jaxb-api, jaxws-api, cxf-rt-wsdl* ja *jws-api*.

Suurimaks raskuskohaks oli *Micronaut*'i noorus. Tänu asjaolule, et raamistik on veel üsna uus ja vähe kasutatud, leidub selle kohta internetis vähe materjale ja lahenduskäike. Seetõttu mõne probleemiga hätta jäädes satuti probleemi ette, kus abiks oli vaid raamistiku dokumentatsioon ja leidlikkus.

5.1 REST API

Teenusele lisati kaks *endpoint*'i. Üks neist kasutab *HTTP GET* meetodi ja teine *HTTP PUT* meetodi.

HTTP GET meetodi kasutatakse teenuselt info küsimiseks. Õnnestunud *GET* päringutele tagastatakse andmed tavaliselt *XML* või *JSON* formaadis ja *HTTP* staatuskood 200 (*OK)*. Ebaõnnestunud päringud tagastavad enamasti *HTTP* staatuskoodi 404 (*Not foun*d) või

400 (*Bad request)*. *HTTP* disaini kohaselt on *GET* päringud mõeldud ainult andmete küsimiseks, mitte nende muutmiseks. See teeb *GET* päringu turvaliseks päringuks [29].

Loodud teenus tagastab *GET* päringuga info *JSON* formaadis. Ebaõnnestunud päringute võimalikud staatuskoodid on: 400 (*Bad request*) (tagastatakse juhul kui tehtav päring ei ole korrektne.), 401 (*Not authenticated)* (tagastatakse juhul kui päringuga ei anta kaasa autoriseerimiseks vajalikke andmeid), 403 (*Forbidden*) (tagastatakse juhul kui päringu autoriseerimine ei õnnestu), 404 (*Not found*) (tagastatakse juhul, kui seoses päringu kasutajaga ei leidu *MIT*'is andmeid või kui päritud aadress ei ole antud hetkel saadaval) ja 500 (*Exception*) (tagastatakse juhul, kui päringu protsessimise käigus juhtub mõni ootamatu viga).

Tagastatav *JSON* andmemudel on all joonisel näidatud.

```
{
    "data": {
        "isCampaignEnabled": true,
        "allowEmailMarketing": true,
        "allowSMSMarketing": true,
        "allowTrafficDataProcessing": true,
        "allowLocationDataProcessing": true,
        "allowProfilingUsage": true,
        "allowDirectMarketing": true,
        "allowTeleMarketing": true
    }
}
```

Joonis 10. Loodud teenuse tagastatav *JSON* formaadis andmemudel *GET* ja *PUT* päringule.

HTTP PUT meetodi kasutatakse info uuendamiseks. Sellele päringule antakse kaasa päringu keha, mis sisaldab soovitud muudatusi. Aadress, kuhu päring tehakse, sisaldab muudetava objekti *ID*'d, millega määratakse, millise objekti infot muudetakse [29].

Vahel kasutatakse *PUT* päringut ka uute andmete loomiseks. Sellisel juhul antakse kaasa päringu keha, kuid aadress ei sisalda enam kindlale objektile viitavat *ID*'d [29].

Õnnestunud päringule tagastatakse *PUT* puhul *HTTP* staatuskood 200 või 204, kui tagastatakse tühi vastus. Kui *PUT*'i kasutatakse uue info loomiseks, siis tagastatakse õnnestunud päringule *HTTP* staatuskood 201. Päringu vastuse sisu on valikuline, see võib olla ka tühi [29].

PUT ei ole turvaline päring, kuna see muudab või loob serveris olevaid andmeid. *PUT* päringud on idemponentsed. See tähendab, et kui *PUT* päringuga muuta või luua uusi andmeid, siis sama päringut uuesti tehes jääb serveris olev info selliseks nagu ta esimesel päringul tehti ja ei uuendata seda topelt. *PUT* päringute idemponentsetena hoidmine on turvalisuse pärast rangelt soovituslik [29].

Loodud teenuse *PUT* meetod tagastab õnnestunud päringu puhul staatuskoodi 200 *OK*, kuna meetod ei tagasta tühja keha, vaid muudetud andmete värske seisu. Ebaõnnestunud päringu võimalikud staatuskoodid ühtivad *GET* meetodi omadega.

PUT päringule saadetav päringu keha näide on näidatud all joonisel.

```
{
    "allowEmailMarketing": true,
    "allowSMSMarketing": true,
    "allowTrafficDataProcessing": true,
    "allowLocationDataProcessing": true,
    "allowProfilingUsage": true,
    "allowDirectMarketing": true,
    "allowTeleMarketing": true
}
```

Joonis 11. Loodud teenuse *PUT* päringu keha.

5.2 Micronaut raamistiku implementeerimine

Käesolevas peatükis kirjeldatakse, kuidas õnnestus *Micronaut* raamistiku kasutamine mikroteenuse ehitamisel.

5.2.1 Teenuse esmase seadistamise hüpoteesi paikapidavus

Leitud materjalide ja teiste arendajate kogemuste põhjal püstitati hüpotees, et *Micronaut* raamistikul on uue mikroteenuse esmane seadistus lihtne ning vähe aeganõudev. Töö protsess näitas, et see nii ka oli. Teenuse esmane ülesehitus on lihtne. Teenuse esmane loomine on võimalik vaid ühe käsuga ja kõik minimaalselt vajalikud moodulid on sellega seadistatud. Kui *Micronaut* on arvutile paigaldatud, siis tuleb kirjutada *terminali* käsk, mis on käesoleva lõigu all välja toodud, et luua tühi *Micronaut* projekt [30].

```
mn create-app my-demo-app
```

Joonis 12. Käsk tühja Micronaut projekti loomiseks.

Lisaks terminali käsule on *Micronaut* projekti loomiseks olemas ka *UI* liides (User Interface) [31].

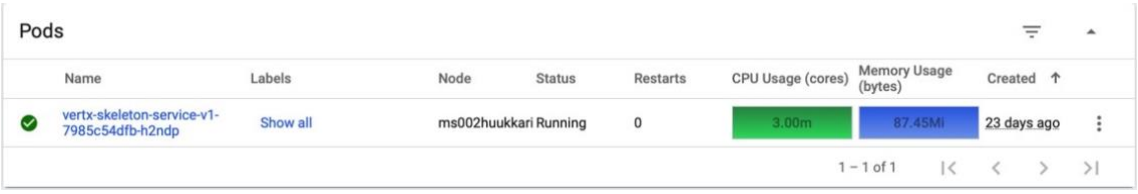
REST API teenuse ehitamine *Micronaut* raamistikul ei valmistanud raskusi, kuna *Micronaut* on ehitatud üles järgima samu arendusvõtteid, mida *Spring Boot* [22]. *REST API* arhitektuuriga seotud konfigureerimine ja annotatsioonide määramine ei erine oluliselt *Spring Boot*'ist [16], [32].

5.2.2 Mõõdikute võrdlus ettevõtte serveris olevate Micronaut, Spring ja Vert.x skeleton teenustega

Käesolevas peatükis võrreldakse ettevõtte serveris paigaldatud *Micronaut*, Spring Boot ja *Vert.x* raamistikul olevaid mikroteenuseid. Kõik teenused on *skeleton* teenused, mis on *REST API* teenused ning omavad kolme *endpoint*'i. Neid teenuseid ei kasutata ja nad töötavad ilma koormuseta. Tegemist on serveril kasutusena seisvate teenustega.

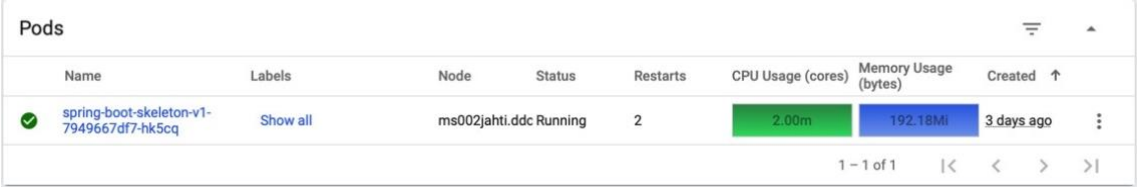
Mainitud teenustel võrreldakse nende *CPU* (protsessori) hõivatust ja mälukasutust passiivses olekus. Mõõdikute andmed on võetud läbi teenusega ühendatud *Kubernetes*'est. Saadud andmed annavad hea ülevaate raamistike jalajäljest.

Vert.X skeleton'i mälukasutus on kolmest raamistikest kõige väiksem. Mälukasutus on 88 *mebibyte* (edaspidi Mi). CPU kasutus on 3 *millicpu*'d.



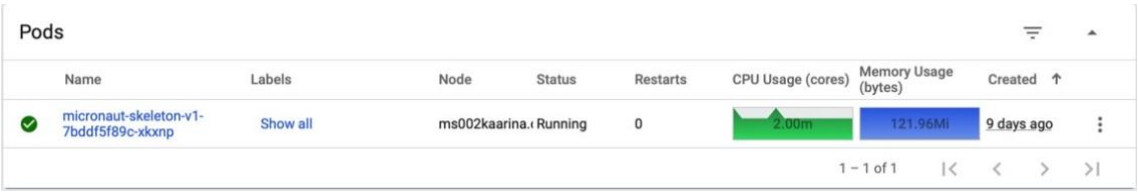
Joonis 13. *Vert.x* raamistiku *skeleton*'i *pod*'i ülevaade *CPU*- ja mälukasutusest.

Spring Boot'i *skeleton*'i mälukasutus on kõige suurem. See on 192 Mi. *CPU* kasutus on 2 *millicpu*'d.



Joonis 14. *Spring Boot* raamistiku *skeleton*'i *pod*'i ülevaade *CPU*- ja mälukasutusest.

Micronaut'i *skeleton*'i mälukasutus jääb teisele kohale. Ta edestab *Spring Boot*'i raamistikku 70 Mi'ga, kasutades mälu 122 Mi. *CPU* kasutus on 2 *millicpu*'d.



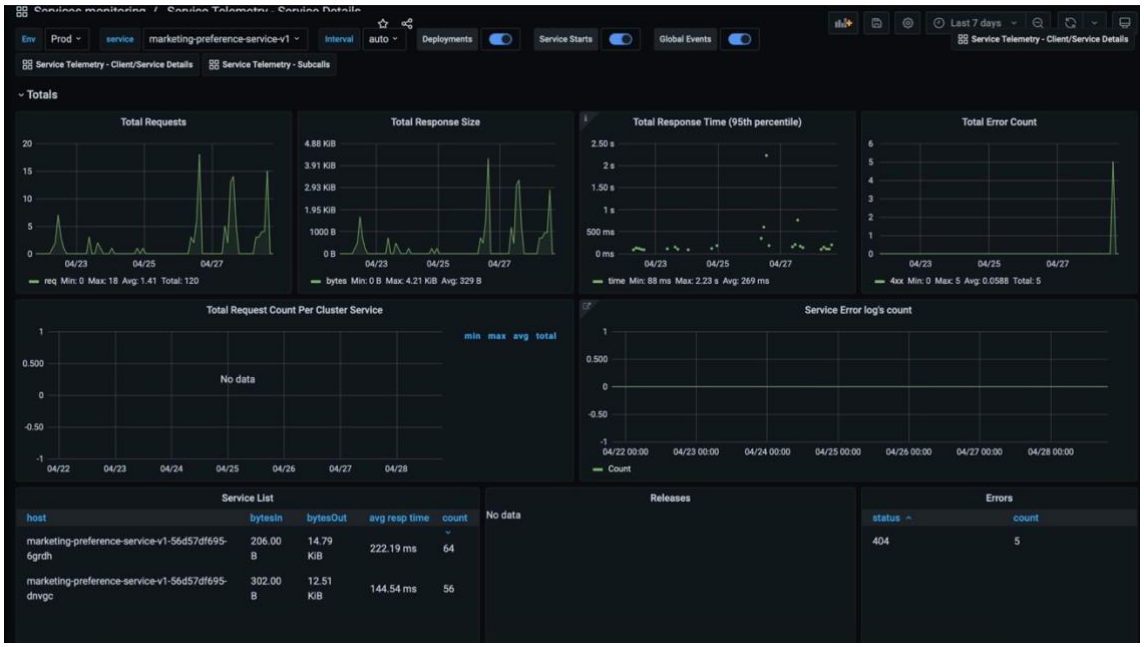
Joonis 15. *Micronaut* raamistiku *skeleton*'i *pod*'i ülevaade *CPU*- ja mälukasutusest.

Kuigi analüüsi tulemused näitavad, et *Vert.x* jalajälg on *Micronaut*'ist parem, teda teenuse kirjutamisel ei kasutatud. Põhjuseks on, et *Micronaut* on *Spring Boot*'iga võimalikult lähedane [16]. Samuti ei ole *Vert.x* puhul tegemist raamistikuga, vaid abivahendiga JVM rakenduste kirjutamisel [33].

5.2.3 Monitoorimise lisamise protsessi kirjeldus

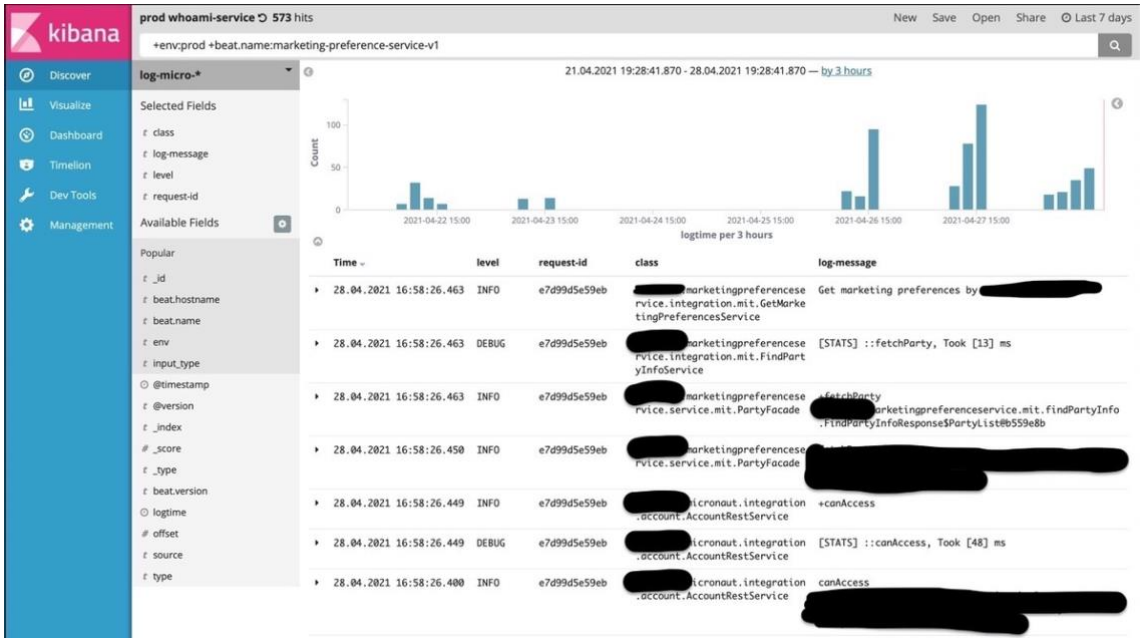
Peatükis 4.7.1 leiti, et teenust jälgivate tööriistade lisamine ei sõltu raamistikust. Loodud mikroteenusega ühendati edukalt kõik ettevõtte kasutatavad teenuse jälgimisvahendid. Siin peatükis on lisatud piltide abil visuaalne näide, kuidas teenust jälgitakse.

Grafana on avatud lähtekoodiga lahendus teenuse jooksvate andmete analüüsimiseks. Selle abil on võimalik teenuse mõõdikuid pärida, visualiseerida, hoiatada rikete eest olenemata sellest, kus need on salvestatud [25]. Lõigu all on visuaalne näide loodud mikroteenuse *Grafana dashboard*'ist. Andmete jõudmine teenusest *Grafana*'ni on lahti seletatud peatükis 4.7.1.



Joonis 16. Loodud mikroteenuse *Grafana* näide.

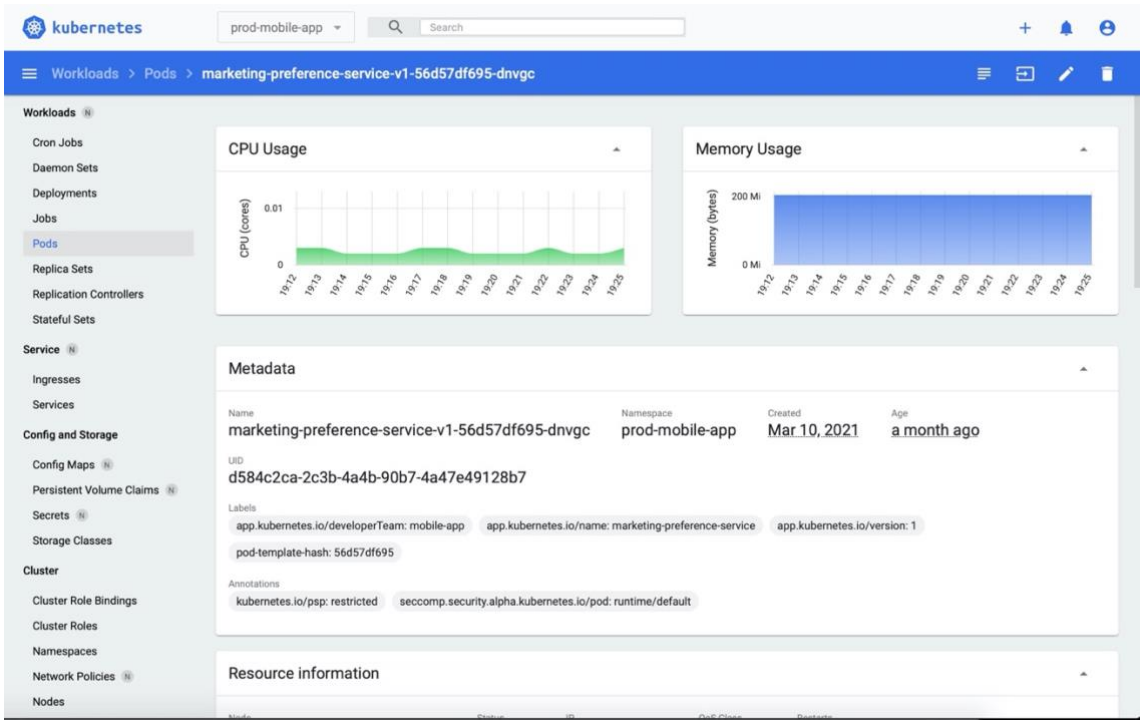
Kibana on avatud kasutajaliides, mis võimaldab otsida, vaadata ja visualiseerida teenuse logisid. All on näide loodud mikroteenuse logidest, mille *Kibana* on teenuselt saanud [34]. Logide jõudmine teenusest *Kibana*'ni on lahti seletatud peatükis 4.7.1.



Joonis 17. Loodud mikroteenuse *Kibana* logide näide.

Kubernetes on platvorm konteineriseeritud töökoormuste ja teenuste haldamiseks, mis hõlbustab nii deklaratiivset seadistamist kui ka automatiseerimist. All on näide loodud

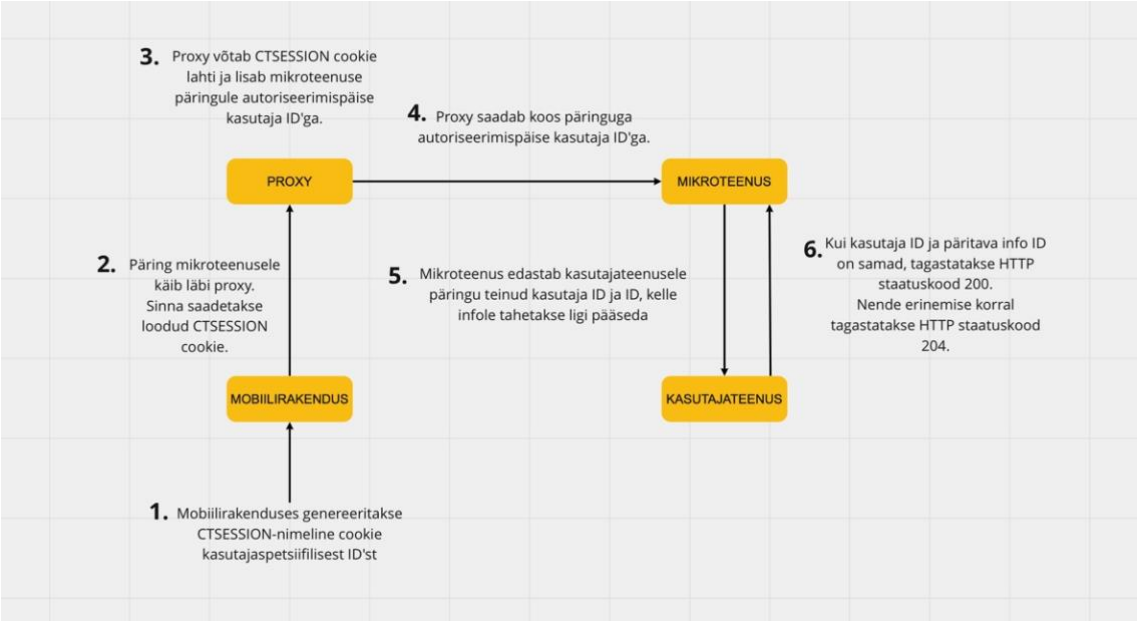
mikroteenuse *Kubernetes*'e *dashboard*'ist [35]. Andmete jõudmine teenusest *Kubernetes*'eni on lahti seletatud peatükis 4.7.1.



Joonis 18. Loodud mikroteenuse *Kubernetes*'e *dashboard*'i näide.

5.3 Autoriseerimisprobleemi lahendamine

Autoriseerimisprobleemi lahenduse realiseerimine oli autori jaoks väike ülesanne. Ettevõttel on juba varasemalt loodud kõikide mikroteenuste jaoks ühine autoriseerimisteenus. Selle teenuse käest info saamise teekond on lahti kirjutatud all olevas joonises.



Joonis 19. Mobiilirakendusest autoriseerimise info jõudmine mikroteenuseni.

Autoriseerimisteenuse kasutuselevõtt nõudis järgmist: üldine kood, et implementeerida autoriseerimisteenuse vastuseid, edasi on *REST API* teenuse käest info küsimine ja sinna saatmine.

5.4 Andmemudeli sobivuse probleemi lahendamine

SOAP'iga seotud teemad, mobiilirakendustele sobitamine, boilerblate (korduvkasutatud) koodi vähendamine mobiilirakendustes.

Mikroteenuse poolt *MIT*'ist *SOAP* päringuga info pärimine oli tänu *WSDL* faili lugemise lisamoodulitele raskusi mittevalmistav. *MIT*'ist läbi *SOAP* päringu pärimise andmeklassid on kirjeldatud kõik spetsiaalses *WSDL* failis. See sisaldab kõike vajalikku infot: aadress ja keskkond, kuhu päring teha ja päringute andmemudelid. Selle *WSDL* faili järgi *Java* klasside loomiseks on olemas eraldi moodul. Tuleb *Gradle* (avatud lähtekoodiga projekti ehitamise automatiseerimise tööriist) konfiguratsiooni failis lisada *WSDL ja binding* failide asukohad ning projekti ehitamisel genereeritakse automaatselt kõik päringu jaoks vajalikud klassid. Pärast seda on vaja õigeid abiklasse kasutades kutsuda välja päringu tegemise meetod. Vastus tuleb *MIT*'ist *XML*'is, kuid meie teenus tänu *WSDL* faili genereeritud klassidele suudab selle *XML* vastuse konverteerida *Java* klassiks ning sealt edasi teenus ise juba töötleb infot selliseks nagu ette nähtud ning tagastatakse mobiilirakendustele *JSON* formaadis.

Tänu sellele on kogu andmemudeli muutmine mikroteenuses ja mobiilirakendustele tagastatakse andmed juba muudetud kujul.

Hoides andemudeli muutmise loogika mobiilirakenduste asemel mikroteenuses, välditakse uute muudatuste korral suuremat tööd. Uuendusi tehakse ainult mikroteenuses ning mobiilirakendused ei pea andmemudeliga tegelema. Kui see lüli jääks mikroteenusest välja, siis peaks mõlemad mobiiliplatvormid eraldi andmemudeli muutmiseks tööd tegema.

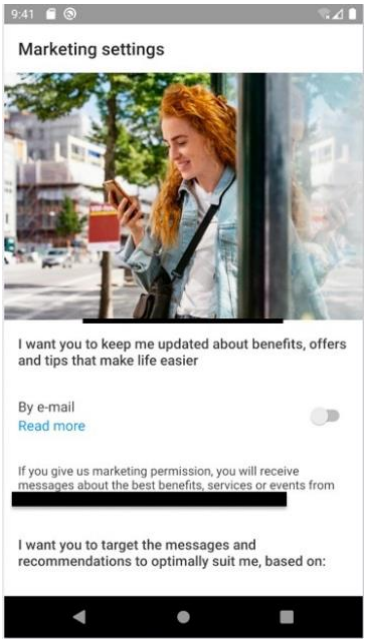
6 Tulemused

Käesolevas peatükis räägitakse lõputöö praktilises osas valminud mikroteenusest. Näidatakse ekraanikuvasid loodud teenuse kasutusest mobiilirakendustes, arutatakse teenuse tulevikust ja näidatakse, kuidas *Micronaut*'i kasutusele võtmine teenuse jalajälge muutis.

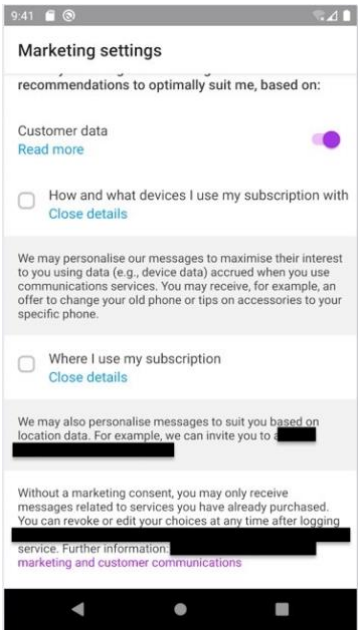
6.1 Loodud lahendus ja selle kasutus mobiilirakendustes

Lõputöö tulemusena valmis vähese mälukasutusega täisfunktsionaalne *REST API* mikroteenus *Micronaut* raamistikul. Teenus on peale valmimist võetud aktiivselt kasutusesse mõlema mobiiliplatvormi rakenduste poolt.

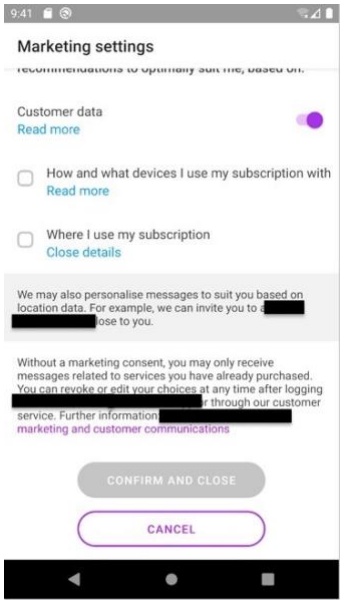
Järgmised kuvatõmmised näitavad, kuidas mobiilirakenduses on kuvatud vaade, kus on kasutajal võimalik muuta oma reklaami kuvamise eelistusi. Antud vaate kuvamisel küsib mobiilirakendus mikroteenuselt kasutaja eelistused, kuvab need vastavates lahtrites ning nende muutmisel saadetakse uus päring teenusele, et eelistusi *MIT*'is muuta.



Joonis 20. Vaade mobiilirakenduses 1.



Joonis 21. Vaade mobiilirakenduses 2.



Joonis 22. Vaade mobiilirakenduses 3.

6.2 Teenuse tulevik

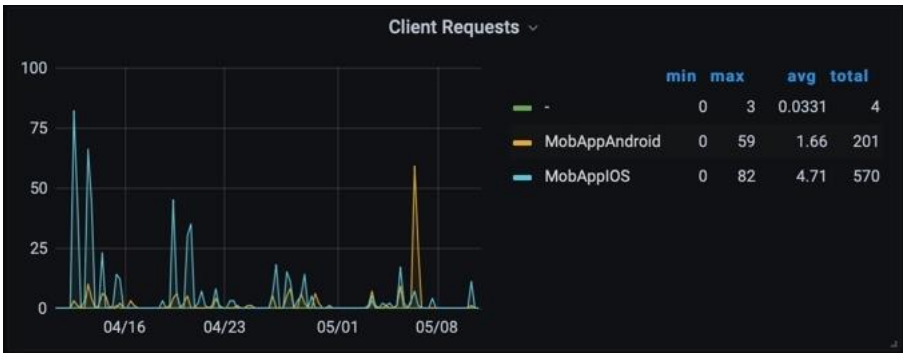
Antud lõputöö raames valminud mikroteenust kasutatakse lõputöö valmimise hetkel saja viiekümne tuhande kasutaja poolt kasutatavates *IOS* ja *Android* mobiilirakendustes igapäevaselt. Loodud mikroteenus jääb mainitud mobiilirakenduste kasutada nende eluea lõpuni, kuna tegemist on funktsionaalsusega, mida antud hetkel ei ole plaan rakendustest eemaldada. Antud teenust tuleb hoida edaspidi ajakohasena, et vältida igasuguseid

turvariske. Samuti tuleb olla valmis selleks, et tulevikus võib tekkida vajadus andmemudeli muudatusi sisse viia. Suure tõenäosusega sellega teenuse tulevik piirdub ja puudub aktiivne arendus.

6.3 Teenuse hõivatus ja mälukasutus

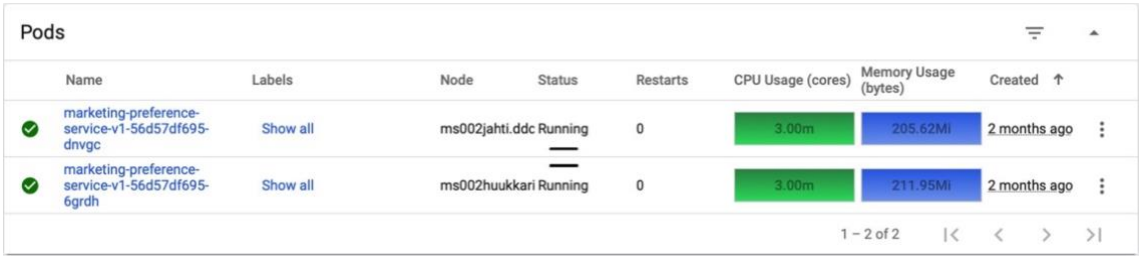
Käesoleva lõputöö tulemuste kirjutamise ajaks on valminud mikroteenus kuu aega kasutuses olnud. Käesolevas peatükis võtame ette mõned jooksvad andmed loodud teenusest.

Viimase 30 päeva jooksul on mobiilirakendustelt tehtud päringuid teenusele 775 korda. 201 neist kuulub *Android*'i mobiilirakendusele ja 570 *IOS*'ile. 4 päringut on tehtud arendajate poolt ja ei lähe kirja mobiiliplatvormina.

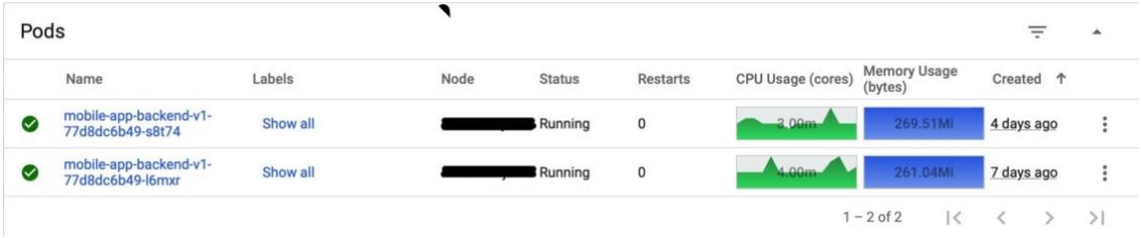


Joonis 23. Kuu aja päringute hulk mobiilirakendustelt mikroteenusele.

Joonis 24 on näidatud loodud mikroteenuse *CPU*- ja mälukasutust. Joonis 25 on näidatud samaväärse *Spring Boot* raamistikul oleva mikroteenuse andmeid. Selle põhjal saab öelda, et mälukasutuse võit sellise väikese teenuse pealt oli 60 - 70 Mb. *CPU* kasutus *Spring Boot* teenusel oli ühel *pod*'il võrreldes *Micronaut*'i 3 *millicpu*'ga 4 *millicpu* peal.



Joonis 24. Loodud mikroteenuse *CPU*- ja mälukasutus.



Joonis 25. Loodud mikroteenusega samaväärne *Spring Boot* raamistikul oleva teenuse *CPU*- ja mälukasutus.

7 Kokkuvõte

Töö põhieesmärgiks oli luua *REST API* mikroteenus Micronaut raamistikul. Eesmärgiks oli luua projekti kuuluvatele mobiilirakendustele mikroteenus, mis lahendaks ettevõtte poolt esitatud autoriseerimise ja andmemudeliga seonduvad probleemid. Teiseks eesmärgiks oli saada *Micronaut* raamistikul kogemusi ja analüüsida, kas raamistik võtab vähem mälu, kui ettevõttes enim kasutatud *Spring Boot*. Pikemas perspektiivis oli eesmärk saada *Micronaut* raamistikuga piisavalt kogemusi, et alustada selle juurutamist ettevõttesse mikroteenuste jalajälje parandamiseks.

Töö tulemusena loodi *REST API* mikroteenus *Micronaut* raamistikul *Java* programmeerimiskeeles. Teenuse jälgimiseks kasutati *Kibana*'t, *Kubernetes*'t ja *Grafana*'t. Loodud mikroteenus on lõputöö valmides aktiivselt kasutuses *IOS* ja *Android* platvormi mobiilirakendustes, millel on kokku üle saja viiekümne tuhande kasutaja. Teenus tagastab mobiilirakendustele infot nende kasutajate reklaami kuvamise eelistuste kohta ja pakub võimalust neid ka muuta.

Micronaut raamistiku analüüsimisel oli rõhk tema võrdlemisel *Spring Boot*'iga, kuna seda raamistikku tahetakse tulevikus asendama hakata. Võrreldi nende mälukasutuse jalajälge, protsessori kasutust ja teenuse vastamiskiirust. Lõpptulemusena kasutusesse läinud *Micronaut* raamistikul teenuse võrdlemisel samaväärse *Spring Boot* teenusega kinnitati analüüsi osas püstitatud hüpoteesi, et *Micronaut* raamistiku mälu- ja *CPU*-kasutus on *Spring Boot*'ist väiksem.

Loodud teenus lahendas ettevõtte poolt esitatud autoriseerimise ning andmemudeli probleemid ja samuti optimeeris mikroteenuse mälukasutust.

Kasutatud kirjandus

[1] Matthias Graf, „Which Java Microservice Framework Should You Choose in 2020? “, 2020. [Võrgumaterjal]. Available: https://betterprogramming.pub/which-java-microservice-framework-should-you-choose-in-2020-4e306a478e58. [Kasutatud 24. aprill 2021].

[2] Sanjay Podder, Adam Burden, Shalabh Kumar Singh, Regina Maruca, „How green is your software?“, 2020. [Võrgumaterjal]. Available: https://hbr.org/2020/09/how-green-is-your-software. [Kasutatud 28. aprill 2021].

[3] Omkarchalke, „Basics of SOAP – Simple Object Protocol“, 2019. [Võrgumaterjal]. Available: https://www.geeksforgeeks.org/basics-of-soap-simple-object-access-protocol/. [Kasutatud 24. aprill 2021].

[4] Ashushrma378, „Difference Between SOAP and WSDL“, 2020. [Võrgumaterjal]. Available: https://www.geeksforgeeks.org/difference-between-soap-and-wsdl/. [Kasutatud 24. aprill 2021].

[5] Steven Davelaar, „Performance Study - REST vs SOAP for Mobile Applications“, 2015. [Võrgumaterjal]. Available: https://www.ateam-oracle.com/performance-study-rest-vs-soap-for-mobile-applications. [Kasutatud 28. aprill 2021].

[6] Margaret Rouse, „Micronaut Framework“, 2021. [Võrgumaterjal]. Available: https://searchapparchitecture.techtarget.com/definition/Micronaut-framework. [Kasutatud 28. aprill 2021].

[7] Zachary Klein, „Micronaut: A Java Framework for the Future, Now“, 2018. [Võrgumaterjal]. Available: https://objectcomputing.com/resources/publications/sett/july-2018-micronaut-framework-for-the-future. [Kasutatud 28. aprill 2021].

[8] Graeme Rocher, „Micronaut 2.0 vs Quarkus 1.3.1 vs Spring Boot 2.3 Performance on JDK 14“, 2020. [Videomaterjal].

Available: https://www.youtube.com/watch?v=rJFgdFIs_k8. [Kasutatud 24. aprill 2021].

[9] Graham Charters, „Microservice Performance That Saves You Money“, 2020. [Võrgumaterjal]. Available: https://dzone.com/articles/microservice-performance-that-saves-you-money. [Kasutatud 28. aprill 2021].

[11] Jeff Scott Brown, „The Future of Micronaut“, 2020. [Võrgumaterjal]. Available: https://www.techwell.com/techwell-insights/2020/07/future-micronaut. [Kasutatud 25. aprill 2021].

[12] Graeme Rocher, „Micronaut 1.0 RC1 and the Power of Ahead-Of-Time compilation“, 2018. [Võrgumaterjal]. Available: https://micronaut.io/2018/09/30/micronaut-1-0-rc1-and-the-power-of-ahead-of-time-compilation/. [Kasutatud 25. aprill 2021].

[13] Michael Krimgen, „Ahead of Time Compilation (AoT)“, 2020. [Võrgumaterjal]. Available: https://www.baeldung.com/ahead-of-time-compilation. [Kasutatud 29. aprill 2021].

[14] Glen McCluskey, „Using Java Reflection“, 1998. [Võrgumaterjal]. Available: https://www.oracle.com/technical-resources/articles/java/javareflection.html. [Kasutatud 29. aprill 2021].

[15] GeeksforGeeks, „Reflection in Java“, 2017. [Võrgumaterjal]. Available: https://www.geeksforgeeks.org/reflection-in-java/. [Kasutatud 29. aprill 2021].

[16] Desosa, „Micronaut – A Perfect Microservice Framework?“, 2020. [Võrgumaterjal]. Available: https://desosa.nl/projects/micronaut/2020/04/09/microservices-in-detail.html#fn:website-micronaut. [Kasutatud 10. mai 2021].

[17] Graeme Rocher, „Micronaut vs Quarkus vs Spring Boot Performance on JDK 14“, 2020. [Võrgumaterjal]. Available: https://micronaut.io/2020/04/07/micronaut-vs-quarkus-vs-spring-boot-performance-on-jdk-14/. [Kasutatud 25. aprill 2021].

[18] GeeksforGeeks, „Facade Design Pattern“. [Võrgumaterjal].
Available: https://www.geeksforgeeks.org/facade-design-pattern-introduction/. [Kasutatud 16. mai 2021].

[19] Bill Doerrfeld, „Is REST Still a Good API Design Style to Use?“, 2019. [Võrgumaterjal]. Available: https://nordicapis.com/is-rest-still-a-good-api-design-style-to-use/. [Kasutatud 28. aprill 2021].

[20] Micronaut documentation, „Introduction“, 2020. [Võrgumaterjal].
Available: https://docs.micronaut.io/latest/guide/index.html. [Kasutatud 25. aprill 2021].

[21] Oracle, „Oracle Java SE Support Roadmap", 2021. [Võrgumaterjal].
Available: https://www.oracle.com/java/technologies/java-se-support-roadmap.html. [Kasutatud 15. mai 2021].

[22] Michael Pratt, „Introduction to Micronaut Framework“, 2019. [Võrgumaterjal].
Available: https://www.baeldung.com/micronaut. [Kasutatud 25. aprill 2021].

[23] Tartu Ülikooli arvutiteaduse instituut Programmeerimise Alused 2, „Mis on mood ul?“, 2016/2017. [Võrgumaterjal].
Available: https://courses.cs.ut.ee/2016/alused2/fall/Main/PART4A. [Kasutatud 14. mai 2021].

[24] ElasticSearch Service Documentation, „Enable Logging and Monitoring".
[Võrgumaterjal]. Available: https://www.elastic.co/guide/en/cloud/current/ec-enable-logging-and-monitoring.html. [Kasutatud 15. mai 2021].

[25] Grafana documentation, „What is Grafana“. [Võrgumaterjal]. Available: https://grafana.com/docs/grafana/latest/getting-started/. [Kasutatud 29. aprill 2021].

[26] Grafana documentation, version 7.5.x,
„Getting Started With Grafana and InfluxDB". [Võrgumaterjal].
Available: https://grafana.com/docs/grafana/latest/getting-started/getting-started-influxdb/. [Kasutatud 15. mai 2021].

[27] Kubernetes documentation, „Pods", 2021. [Võrgumaterjal].
Available: https://kubernetes.io/docs/concepts/workloads/pods/. [Kasutatud 15. mai 2021]

[28] Google Cloud documentation, „Containers 101: What are Containers?“. [Võrguma terjal]. Available: https://cloud.google.com/containers. [Kasutatud 10. mai 2021].

[29] Rest Api Tutorial, „Using HTTP Methods for RESTful Services". [Võrgumaterjal] . Available: https://www.restapitutorial.com/lessons/httpmethods.html.
[Kasutatud 15. mai 2021].

[30] Micronaut documentation SNAPSHOT versioon, „Micronaut CLI“.
[Võrgumaterjal]. Available: https://docs.micronaut.io/snapshot/guide/index.html#cli.
[Kasutatud 27. aprill 2021].

[31] Object Computing, „Micronaut Launch". [Võrgumaterjal].
Available: https://micronaut.io/launch. [Kasutatud 15. mai 2021].

[32] Tioman Gally, „Micronaut vs Spring Boot", 2020. [Võrgumaterjal].
Available: https://medium.com/@Tioman/micronaut-vs-spring-boot-b1f2c05bb9b
[Kasutatud 15. mai 2021]

[33] Alain Lompi,
„Building asynchronous and reactive applications with Eclipse Vertx", 2018.
[Võrgumaterjal]. Available: https://blog.senacor.com/an-introduction-to-asynchronous-and-reactive-programming-with-eclipse-vert-x/. [Kasutatud 15. mai 2021]

[34] Elasticsearch , „What is Kibana Used for“. [Võrgumaterjal]. Available: https://ww w.elastic.co/what-is/kibana. [Kasutatud 29. aprill 2021].

[35] Kubernetes documentation, „What is Kubernetes?“. [Võrgumaterjal].
Available: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/.
[Kasutatud 29. aprill 2021].

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

Mina, Kristjan Mäeots

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "REST API mikroteenuse disain ja arendamine *Micronaut* raamistikus ettevõtte näitel", mille juhendaja on German Mumma
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

16.05.2021

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