Kauno technologijos universitetas

Informatikos fakultetas

Prieigos valdymo metodas mikropaslaugų architektūroje

Baigiamasis magistro projektas

Donatas Kukta

Projekto autorius

Prof. Algimantas Venčkauskas

Vadovas

2022, Kaunas

Kauno technologijos universitetas

Informatikos fakultetas

Prieigos valdymo metodas mikropaslaugų architektūroje

Baigiamasis magistro projektas

Informacijos ir informacinių technologijų sauga (6211BX008)

Donatas Kukta

Projekto autorius

Prof. Algimantas Venčkauskas

Vadovas

Prof. Agnius Liutkevičius

Recenzentas

Kaunas, 2022

Kauno technologijos universitetas

Informatikos fakultetas

Donatas Kukta

Prieigos valdymo metodas mikropaslaugų architektūroje

Akademinio sąžiningumo deklaracija

Patvirtinu, kad:

1. baigiamąjį projektą parengiau savarankiškai ir sąžiningai, nepažeisdamas kitų asmenų autoriaus ar kitų teisių, laikydamasis Lietuvos Respublikos autorių teisių ir gretutinių teisių įstatymo nuostatų, Kauno technologijos universiteto (toliau – Universitetas) intelektinės nuosavybės valdymo ir perdavimo nuostatų bei Universiteto akademinės etikos kodekse nustatytų etikos reikalavimų;

2. baigiamajame projekte visi pateikti duomenys ir tyrimų rezultatai yra teisingi ir gauti teisėtai, nei viena šio projekto dalis nėra plagijuota nuo jokių spausdintinių ar elektroninių šaltinių, visos baigiamojo projekto tekste pateiktos citatos ir nuorodos yra nurodytos literatūros sąraše;

3. įstatymų nenumatytų piniginių sumų už baigiamąjį projektą ar jo dalis niekam nesu mokėjęs;

4. suprantu, kad išaiškėjus nesąžiningumo ar kitų asmenų teisių pažeidimo faktui, man bus taikomos akademinės nuobaudos pagal Universitete galiojančią tvarką ir būsiu pašalintas iš Universiteto, o baigiamasis projektas gali būti pateiktas Akademinės etikos ir procedūrų kontrolieriaus tarnybai nagrinėjant galimą akademinės etikos pažeidimą.

Donatas Kukta

Patvirtinta elektroniniu būdu

Donatas Kukta. Prieigos valdymo metodas mikropaslaugų architektūroje. Magistro baigiamasis projektas / vadovas prof. Algimantas Venčkauskas; Kauno technologijos universitetas, Informatikos fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): Informatikos inžinerija (Informatikos mokslai)

Reikšminiai žodžiai: mikropaslauga, prieiga, valdymas.

Kaunas, 2022. 89 p.

Santrauka

Tarp pasiskirsčiusių modulių yra sudėtinga užtikrinti saugų ir efektyvų prieigos valdymo mechanizmą. Ši problema programuotojų bendruomenėje yra vis dar nagrinėjama ir ieškoma efektyvių, bet saugių sprendimų. Šio projekto tikslas – palengvinti prieigos valdymą mikropaslaugų architektūroje sukuriant metodą, kuris būtų saugus, efektyvus ir lengvai pritaikomas.

Išanalizavus moksliniuose šaltiniuose pateikiamas mikropaslaugų architektūros savybes ir nagrinėjamus skirtingus prieigos valdymo metodus, pastebėtos dvi skirtingos strategijos kurios grindžiamos išorine centralizuota autorizacija ir vidinio perimetro sauga. Todėl nuspręsta suprojektuoti tokį metodą, kuris apjungia šias dvi strategijas naudojantis išoriniais ir vidiniais žetonais. Šių žetonų mainuose dalyvauja autentifikavimo, autorizavimo, užklausų valdytojo mikropaslaugos ir verslo posistemė. Toks sprendimas leidžia atskirti žetonus, kuriuos valdo klientas nuo tų, kurių reikia verslo posistemei autorizuoti užklausas. Šis metodas ne tik lengvai plečiamas ir pritaikomas, bet ir suteikia galimybę užklausas autorizuoti remiantis ne tik kliento leidimais, bet ir bendra sistemos būkle, papildomomis taisyklėmis.

Realizuotas prototipas išpildo esmines siūlomo prieigos valdymo metodo funkcijas, kurios realizuotos *Service Fabric* platformoje. Tarpinių eksperimentų metu pastebėta, jog ši platforma turi greitaveikos problemą – oficialioje *Microsoft* dokumentacijoje rekomenduojamą naudoti atvirkštinį tarpinį serverį, kuris leidžia patogiau pasiekti klasteryje esančias mikropaslaugas. Pritaikius optimizaciją, kuri nenaudoja pastarojo infrastruktūrinio elemento, prieigos valdymo režimas parodė dvigubai geresnius rezultatus. Siūlomo centralizuoto prieigos valdymo metodo tyrimo rezultatai palyginami su decentralizuotu režimu. Jie identifikavo, kad siūlomas metodas yra 7% efektyvesnis, nei decentralizuotas. Tačiau esminis skirtumas tarp jų yra specifinis panaudojimo atvejis, kai verslo posistemės mikropaslaugos siunčia bent keletą vidinių autorizuotų užklausų. Nustatyta, kad tokiu atveju siūlomo prieigos valdymo metodo užklausų apdorojimo laikai yra 30% trumpesni, nei decentralizuoto.

Donatas Kukta. Access Control Method in Microservice Architecture. Master's Final Degree Project / supervisor prof. Algimantas Venčkauskas; Faculty of Informatics, Kaunas University of Technology.

Study field and area (study field group): Informatics Engineering (Computing)

Keywords: microservice, access, control.

Kaunas, 2022. 89 p.

Summary

Implementation of effective and robust access control in distributed modules is a challenging puzzle to solve. To this day software engineering community is searching for most optimal solution for this matter. Therefore, the main goal of this project is to ease access control in microservice-based architecture by creating a method which is secure, effective, and easily adaptable.

Analysis of scientific papers identified two main strategies for managing access in microservice architecture which are based on either central authorization or security of inner perimeter. Therefore, it was decided to create a method which merges these both strategies by using external and internal tokens for access management. Proposed method incorporates authentication, authorization, request manager and business microservices for specific token flow. This solution allows separation of concerns where the client of the system is unaware and has no control of internal authorization flow, which improves both usability and security of the system. In addition to that, this method can conveniently base request authorization not only on user permissions, but also on general state of the environment or any other additional rules.

Implementation of proposed access control prototype fulfills its essential attributes and is using Service Fabric platform. However, during interim experiments on the platform an efficiency fault was discovered within the reverse proxy component, which is suggested to use by official Microsoft documentation. This matter had a need of creating and applying an optimization, which avoids using default reverse proxy for internal cluster communication altogether. After doing so, performance tests concluded that optimized communication is twice faster than using reverse proxy. Regarding proposed access control method, it is 7% more efficient than the decentralized variant of the prototype. However, the main uniqueness of suggested method is located within one specific use case, where business microservice sends authorized requests internal to its cluster. In such case optimized and centralized variant processes requests 30% faster than decentralized variant.

Turinys

Paveikslų sąrašas

Lentelių sąrašas

Santrumpų ir terminų sąrašas

Santrumpos:

IDT – Identiteto Teikėjas.
DB – Duomenų Bazė.
MP – Mikropaslauga.
API (angl. *Application Programming Interface*) – Programuojama Aplikacijos Sąsaja .
JWT (angl. *Json Web Token*) – kriptografinis žetonas.
EdDSA (angl. *Edwards-curve Digital Signature Algorithm*) – kriptografinio parašo algoritmas.
DNS (angl. *Domain Name System*) – domeno vardų paslauga.
SDK (angl. *Software Development Kit*) – programinės įrangos kūrimo įrankių rinkinys.
REST (angl. Representational State Transfer) – architektūrinis stilius, kuris padeda nuosekliai apibrėžti žiniatinklio API.
ACID (angl. *Atomic, Consistent, Isolation, Durable*) – esminės reliacinės duomenų bazės savybės: atomiškumas, pastovumas, izoliacija ir patvarumas.

Darbe vartojamos sąvokos:

Šardas – duomenų bazės, kuri yra horizontaliai išskaidyta, dalis.
Resursas arba resurso mikropaslauga – tai saityno programa, kurią programinės įrangos kūrėjas integruoja į siūlomą prieigos valdymo metodą.
Mazgas – loginis kompiuterinis vienetas (pvz.: virtualus arba fizinis įrenginys), į kurį yra diegiama programinė įranga.
Klasteris – mazgų visuma, kurie veikia kaip vieninga sistema.
Išorinė užklausa – tai užklausa, kuri sukuriama už programų sistemos ribų.
Vidinė užklausa – tai užklausa, kuri sukuriama programos sistemos ribose.
Apkrovos paskirstytojas (angl. *Load Balancer*) – tai programa, kuri tolygiai paskirsto užklausas tarp programų sistemos modulių.
Atvirkštinis tarpinis serveris (angl. *Reverse Proxy*) – tai programa, kuri kontroliuoja prieigą prie vidinių programų sistemos paslaugų persiųsdama joms užklausas.

Įvadas

Programinės įrangos kūrėjai vis dažniau projektuoja programų sistemas remdamiesi mikropaslaugų architektūra, kuri leidžia efektyviau dirbti mažose komandose, išsprendžia plečiamumo problemą ir suteikia sąlygas lanksčiam programinės įrangos kūrimo procesui. Tačiau saugumas yra vienas iš aspektų, kuriuos mikropaslaugų architektūra apsunkina. Tarp pasiskirsčiusių modulių yra sunku užtikrinti duomenų mainų konfidencialumą ir transakcijos vientisumą. Prieigos valdymo problema programuotojų bendruomenėje yra vis dar nagrinėjama ir ieškoma efektyvių, bet saugių sprendimų.

Projekto tikslas – palengvinti prieigos valdymą mikropaslaugų architektūroje sukuriant metodą, kuris būtų saugus, efektyvus ir lengvai pritaikomas.

Tikslui pasiekti keliami šie uždaviniai:

1. Išanalizuoti esamas prieigos valdymo problemas, su kuriomis susiduriama mikropaslaugų architektūroje;
2. Suprojektuoti prieigos valdymo metodą mikropaslaugų architektūroje, kuris būtų saugus, efektyvus ir lengvai pritaikomas;
3. Eksperimento metu realizuoti suprojektuotą prieigos valdymo metodo prototipą mikropaslaugų architektūroje;
4. Atlikti realizuoto prototipo tyrimą išanalizuojant jo efektyvumą skirtingomis apkrovomis ir darbo režimais.

Pirmame skyriuje pateikiama dalykinės srities objektų apžvalga ir detali prieigos valdymo strategijų analizė. Antrame skyriuje pateikiamas teorinis prieigos valdymo metodo projektas, kuris taikomas mikropaslaugų architektūroje. Trečiame skyriuje detalizuojamas sukurtas prototipas mikropaslaugų architektūroje, kuris įgyvendina esminius siūlomo prieigos valdymo metodo projekto reikalavimus. Ketvirtame skyriuje atliekamas išsamus greitaveikos tyrimas ir palyginamas sukurto prototipo efektyvumas įvairių apkrovų ir darbo režimų atžvilgiu.

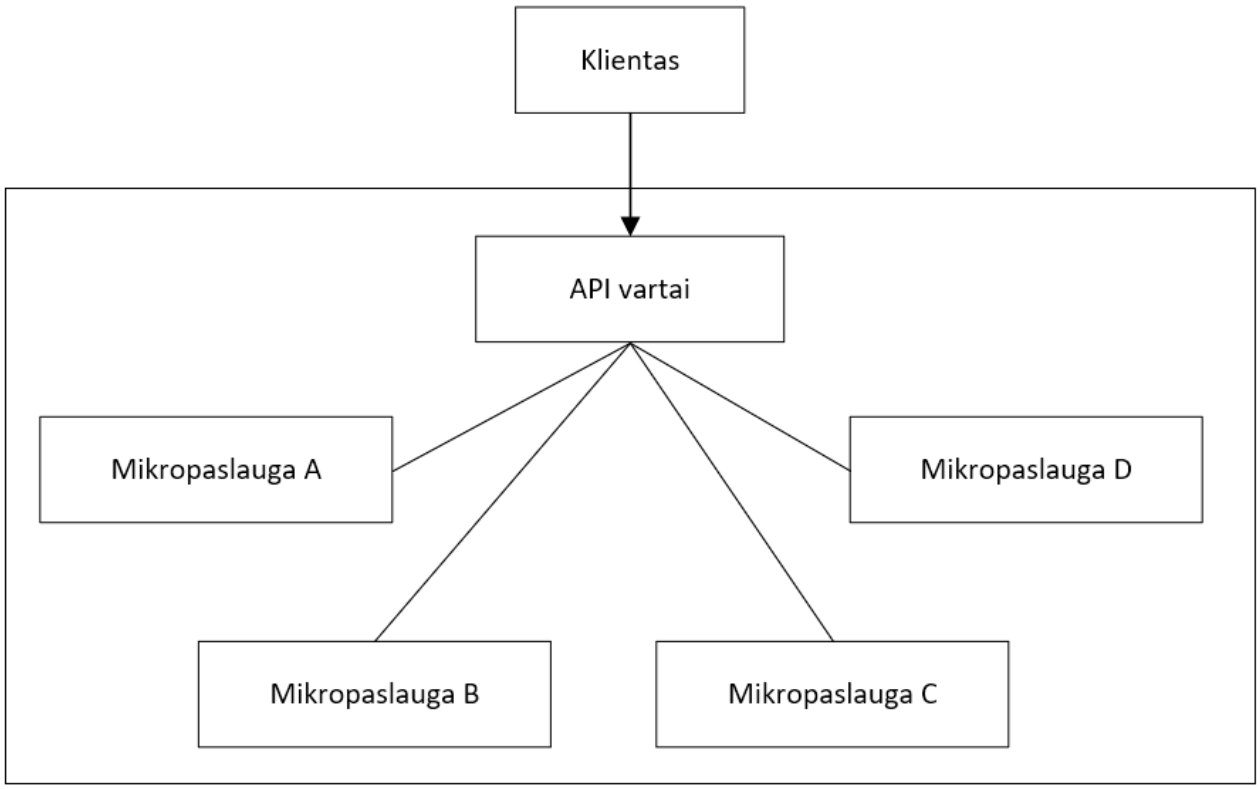
1. Prieigos valdymo metodų mikropaslaugų architektūroje analizė

1.1. Mikropaslaugų architektūros ypatumų analizė

1.1.1. Mikropaslaugų architektūros apibrėžimas

Mikropaslaugų architektūra – tai nepriklausomų aplikacijų visuma, kurios sąveikaudamos tarpusavyje sudaro bendrą programų sistemą. Ideologine prasme, kiekviena mikropaslauga turėtų atspindėti loginį verslo procesą. Ta pati mikropaslauga gali būti naudojama skirtinguose procesuose ar kontekstuose. Mikropaslaugų architektūra remiasi paskirstytosios kompiuterijos principais ir turi daug įvairių skirtingų išdėstymo variacijų. Vienos iš jų konceptualus modelis pavaizduotas **pav. 1.1**. Šis išdėstymas turi *API* vartus, kurie yra įėjimo taškas į programų sistemą – gavę užklausą iš kliento, *API* vartai kviečia reikiamas mikropaslaugas ir suformuoja pilną atsakymą klientui.

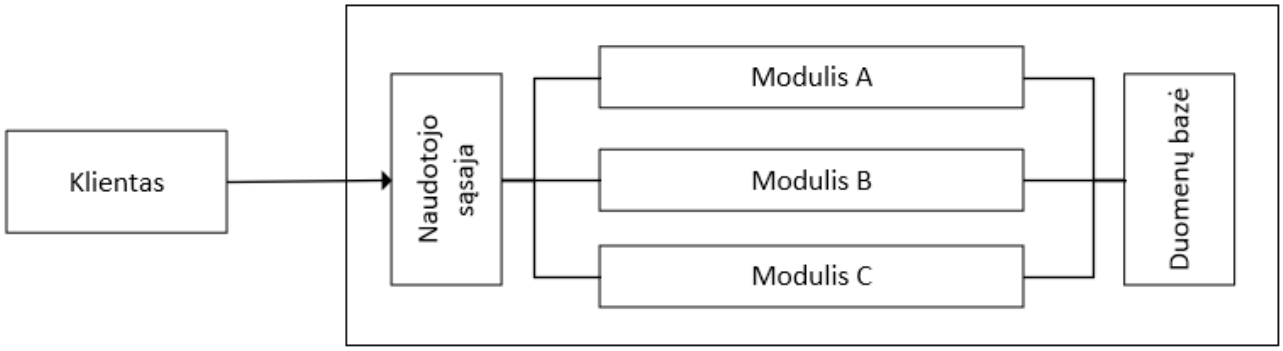
Nors išdėstymo variacijos yra įvairios, mikropaslaugų architektūrą apibūdina ir kiti ypatumai, tokie kaip lankstumas, plečiamumas, patikimumas ir kompleksiškumas [1]. Prie kiekvienos mikropaslaugos kūrimo gali dirbti mažesnės apimties autonomiškos komandos, todėl vystymo procesas yra pilnai suderinamas su lanksčia (angl. *Agile*) metodologija. Mikropaslaugos iš esmės yra savarankiškai veikiančios programos dalys, todėl jas lengva plėsti horizontaliai. Patikimumo kriterijų mikropaslaugų architektūroje garantuoja du pagrindiniai aspektai – pilnai automatizuotas diegimas bei mikropaslaugų izoliacija – vienos mikropaslaugos sutrikimas neturėtų daryti įtakos likusiai programų sistemai. Tačiau už visų šių privalumų slypi esminis trūkumas – kompleksiškumas, kuris yra natūralus paskirstytoms sistemoms.



pav. 1.1 Konceptualus mikropaslaugų architektūros modelis

1.1.2. **Mikropaslaugų ir monolitinės architektūros palyginimas**

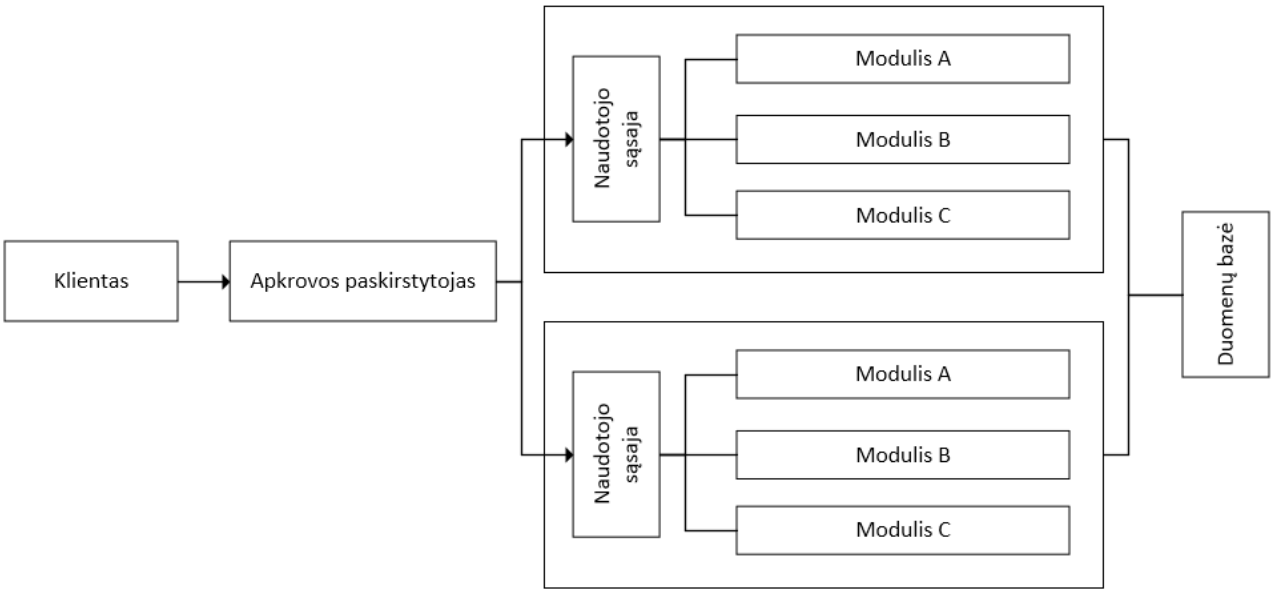
Iki šiol buvo įprasta kuriant programų sistemas remtis monolitine architektūra. Monolitinis architektūros esminis bruožas yra viso sistemos funkcionalumo enkapsuliavimas vienoje aplikacijoje, kaip pavaizduota **pav. 1.2**. Programiniai moduliai, tokie kaip sąsajos, verslo logika ir duomenų valdymas, sudaro bendrą visumą vienoje aplikacijoje. Tai pakankamai efektyvus programinės įrangos kūrimo būdas, su sąlyga, kad pirminiai nefunkciniai reikalavimai (ypač greitaveikos) programų sistemai keisis nežymiai.



pav. 1.2 Konceptualus monolitinės sistemos modelis

Tačiau per pastarąjį dešimtmetį buvo stebimas spartus internetinių aplikacijų vystymasis, todėl programų sistemos natūraliai susiduria su greitaveikos problema didėjant duomenų transakcijų skaičiui kartu su duomenų apimtimi. Vienas iš esminių monolitinės architektūros trūkumų yra plečiamumo ribotumas. Iš esmės yra 2 strategijos programų sistemos plėtimui – horizontali ir vertikali. Vertikalus plėtimas tai tėra fizinių resursų papildymas, t.y. aparatinės įrangos tobulinimas. Įprastai tai tėra laikinas ir ilgainiui brangus sprendimas, nes esminė problema nėra sprendžiama. Horizontalus plėtimas, šiuo atveju, tai – keleto tokių pačių aplikacijų diegimas vietoje vienos. Greičiausiai pritaikomas apkrovos paskirstytojas (angl. *Load Balancer*), kuris paskirsto krūvį tarp keleto tokių pačių programinių vienetų.

Horizontalaus monolito plėtimo atveju yra neišvengiamas visos aplikacijos greitaveikos didinimas, nes monolitinė programa yra visų modulių rinkinys, kurie yra stipriai susieti. Todėl, net jeigu greitaveikos problema iškyla tik tam tikruose programų sistemos moduliuose, horizontalus plėtimas pritaikomas visiems, kaip pavaizduota **pav. 1.3**.



pav. 1.3 Horizontalaus monolitinės aplikacijos plėtimo modelio pavyzdys

Šią problemą iš esmės padeda spręsti mikropaslaugų architektūrinis modelis, kuris išskaido skirtingus aplikacijos modulius į tarp savęs komunikuojančias izoliuotas ir savarankiškas programas. Tai leidžia efektyviai pritaikyti horizontalaus plėtimo strategiją, nes kiekvieną mikropaslaugą galima plėsti nepriklausomai nuo kitų programinių vienetų, kadangi jos yra tarpusavyje nėra stipriai susijusios.

Tačiau mikropaslaugų architektūra ne visuomet gali būti pateisinama greitaveikos aspektu. Eksperimento metu [2] buvo pastebėta, jog monolitinė architektūra gali apdoroti užklausas greičiau su vidutine-smulkia apkrova, nei mikropaslaugų architektūra. Viena iš pagrindinių priežasčių – mikropaslaugų architektūroje kiekvienas servisas yra atskiras loginis įrenginys tinkle. Todėl kiekvienos užklausos apdorojimo metu prisideda papildomi laiko kaštai, kurie atsiranda dėl komunikacijos tinkle. Tačiau monolitinėje aplikacijoje tarp programinių modulių vyksta tiesioginė komunikacija kompiuterio atmintyje, todėl minėtų papildomų kaštų išvengiama. Svarbu atkreipti dėmesį, kad eksperimentas atliktas viename loginiame serveryje, tačiau mikropaslaugų architektūroje kiekvienas servisas gali lokalizuotas skirtinguose tinkluose, todėl papildomi laiko kaštai atsirandantys dėl tinklo pralaidumo spartos gali būti žymiai didesni. Todėl darytina išvada, kad programų sistemas, kurios patirs mažesnę apkrovą, patartina kurti monolitinės architektūros pagrindu dėl greitaveikos pagerinimo.

1.1.3. **Mikropaslaugų architektūros privalumai**

Mikropaslaugų architektūra yra vis dažniau adaptuojama organizacijose dėl šių esminių privalumų [1]:

- Vertikali dekompozicija

  Mikropaslaugų architektūra turėtų atspindėti kiekvieną loginį verslo proceso modulį, kurį norima skaitmenizuoti. Pavyzdžiui, jeigu organizacija, kuri užsiima el. prekyba, identifikuoja tokius pardavimo proceso žingsnius, kaip: prekių paieška, jų pasirinkimas ir užsakymo valdymas, atsiskaitymas ir prekės pristatymas, tuomet kiekvienas iš šių žingsnių galėtų turėti

po atskirą mikropaslaugą. Tokiu atveju prie kiekvienos mikropaslaugos, kurios atspindėtų esminį verslo funkcionalumą, galėtų dirbti mažesnės programuotojų komandos ir jos viena kitai vystymo metu įtakos tiesioginės nedarytų, nepaisant to, kad kuria tą pačią programų sistemą. Tai leistų pritaikyti lanksčią produkto vystymo metodologiją ir vystyti programinę įrangą trumpomis iteracijomis. Tai ypač patrauklu didelėms organizacijoms, kurios stengiasi skaidyti savo darbo procesus, kad juos būtų lengviau kontroliuoti.

– Autonomiškumas

  Tai mikropaslaugų architektūros principas, kuris nusako, jog kodo pakeitimas vienoje mikropaslaugoje neturėtų daryti tiesioginės įtakos kitoms. Tai ypač svarbu ir patikimumo aspektui tuomet, kai programinė įranga jau yra įdiegta. Jeigu mikropaslaugų architektūroje sutrinka vienos iš aibės mikropaslaugų veikimas, autonomijos dėka likusios mikropaslaugos nebus paveiktos tokio incidento atžvilgiu. Priklausomai nuo realizacijos, bendras sistemos veikimas nesutriks, tačiau naudotojui kils sunkumų pasinaudoti konkrečia funkcija, kuri realizuota toje mikropaslaugoje. Tačiau autonomiškumas suteikia galimybę įgyvendinti pilnai automatizuotą mikropaslaugų diegimą, todėl tokio incidento atveju neturėtų kilti problemų iš naujo įdiegti dabartinės arba, tarkime, viena versija senesnės mikropaslaugos.

– Plečiamumas

  Didelėms organizacijoms, kurios susiduria su dideliu naudotoju kiekiu ir didele duomenų apkrova, yra labai svarbus plečiamumo aspektas. Mikropaslaugų atveju kiekviena tokios programų sistemos dedamoji yra izoliuota ir nepriklausoma, todėl jas lengva horizontaliai plėsti diegiant replikas. Svarbu atkreipti dėmesį, kad nėra būtina plėsti visos sistemos, kaip kad monolito atžvilgio – pakanka tik individualių mikropaslaugų. Kryptingas plėtimasis būdingas ir organizacinėje struktūroje, kai kyla poreikis plėsti programuotojų komandų skaičių. Kadangi mikropaslaugų architektūra užtikrina programinės įrangos moduliškumą ir atskirtį, organizacijai yra lengviau adaptuoti skirtingas komandas, kurių atsakomybės yra aiškiai atskiriamos.

– Automatizavimas

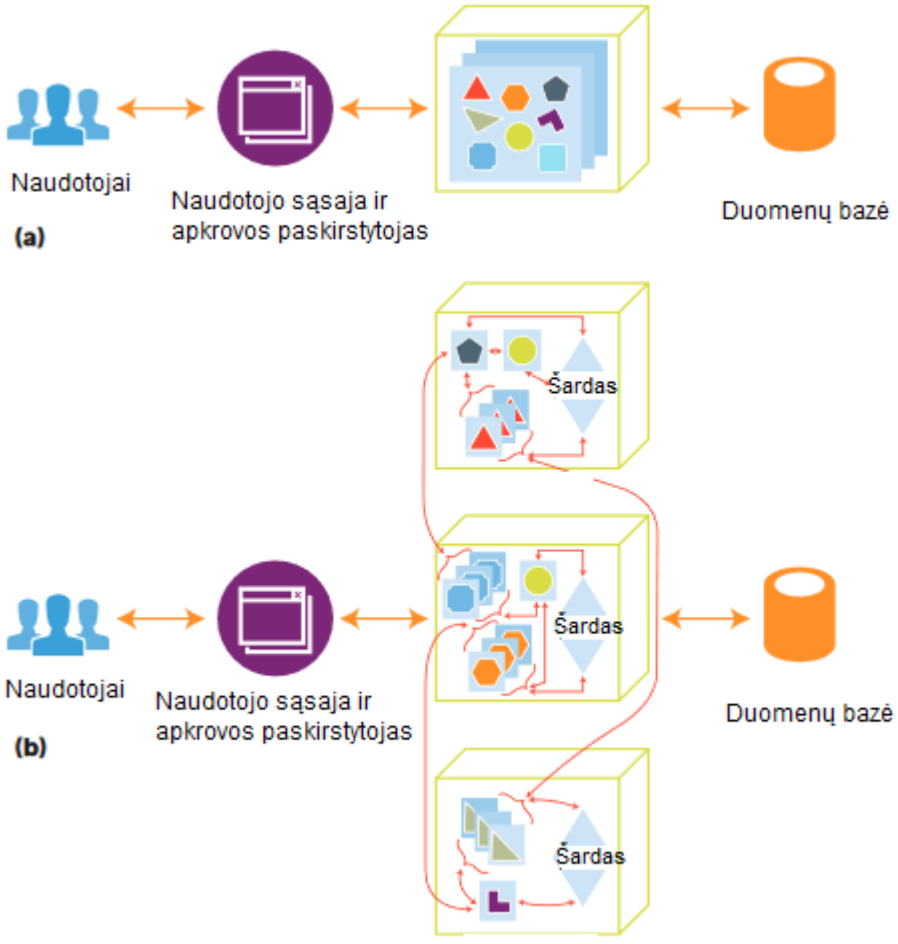
  Mikropaslaugų architektūros principu realizuota programų sistema turėtų turėti pilnai automatizuotą integracijos ir diegimo procesus. Tai sąlyginai lengva užtikrinti, dėl ankščiau minėto autonomiškumo. Kadangi kiekviena mikropaslauga yra mažai priklausoma viena nuo kitos, vienos mikropaslaugos diegimas nedaro įtakos likusiai sistemai. Todėl mikropaslaugas lengva smulkiomis iteracijomis atnaujinti. Prižiūrėti aibę paskirstytų smulkesnių sistemos dalių gali būti lengviau nei vieną didelį monolitą, jei tą daro kelios smulkesnės komandos. Svarbu atkreipti dėmesį į tai, kad visas šis procesas su šiandieninėmis technologijomis yra nesudėtingai automatizuojamas, todėl tai organizacijoms padeda mažinti programinės įrangos vystymo kaštus.

1.2. Esamų iššūkių ir problemų mikropaslaugų architektūroje analizė

1.2.1. Mikropaslaugų sudėtingumas

Nepaisant visų mikropaslaugų architektūros pranašumų, esminis trūkumas yra kompleksiškumas, kuris atsiranda dėl aibės skirtingų aplikacijų toje pačioje programų sistemoje [3]. Monolito atveju, įprastai klientas komunikuoja su aplikacija per vieną bendrą sąsają. Tai reiškia, kad iš esmės sistemos saugumo problemos nuo išorinių grėsmių gali būti sprendžiamos būtent toje bendroje sąsajoje. Mikropaslaugų architektūroje tai nėra įmanoma, nes aibė smulkesnių mikropaslaugų neišvengiamai komunikuoja tarpusavyje, todėl kiekviena mikropaslauga turi savo komunikacijos sąsają. Ši situacija pavaizduota **pav. 1.4**.

Be abejo, mikropaslaugų architektūros saugumą galima realizuoti skirtingomis perimetro saugos strategijomis, tačiau vis tiek reikia atkreipti dėmesį į kitus aspektus, tokius kaip veiksmų registravimas, veikimo stebėjimas, duomenų mainų konfidencialumas, mikropaslaugų prieinamumas. Visi šie aspektai yra sudėtingiau valdomi mikropaslaugų architektūroje. Tad prieigos valdymo metodas mikropaslaugų architektūroje privalo spręsti ne tik į identiteto valdymo problemą, tačiau atsižvelgti į visos mikropaslaugų architektūros saugų valdymo modelį.



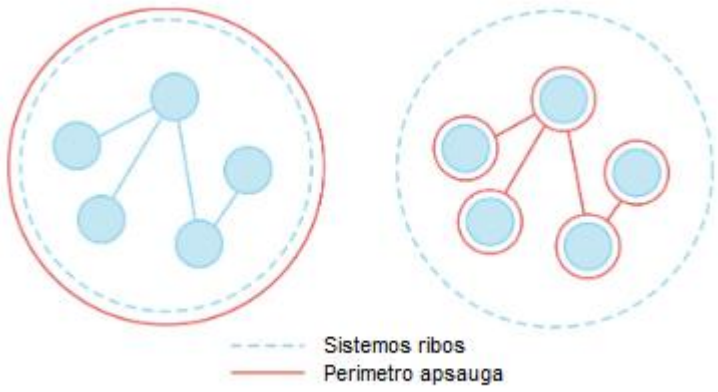
pav. 1.4 Monolitinės (a) ir mikropaslaugų (b) architektūrų kompleksiškumas [3].

1.2.2. **Saugi komunikacija**

Norint realizuoti saugią komunikaciją mikropaslaugų architektūroje, reikia aiškiai sistemos ribas. Mikropaslaugų architektūroje sistemos ribas galima nubrėžti dvejais pagrindiniais būdais, kaip pavaizduota **pav. 1.5**. Pirmoji strategija yra daryti prielaidą, kad vidinė aplinka, kurioje veikia aplikacija yra saugi, o antra – nepasitikėti aplinka ir realizuoti saugumo mechanizmus mikropaslaugų komunikavimo lygyje [4].

Pasirinkus pirmąją strategiją, reikia atsakyti į klausimą- kokioje aplinkoje programų sistema funkcionuos ir ar ja galima pasitikėti? Programų sistemos, grindžiamos mikropaslaugų architektūra, neretu atveju yra naudojamos debesų kompiuterijos paslaugomis, kai organizacijos sutaupo kaštus ties kompiuterių infrastruktūros rūpinimųsi, tačiau tai potencialiai sumažina informacijos konfidencialumą, jeigu aklai pasitikima paslaugos tiekėjo sąžiningumu. Svarbu atkreipti dėmesį, kad kitas tokios strategijos ypatumas yra tas, jog mikropaslauga aklai vykdo bet kokią gautą užklausą. Tai reiškia, jeigu perimetro viduje atakuotojui pavyks sukompromituoti vieną mikropaslaugą, jis potencialiai galės eskaluoti neteisėtą veiklą ir su kitomis mikropaslaugomis.

Antrosios perimetro saugos strategijos atveju aplinka yra nepasitikima, todėl yra privalu užtikrinti saugią komunikacijos liniją tarp mikropaslaugų. Toks metodas reikalauja daugiau kaštų projekto vystymo metu. Tačiau šiuo atveju tampa mažiau svarbu, kur yra įdiegta kiekviena mikropaslauga. Tokiu atveju į saugios komunikacijos sąvoka įeina ne tik užklausų konfidencialumas ar tik autentifikuoto naudotojo užklausų vykdymas bet ir užtikrinimas, kad užklausos siuntėjas iš tiesų yra autentifikuotas atlikti šiai užklausai. Kitaip tariant, užtikrinti, kad mikropaslauga turi teisę siųsti tam tikrą užklausą kitai mikropaslaugai.



pav. 1.5 Perimetro saugos strategijos mikropaslaugų architektūroje [4].

1.2.3. **Veiksmų stebėjimas**

Programų sistemos veikimo stebėjimas yra svarbus aspektas programinės įrangos gyvavimo cikle. Mikropaslaugų architektūros atveju yra ypač svarbu stebėti kiekvienos mikropaslaugos greitaveiką, nes padidėjęs atsakymo laikas vienos mikropaslaugos potencialiai sulėtins visą programų sistemą [5]. Tačiau mikropaslaugų greitaveikos stebėjimas yra kompleksinė problema. Priklausomai nuo

išdėstymo modelio, viena mikropaslauga gali ne tik savarankiškai apdoroti jai skirtą užklausą, bet siųsti papildomas užklausas kitoms mikropaslaugoms.

Šiai problemai spręsti galima realizuoti centralizuotą veiksmų registravimą mikropaslaugų architektūroje [5]. Be abejo, centralizavimas mikropaslaugų architektūros atveju gali padidinti užklausos apdorojimo laiko kaštus, tačiau tai gali stipriai palengvinti incidentų stebėjimo, aptikimo ir tyrimo kaštus. Decentralizuotas veiksmų registravimas pagreitins programų sistemos atsakymo laiką, tačiau veiksmų žurnalus teks peržiūrėti kiekvienai mikropaslaugai atskirai.

Tačiau apart greitaveikos stebėsenos yra taip pat svarbu užtikrinti kiekvieno programų sistemoje veiksmo integralumą. Yra įprasta vesti naudotojo aplikacijos naudotojų veiksmų žurnalus, kuriuo galima rasti, kuriuos konkrečius veiksmus atliko naudotojai sistemoje. Tačiau mikropaslaugų atveju yra svarbi galimybė atsekti veiksmus programų sistemos struktūros lygyje. Kitaip sakant atsekti, kuris subjektas kreipėsi į konkrečią mikropaslaugą ir su kokiu tikslu. Nelogiška veiksmų seka gali signalizuoti IT incidentą, pavyzdžiui aptikus, jog administratoriaus paskyra kreipėsi į pristatymo mikropaslaugą su tikslu pakeisti vienos iš kliento paskyros slaptažodį. Tokiu atveju svarbu atkreipti dėmesį ne tik į tai, jog administratorius bandė atlikti nelegalų veiksmą, tačiau ir į pačias mikropaslaugas, kuriomis bandyta tą veiksmą įgyvendinti.

1.2.4. **Pasiekiamumas ir prieinamumas**

Mikropaslaugų architektūros principas dėl automatizuoto diegimo leidžia lengvai keisti programų sistemos aparatinę infrastruktūrą. Todėl diegimo kaštai iš esmės vienodi, nepriklausomai nuo to, ar mikropaslauga yra diegiama į lokalų kompiuterį, nutolusį virtualų privatų serverį ar debesų kompiuteriją. Tačiau todėl kyla kita pasiekiamumo iššūkis – dėl dinamiško mikropaslaugų architektūros diegimo gali būti sunku atsekti, kur kokia mikropaslauga yra įdiegta. Kitaip tariant, kiekvienai mikropaslaugai reikia žinoti, kokiais adresais yra pasiekiamos kitos susijusios mikropaslaugos. Situaciją apsunkina ir tai, kad kiekviena mikropaslauga turėtų būti diegiama nedarydama įtakos kitoms mikropaslaugoms, todėl mikropaslaugų lokacijos adresai gali kisti realiu laiku.

Viena iš siūlomų pasiekiamumo valdymo strategijų yra aktyvus mikropaslaugų aktyvumo tikrinimas [6]. Metodo principas paprastas – kas tam tikrą intervalą siųsti užklausą į mikropaslaugą, kurios paskirtis yra nustatyti, kad mikropaslauga funkcionuoja sėkmingai. Jeigu gaunamas neigiamas atsakymas, programų sistema reaguoja atitinkamai, pavyzdžiui ištrina neveikiančios mikropaslaugos adresą iš aktyvių komponentų sąrašo, kad kitos mikropaslaugos nebandytų kviesti mikropaslaugos, kurios veikimas yra sutrikęs. Šiam metodui siūloma taikyti du šablonus – grandinės jungiklio arba paslaugų registro.

Grandinės jungiklis seka, kurios ar iš mikropaslaugos yra gaunamas sėkmingas atsakymas. Jeigu jis negaunamas (arba po tam tikro skaičiaus nesėkmingų atsakymų), tuomet vietoj mikropaslaugos kvietimo yra grąžinamas klaidos pranešimas. Tokiu būdu yra užtikrinama, kad mikropaslauga, kurios veikimas yra sutrikęs, nebegaus jokių kitų užklausų. Taip bus sumažinta tikimybė, kad pažeista mikropaslauga toliau bus bandoma trikdyti, eskaluoti jos privilegijas. Kitaip tariant, sutrikusi mikropaslauga tampa nebepasiekiama bendroje programų sistemoje.

Paslaugų registro pagrindinė funkcija yra atskirti mikropaslaugą nuo jos fizinio adreso. Kitos mikropaslaugos turi kreiptis į paslaugų registrą, norėdamos sužinoti tam tikros mikropaslaugos adresą. Todėl šiuo atveju galima nesunkiai pakeisti mikropaslaugų adresą ar jų aibę realiu laiku.

1.2.5. **Skirtumas tarp naudotojo ir kliento užklausų**

Priklausomai nuo verslo panaudojimo atvejo, subjektai, besinaudojantys programų sistema, gali būti skirstomi į naudotojus ir klientus. Naudotojas – tai programų sistemoje registruotas subjektas, kuris asmeniniais tikslais naudojasi aplikacijos suteikiamomis funkcijomis. Tuo tarpu klientas – tai dažniausiai atstovauja kitą organizaciją ir naudojasi programų sistemos paslaugomis siekdamas naudos savo organizacijai. Kitaip tariant, naudotojas tai paprastas asmuo, o klientas – kitos organizacijos atstovas.

Autentifikacijos ir autorizacijos tikslas kliento ir naudotojo atžvilgiu šiek tiek skiriasi. Naudotojo atveju programų sistemos tikslas yra patvirtinti jo tapatybę ir leisti naudotis atitinkamais resursais ir funkcijomis, kurios jam yra galimos. Tačiau kliento atveju, įprastai autorizacijos sąvoka šiek tiek platesnė dėl užklausų kvotų. Išorinių klientų iš kitų organizacijų užklausos dažnai yra monetizuojamos ir turi tam tikras ribas per tam tikrą laiko tarpą. Mikropaslaugų architektūroje užklausų kiekio apribojimus yra sudėtingiau realizuoti dėl to, kad vieną kliento užklausą gali keletas naujų užklausų į kitas mikropaslaugas, todėl svarbu teisingai apibrėžti užklausų kvotos politiką.

Šiuolaikinė praktika klientų užklausoms sekti ir riboti yra specialių prieigos žetonų arba sesijos identifikatorių naudojimas [7]. Svarbu atkreipti dėmesį, kad klientas gali būti ne asmuo, o kita kompiuterinė sistema, kuri naudojasi kitos organizacijos programų sistemos funkcijomis. Tokiu atveju HTTP sesijos mechanizmas netinka, nes yra sunkiai suderinamas su kitomis nei naršyklės sąsajomis.

Tačiau labiausiai paplitusi praktika yra JWT žetonų naudojimas [7]. JWT yra paprastas mechanizmas užklausos validumui įrodyti. JWT žetonas susideda iš 3 pagrindinių dalių – antraštės, žinutės ir kriptografinio parašo. Tiesa, kriptografinis parašas yra neprivalomas, tačiau stipriai rekomenduotinas saugumo sumetimais. JWT žetono validavimo metu svarbu įsitikinti, kad kriptografinio parašo dalis egzistuoja, nes JWT žetonas be jokio kriptografinio parašo taip pat laikomas tinkamu. Tokiu atveju, atakuotojas gali įrašyti bet kokią informaciją į žinutę ir taip eskaluoti neteisėtą prieigą prie sistemos.

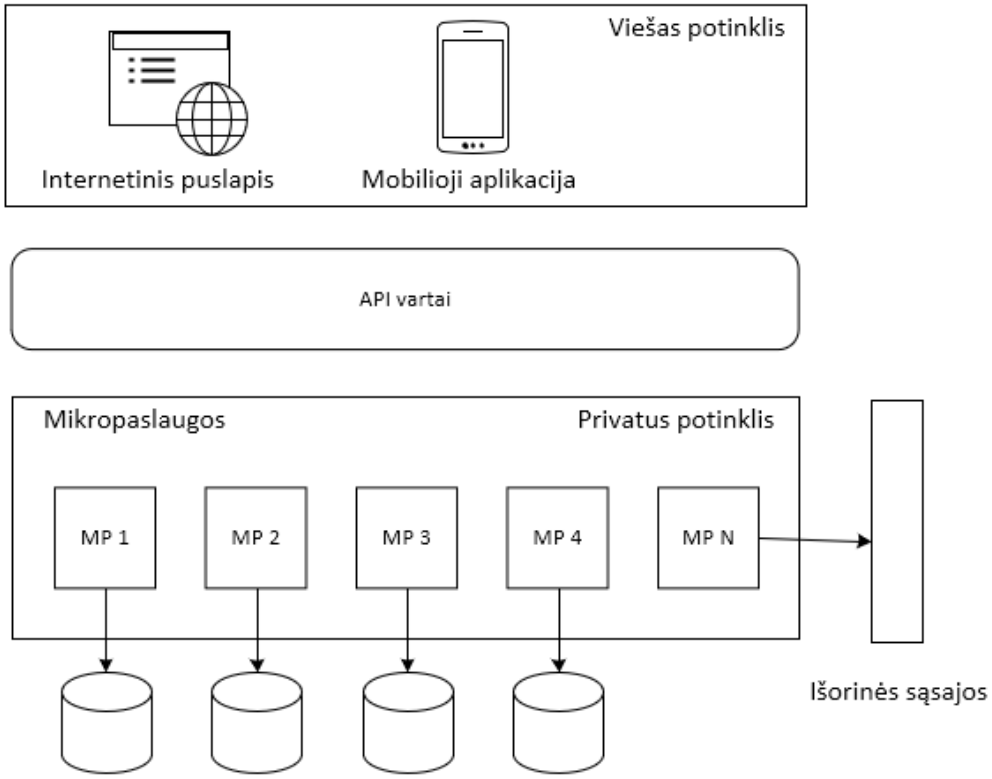
1.3. Prieigos valdymo strategijų mikropaslaugų architektūroje analizė

1.3.1. **Prieigos valdymo uždaviniai**

Kaip aptarta 1.2.2 skyrelyje, norint užtikrinti saugią komunikaciją mikropaslaugų architektūroje, reikia realizuoti identiteto valdymo mechanizmą. Šis mechanizmas turi užtikrinti saugią komunikaciją tarp aibės mikropaslaugų, kurios priklauso tai pačiai sistemai. Šis tikslas gali būti išreiškiamas dvejais pagrindiniais uždaviniais- užtikrinti, gauta užklausa yra patikima ir kad subjektas turi teisę kviesti tam tikrą mikropaslaugą.

1.3.2. **Sluoksninis prieigos valdymo metodas**

Lyginant su monolitinėmis aplikacijomis, mikropaslaugų architektūra turi didesnį atakos paviršių [8]. Kaip pavaizduota **pav. 1.6**, standartinis mikropaslaugų architektūros išdėstymas turi API vartus, į kuriuos kreipiasi išorinis klientas. Jie nukreipia užklausą į atitinkamas mikropaslaugas. API vartai tarnauja kaip centralizuota sąsaja, skirta programų sistemos klientams. Tačiau neturint jokio centralizuoto prieigos valdymo metodo, kiekviena mikropaslauga turėtų realizuoti naudotojo autentifikaciją ir autorizaciją atskirai. Nesant prieigos valdymo mechanizmui, kuris ne tik valdytų naudotojų, bet ir pačių mikropaslaugų prieigą gali kilti papildomų bendro sistemos saugumo problemų, kurios aptartos skyrelyje 1.2.2.

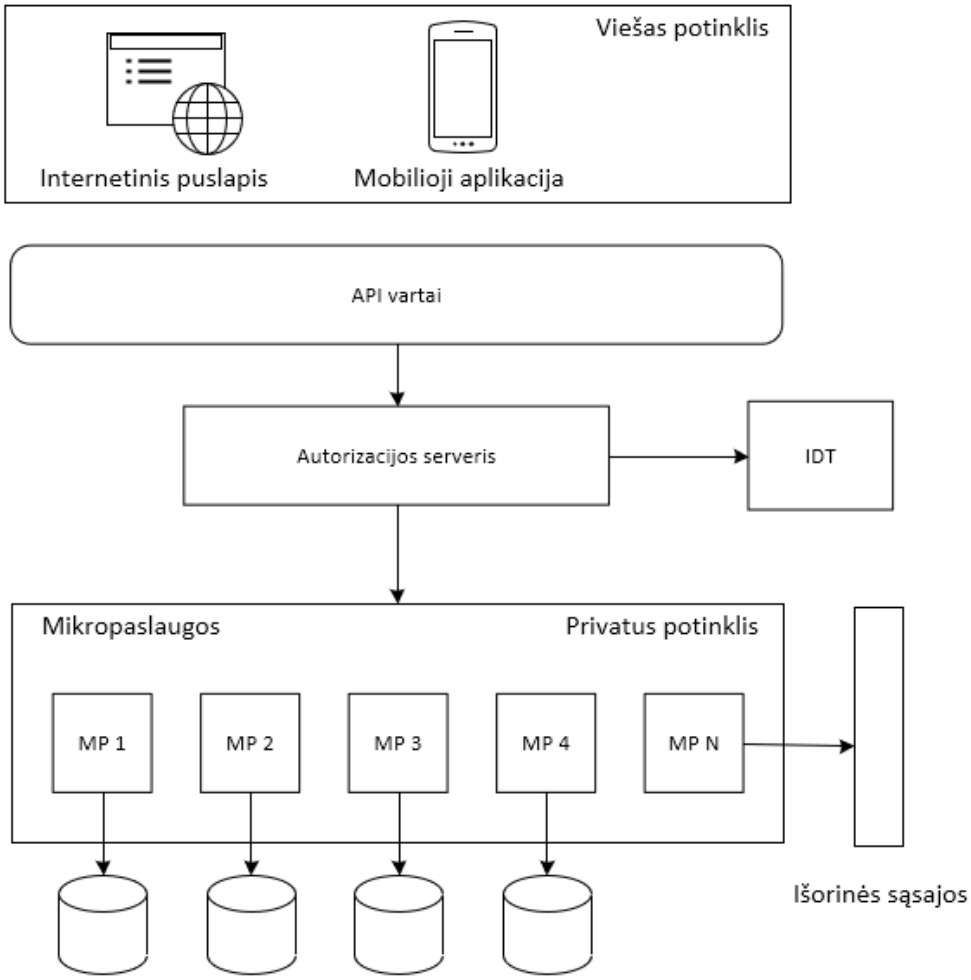


pav. 1.6 Mikropaslaugų architektūros išdėstymas [8].

Kadangi mikropaslaugų aibė priklauso vienai programų sistemai, vertėtų rinktis centralizuotą prieigos valdymo metodą. Taip kiekviena mikropaslauga galėtų fokusuotis ne į bendrą infrastruktūrinį, bet į konkretų verslo logikos funkcionalumą. Kaip pavaizduota **pav. 1.7**, API vartai galėtų kreiptis tiesiai į autorizacijos serverį, kuris patvirtintų naudotojo tapatybę ir teisęs atlikti veiksmus, nurodytus užklausoje. Autorizacijos serveris taip pat komunikuoja su IDT (Identiteto Teikėju), kuris gali būti tiek vidinis, tiek išorinis aktorius. Teoriškai mikropaslauga gavusi užklausą iš autorizacijos serverio galėtų pasitikėti užklausos validumu, nes ją jau validavo autorizacijos serveris.

Tačiau tokia pasitikėjimo strategija atveria potencialias saugumo spragas, aptartas skyrelyje 1.2.2. Galiausiai sėkmės atveju autorizacijos serveris suteikia galimybę realizuoti užklausą. Šio metodo scenarijaus žingsniai [8]:

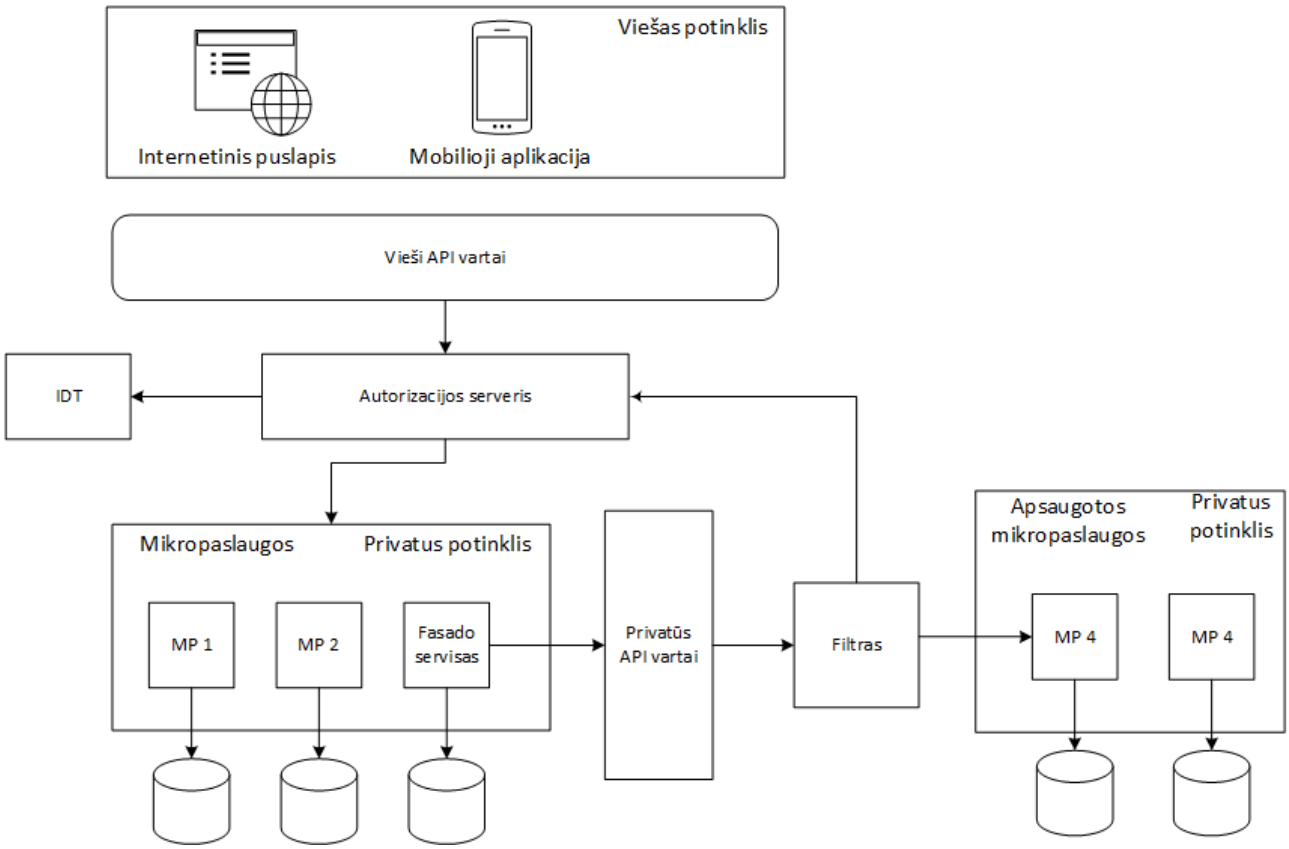
1. Naudotojas prašo prieigos prie mikropaslaugos;
2. Kreipiamasi į autorizacijos serverį su prašymu gauti prieigą prie mikropaslaugos;
3. Autorizacijos serveris autorizuoja ir/ar autentifikuoja naudotoją kreipdamasis į identiteto teikėją;
4. Naudotojas gauna prieigos ir atnaujinimo žetonus ir juos saugo visos sesijos metu;
5. Naudotojas siunčia užklausą į mikropaslaugą, tačiau ji prieš tai žetono pagalba validuojama autorizacijos serverio. Jeigu žetono galiojimo laikas pasibaigęs, naudojamas atnaujinimo žetonas.



pav. 1.7 Mikropaslaugų prieigos apsauga naudojant autorizacijos serverį ir API vartus [8].

Tačiau tokios strategijos atveju potencialiai susiduriama su kita prieigos valdymo problema. Įprastai mikropaslaugų architektūroje esmines funkcijas galima suskirstyti į keletą skirtingų saugumo lygių. Tokia prieigos valdymo strategija, kuri pavaizduota **pav. 1.7**, turi nefunkcinį apribojimą, kad visos mikropaslaugos esančios tame pačiame potinklyje turi vienodą saugumo lygį. Tačiau realybėje mikropaslaugų saugumo lygmenys skiriasi ir prieš atliekant tam tikras kritines funkcijas gali būti reikalinga papildoma užklausos validacija ar autorizacija. Šiai problemai spręsti yra siūlomas privačių API vartų ir filtro strategija [8], pavaizduota **pav. 1.8**. Šiame prieigos valdymo modelyje privatūs API vartai yra pasiekiami tik vidiniai mikropaslaugai – fasadui, kuri rūpinasi tikslingu kritinių

mikropaslaugų iškvietimu. Šis fasadas kreipiasi į vidinius API vartus, kurie toliau siunčia užklausą į filtro servisą. Filtro servisas yra papildomas instrumentas užklausos validumui ir autorizacijai užtikrinti. Filtro objektas yra atskiras nuo autorizacijos serverio, nes filtre turėtų būti tik realizuota papildoma verslo logika, kurią reikalinga realizuoti prieš pasiekiant kritines mikropaslaugas. Vienas iš pavyzdžių – papildomos užklausų kvotos politikos tikrinimas arba užklausos viduje esančios informacijos validumas, kuris priklauso tik nuo verslo logikos. Filtras gali atlikti papildomą autentifikaciją kreipdamasis į autorizacijos serverį. Sėkmės atveju filtras siunčia užklausą sekančio saugumo lygio mikropaslaugoms. Šis modelis vadovaujasi pasitikėjimo saugos perimetro principu, todėl reikalauja papildomų priemonių potinklio saugumo užtikrinimui.



pav. 1.8 Mikropaslaugų apsaugos modelis naudojant privačius apsaugos vartus [8].

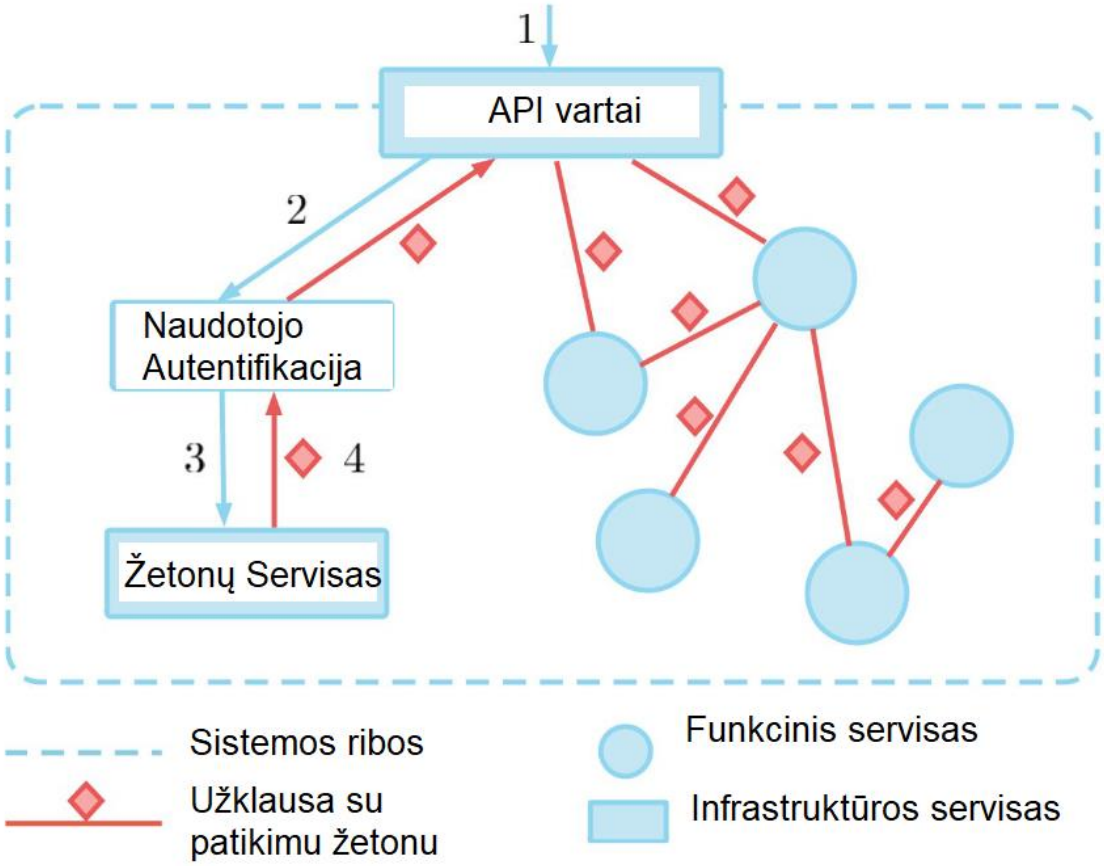
1.3.3. **Prieigos valdymo metodas su vidiniais žetonais**

Mikropaslaugų architektūroje prieigos valdymą naudotojo ir mikropaslaugų atžvilgių galima realizuoti su vidinio žetono modeliu, kaip pavaizduota **pav. 1.9**. Šis modelis užtikrina, kad kiekviena užklausa bus autorizuota vidinio serviso, kuris išduoda žetonus. Priešingai nei įprasta, šie pasirašyti žetonai nėra grąžinami naudotojui. Kitaip tariant, žetono gyvavimo ciklas yra lygus užklausos gyvavimo ciklui. Tokia apsaugos priemonė užtikrina, kad iš išorės atėjusi neautorizuota užklausa bus neįvykdyta. Ši strategija apima šiuos esminius žingsnius [4]:

1. Naudotojo užklausa siunčiama į API vartus.
2. API vartai deleguoja naudotojo autentifikavimui vidiniam prieigos valdymo servisui;
3. Naudotojo autentifikavimo servisas naudotoją autentifikuoja ir kreipiasi į žetonų išdavimo servisą.
4. Žetonų servisas nustato, kokias teises turi naudotojas ir išduoda žetoną.

Tolesniuose žingsniuose, šis žetonas yra pridedamas prie kiekvienos mikropaslaugos užklausos. Tokiu būdu kiekviena mikropaslauga sėkmingai validavusi žetoną su naudotojo teisėmis gali pasitikėti gauta užklausa. Svarbu atkreipti dėmesį į 2 papildomus dalykus – žetono galiojimo laiką ir išorinė naudotojo autentifikaciją. Saugumo sumetimais rekomenduotina, kad žetono galiojimo laikas būtų kiek įmanoma trumpesnis. Tikslus trukmės apribojimas priklauso nuo pačios programų sistemos, tačiau reikia turėti omenyje, kad vienas žetonas galioja vienos užklausos metu. Kadangi šis žetonas naudojamas tik tarp mikropaslaugų ir nėra suteikiamas naudotojui, reikia atkreipti dėmesį ir į išorinio naudotojo identiteto nustatymą. Tačiau šiai problemai spręsti galima naudotojui išduoti atskirą specialų žetoną, kuris galėtų būti validuojamas būtent naudotojo autentifikacijos servise (3 žingsnis).

Tačiau vidinio žetono strategijos realizacija gali sukelti keletą problemų. Pirmiausia, dėl greitaveikos – jeigu naudotojas per trumpą laiko tarpą siunčia didelį kiekį užklausų į API vartus, bendras sistemos atsakymo laikas turėtų smarkiai sumažėti. Tokiu atveju kyla klausimas, ar tikrai kiekvieną kartą būtinai reikia išduoti naują žetoną? Nes potencialus sprendimas šiai problemai galėtų būti žetonų kešavimas API vartų arba naudotojo autentifikacijos servise. Kešavimo atveju reikėtų atsižvelgti į žetonų galiojimo laiką. Tie žetonai, kurių galiojimo laikas jau pasibaigęs arba greitai pasibaigs, galima pašalinti iš kešavimo. Antra, jeigu žetono validavimui yra naudojama bendra paslaptis (simetrinės kriptografijos atveju) arba viešasis raktas (asimetrinės kriptografijos atveju), tai reikia paskirstyti tarp sistemų. Kurti atskirą sertifikatų išdavėją, kuriuo galėtų pasitikėti visos mikropaslaugos yra neefektyvu. Todėl būtų galima pritaikyti priskyrimą automatizuoto diegimo metu. Tačiau pakeitus raktą žetonų servise, būtų privaloma iš naujo įdiegti visas mikropaslaugas, kurios validuoja šį žetoną dėl pasikeitusios rakto reikšmės. Paskirstytose sistemose tai nėra lengvas uždavinys, nes tai ko gero reikštų, kad sistema nebus pilnai prieinama naudotojai bendro raktu pakeitimo metu, kol mikropaslaugos bus diegiamos.

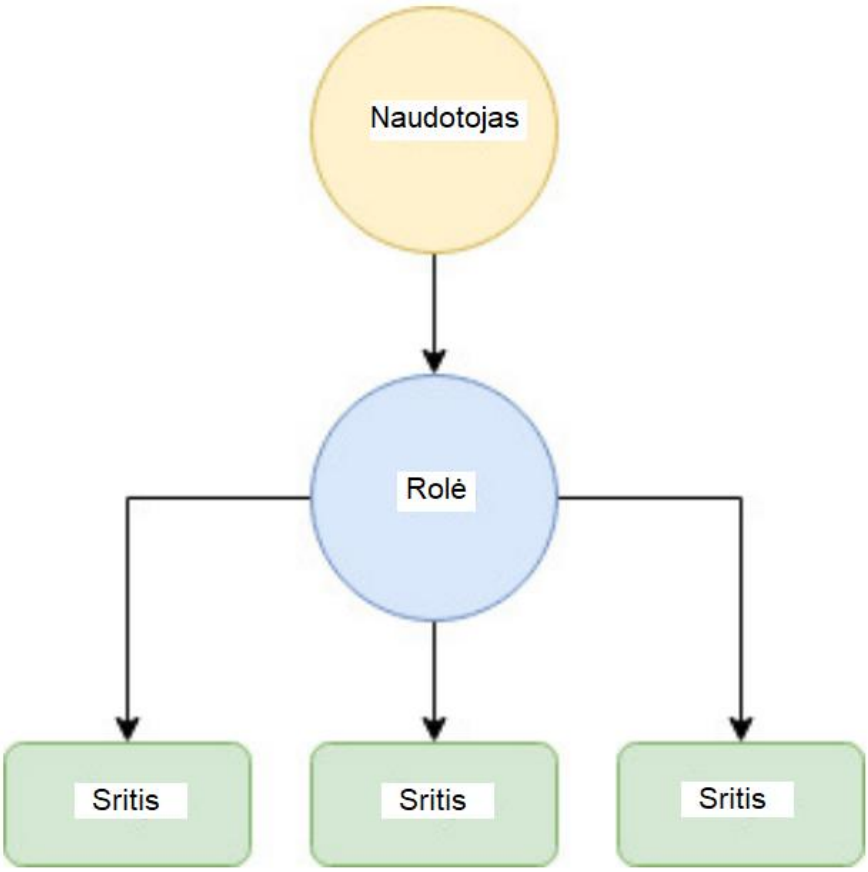


pav. 1.9 Vidiniais žetonais grįsta mikropaslaugų prieigos valdymo strategija [4].

1.3.4. **Identiteto valdymo modelis naudojantis OAuth2**

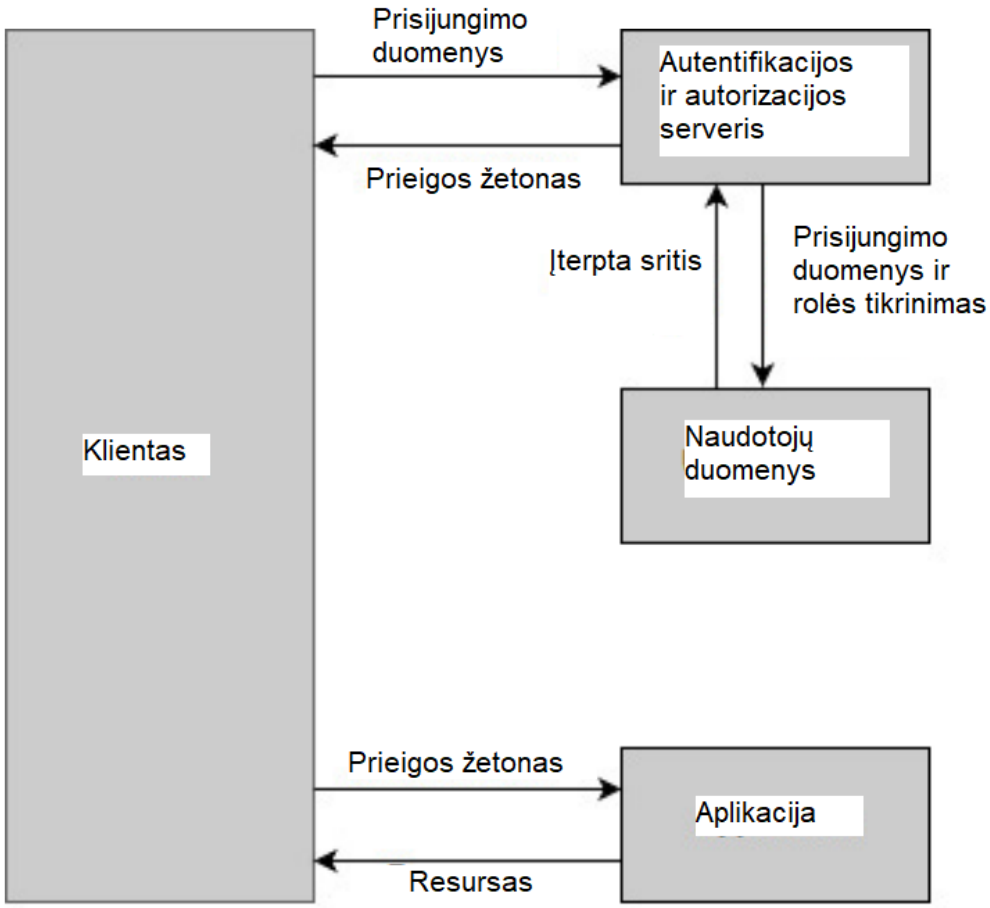
Egzistuoja programos sistemos, kurios grindžia savo paslaugas ne tik savo, bet ir kitų programų sistemų funkcijomis. Pavyzdžiui, šiomis dienomis tapo įprasta naudotojui turėti prisijungimą prisijungti ar užsiregistruoti prie sistemos per kitą, trečiąją aplikaciją. Suteikti programų sistemai prieigą prie kitos sistemos resursų, galima naudoti OAuth2 autorizacijos protokolą. Pagal atliktą tyrimą [9] galima daryti prielaidą, kad OAuth2 yra plačiai naudojamas protokolas, ir išvadą, kad jis padeda aplikacijoms standartizuotai valdyti naudotojo identitetą. OAuth2 protokolas yra pakankamai platus ir lankstus, todėl autorizavimo scenarijų galima realizuoti keliais skirtingais būdais. Tačiau svarbu atkreipti dėmesį, kad sistemos kompleksiškumas dažniausiai padidina klaidų galimybę, o tuo tarpų ir pažeidžiamumų atsiradimo tikimybę, todėl visuomet rekomenduotina taikyti kuo paprastesnį modelį. Tai ne išimtis ir identiteto valdymo modelio realizavimui naudojantis OAuth2 protokolu.

Vienas iš siūlomų OAuth2 autorizacijos modelių yra rolėmis grįstas prieigos valdymas [10]. Rolės modelis vaizduojamas **pav. 1.10**. Siūlomo metodo atveju kiekvienas naudotojas gali turėti keletą rolių ir kiekviena rolė gali turėti vieną ar daugiau sričių. Kiekviena sritis apibūdina kontekstą, kuris bus prieinamas naudotojui bei funkcijas ir apribojimus, kurie bus taikomi naudotojo atžvilgiu. Tai suteikia lankstumą aplikacijos lygyje, kai ta pati rolės naudotojo atžvilgiu turi skirtingą prieigą, priklausomai nuo konteksto.



pav. 1.10 Konceptualus naudotojo rolės objekto modelis [10].

Esminis autorizavimo scenarijus [10] yra pavaizduotas **pav. 1.11**. Pirmiausia klientas arba jį atstovaujanti trečioji šalis (pvz.: kita aplikacija) pateikia naudotojo prisijungimo duomenis autentifikacijos ir autorizacijos serveriui (arba mikropaslaugai). Tuomet yra tikrinama, ar prisijungimo duomenys sutampa su sistemoje esančiais duomenimis. Jeigu duomenys sutampa, yra generuojamas žetonas, į kurį įtraukiamos ir naudotojui priklausančios rolės sritys. Sugeneruotas žetonas gražinamas klientui. Tuomet klientas gali atlikti užklausą į aplikacija su tikslu pasinaudoti konkrečiu resursu, pridėdamas prieš tai išduotą prieigos žetoną. Svarbu atkreipti dėmesį į žetono galiojimo laiką ir žetono atšaukimą. Rekomenduotina išduoti žetonus su kuo trumpesniu galiojimo laiku, norint padidinti jo saugumą.



pav. 1.11 Siūlomas autorizacijos scenarijus [10].

Taip pat reikėtų apsvarstyti galimybę atšaukti žetono galiojimą, jeigu išduotas žetonas galioja ilgai. Tokia funkcija praverčia tokiu atveju, jeigu ilgai galiojantis žetonas buvo perimtas atakuotojo ir klientas (arba trečioji aplikacija, kuri jį atstovauja) nori užtikrinti, kad senu sukompromituotu žetonu nebus įmanoma pasinaudoti. OAuth2 protokolo standartas [11] apibrėžia, jog žetonų atšaukimo funkcija turėtų būti suteikiama per papildomą sąsajos prieigos tašką, kuris priimtų atšauktą žetoną. Kartu su žetonu galima (bet neprivaloma) nurodyti žetono tipą (prieigos, atnaujinimo). Svarbu turėti omenyje, kad nėra prasmės atšaukti žetonų, kurių galiojimo laikas jau pasibaigęs.

1.3.5. **Prieigos valdymo strategijų palyginimas**

Išnagrinėtos pagrindinės prieigos valdymo strategijos mikropaslaugų architektūroje yra savotiškai skirtingos. Atsakyti į klausimą „kuri prieigos valdymo strategija yra geriausia?“ yra neįmanoma, nes kiekviena jų turi savus pliusus ir minusus bei pritaikymo atvejus. Todėl reikėtų užduoti klausimą „kuri prieigos valdymo strategija yra tinkamiausia konkrečiu atveju?“. Šio klausimo atsakymui yra privalu šias strategijas tarpusavyje palyginti pagal esminius kriterijus – kompleksiškumą, suderinamumą, perimetro saugą, veiksmų stebėjimą ir greitaveiką.

Mažiausiu sistemos išdėstymo kompleksiškumu pasižymi sluoksninis prieigos valdymo metodas vien dėl to, nes iš esmės jis turi tik vieną įvesties tašką visoje sistemoje. Tai reiškia, kad saugumo mechanizmas egzistuoja tik vienoje vietoje ir jis nepriklauso nuo likusios sistemos infrastruktūros. Todėl jį įdiegti ir palaikyti yra sąlyginai paprasta. Priešinga situacija yra su OAuth2 identiteto valdymo modeliu, nes jo realizacija gali būti įvairi. Jo viena iš paskirčių yra suderinamumas su kitomis sistemomis ir šio kriterijaus vertinimas gali reikšti sudėtingą identiteto valdymo metodą tarp skirtingų paslaugų. Tačiau vidinių žetonų atveju, saugos mechanizmo kompleksiškumas nėra paprastas, nes topologinė tinklo schema turi papildomas identiteto valdymo paslaugas.

Akivaizdu, kad OAuth2 modelio naudojimas mikropaslaugų architektūroje be abejo išsprendžia suderinamumo su kitomis sistemomis. Tačiau vidinių žetonų atveju taip pat neturėtų būti sudėtinga išspręsti suderinamumo problemą, nes vidiniai žetonai klientui tiesiogiai nėra suteikiami, taip „slepiant“ identiteto valdymo kompleksiškumą. Tačiau sluoksninę mikropaslaugų sistemą neretai būna sudėtingiau suderinti su kitomis sistemomis, nes identiteto valdymo mechanizmas yra bendras tiek vidinėms, tiek išorinėms sistemoms. Todėl norint integruoti klientus, dažniausiai reikia papildyti ar iš pagrindų keisti jau esantį centralizuotą apsaugos mechanizmą.

Kalbant apie perimetro saugą, sluoksninis metodas įprastai jos neturi, nes remiamasi tik vienkartine autorizacija. Tačiau perimetro sauga gali būti realizuota prie autorizuotų užklausų pridedant užklausos autentifikavimo mechanizmą (pvz.: JWT žetoną). Taip pat ir OAuth2 modeliu – pirmoji užklausa į resursą visada bus autentifikuota, tačiau priklausomai nuo to, kas su užklausa bus daroma toliau, ar bus realizuota tolimesnė autorizacija, priklauso nuo pritaikymo atvejo. Įprastai rekomenduojama tai realizuoti. Vidinių žetonų atveju perimetro sauga yra neišvengiama, nes paslaugų komunikacija būtent grįsta šių žetonų pasitikėjimu.

Veiksmų stebėjimo užtikrinimas yra vienas iš svarbesnių saugios sistemos bruožų. Sluoksninėse sistemoje tai užtikrinti yra sąlyginai paprasta, nes visos užklausos yra autorizuojamos vienoje vietoje. Kita vertus, OAuth2 modelis taip pat gali būti pakankamai paprastas, jeigu turime vieną autorizavimo paslaugą. Kuo daugiau autorizavimo paslaugų turime, tuo sudėtingėja veiksmų žurnalo vedimas. Vidinių žetonų modelis apsunkina stebėjimą dėl to, nes po autorizavimo fakto nėra aišku, kur ir kaip užklausa keliaus toliau. Todėl kiekviena paslauga turi užtikrinti korektišką veiksmų auditavimą.

Sparčiausia greitaveika pasižymi sluoksninis metodas, nes jame užklausa yra patvirtinimą vieną kartą viso jos gyvavimo metu. OAuth2 atveju įprastai naudojamos autorizacijos paslauga, kuria būtina pasinaudoti prieš kreipiantis į resursus, todėl klientas turi pats rūpintis autorizacija. Vidinių žetonų atvejų greitaveika yra prasčiausia, nes yra naudojami atskiros autentifikavimo ir autorizavimo paslaugos. Papildomai, kiekviena užklausa gauna papildomą vidinį žetoną, kuriuo pasitiki visos vidinės paslaugos. Šio žetono išdavimas ir jo siuntimas kiekvienos sekančios užklausos metu reikalauja papildomų laiko kaštų bei sumažina bendrą tinklo pralaidumą, priklausomai nuo žetono ir perduodamos informacijos kiekio.

Apibendrinant **lentelė 1.1**, darytina išvada, jog vidinių žetonų prieigos valdymo strategija yra bene saugiausias metodas. Priešingai nei Sluoksninio ar OAuth2 modeliai, vidiniai žetonai „neišduoda“ autorizacijos struktūros klientams. Taip pat vidiniai žetonai užtikrina perimetro saugą, todėl sistema „neturi ribų“. Kitaip tariant, tokia programų sistema bus apsaugota nuo neautorizuotos prieigos iš vidinio potinklio į neapsaugotus resursus atakos. Tačiau vidutinių žetonų strategija nusileidžia OAuth2, kai kalbama apie integraciją su kitomis sistemomis, nes vidinių žetonų strategija neturi sutarto autorizacijos protokolo, priešingai nei OAuth2 modelis. Taip pat vidiniai žetonai turi atsargiai spręsti greitaveikos problemą, nes vidinių žetonų išdavimas ir perdavimas kartu su kiekviena užklausa reikalauja papildomų laiko ir tinklo pralaidumo kaštų.

lentelė 1.1 Prieigos valdymo strategijų palyginimas

Kriterijus	**Monolitinis**	**Sluoksninis**	**Vidiniai žetonai**	**OAuth2 modelis**
Saugumo mechanizmo kompleksiškumas	Paprastas	Paprastas	Vidutinis	Sudėtingas
Programų sistemos sudėtingumas	Vidutinis	Didelis	Vidutinis	Didelis
Suderinamumas su išorinėmis sistemomis	Paprastas	Sudėtingas	Vidutinis	Lengvas
Perimetro sauga	Yra	Gali būti	Yra	Gali būti
Veiksmų stebėjimas	Paprastas	Paprastas	Vidutinis	Priklauso nuo realizacijos
Greitaveika	Sparti	Sparti	Vidutinė	Vidutinė
Plečiamumas	Prastas	Efektyvus	Vidutinis	Vidutinis

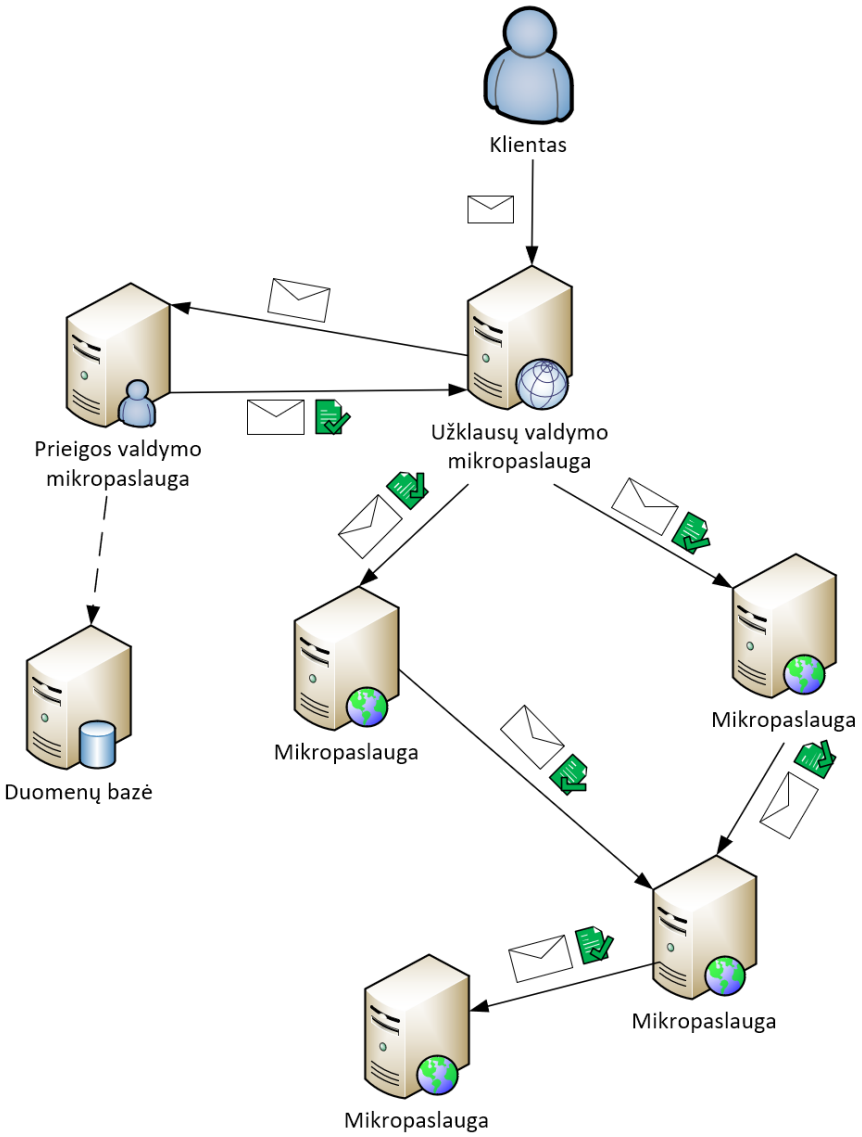
Prieigos valdymo metodo mikropaslaugų architektūroje analizės išvados

1. Analizės metu ne tik išanalizuoti prieigos valdymo metodai, bet ir pačios mikropaslaugų architektūros specifika. Darytina išvada, jog prieigos valdymo modelio realizacija privalo atsižvelgti ir į kitas problemas, tokias kaip plečiamumas, lankstumas ir greitaveika, su kuriomis susiduria mikropaslaugų architektūra;
2. Pastebėta, kad analizuoti identiteto valdymo metodai mikropaslaugų architektūros kontekste yra kompleksiški. Norint išvengti galimų realizacijos spragų ir palengvinti metodo realizaciją skirtingose programų sistemose, prieigos valdymo metodas turėtų būti kuo paprastesnis.
3. Pastebėta, jog išanalizuoti prieigos valdymo metodai iš skirtingų šaltinių turi bendrų požymių, kaip kad naudojamas tas pats API vartų infrastruktūrinis komponentas ar prieigos valdymo informacijos perdavimui naudojami JWT žetonai;
4. Darytina išvada, kad mikropaslaugų architektūra kuri pasitiki vien perimetro sauga komunikacijos konfidencialumui užtikrinti nėra visiškai saugi, nes išlieka atakos iš perimetro vidaus nutikimo rizika;

2. Prieigos valdymo metodo mikropaslaugų architektūroje projektas

2.1. Prieigos valdymo metodo mikropaslaugų architektūroje koncepcija

Remiantis analizės išvadomis, kuriamas mikropaslaugų architektūroje prieigos valdymo metodas privalo atsižvelgti į plečiamumo, lankstumo ir greitaveikos problemas. Kitaip tariant, kuriamo modelio kokybės kriterijai yra praktinis pritaikomumas (lankstumas), horizontalus plečiamumas ir kuo mažesnės laiko sąnaudos saugumo užtikrinimui. Sekantis tikslas – prieigos valdymo mechanizmas negali pasitikėti perimetro sauga. Darytina prielaida, kad programų sistemos architektūrinis išdėstymas yra viešas, t.y. serveriai ir mikropaslaugos gali būti viešai pasiekiamos. Todėl prieigos valdymo metodas privalo užtikrinti, kad kiekvienos užklausos validavimu mikropaslaugos rūpinsis pačios.



pav. 2.1 Prieigos valdymo metodas grindžiamas vidiniais žetonais

Todėl bene tinkamiausias metodas šiuo atveju yra vidinių žetonų strategija, kai kiekviena mikropaslauga validuoja užklausą remiantis papildomu žetonu, kuris buvo suteiktas autorizacijos serviso būtent šiai užklausai, kaip pavaizduota **pav. 2.1**. Šį metodą galima pagerinti įdiegiant papildomą išorinį žetoną, kuris bus suteikiamas klientui. Šis žetonas bus skirtas autentifikuoti klientą.

Klientas, turintis galiojantį autentifikacijos žetoną galės siųsti užklausą į programų sistemą. Tačiau prieš pradedant vykdyti užklausą, ji bus autorizuojama prie jos pridedant papildomą prieigos žetoną, kuriame yra nurodyta detali informacija apie paslaugas, kurias klientas gali pasiekti. Kai užklausa su vidiniu žetonu pasieks paslaugą, ši turės šį žetoną validuoti. Į validavimą įeis ne tik žetono galiojimo trukmės patikrinimas, bet ir autorizacija reikiamoms paslaugoms. Tokia prieigos valdymo strategija turi šiuos privalumus:

– Išorinė ir vidinė autorizacijos struktūra

Naudojantis vidiniais žetonais yra neišvengiamai sukuriama atskirtis tarp vidinių programų sistemos žetonų, kurie naudojami užklausose tarp paslaugų ir išorinių, kuriais naudojasi klientas, siųsdamas užklausą sistemai. Toks žetonų valdymas leidžia atskirti vidinio žetono struktūrą nuo išorinio, taip paslepiant nuo kliento tam tikras vidinės autorizacijos detales tokias kaip naudotojo identifikacinius duomenis, užklausų laukimo laikus, smulkias prieigos detales ar kitus kintamuosius, kurie turi reikšmę tik programų sistemai užklausos apdorojimo metu.

– Saugios sistemos ribos

Vidinių žetonų naudojimas užtikrina tarp mikropaslaugų esančio autorizacijos saugumo egzistavimą. Kitaip sakant, kiekviena užklausa privalo būti validuojama, todėl „viešos prieigos“ problema nesukelia papildomų saugumo rizikų. Tai ypač patogu, kai norima paslaugas diegti skirtingose aplinkose, serveriuose, tarp kurių užtikrinti išorinio perimetro saugą yra sudėtinga. O piktavaliui pralaužus išorinį perimetrą, jo galimybės vis tiek bus paribotos dėl griežtos užklausų validavimo (autorizavimo) politikos.

– Centralizuota autentifikacija ir autorizacija

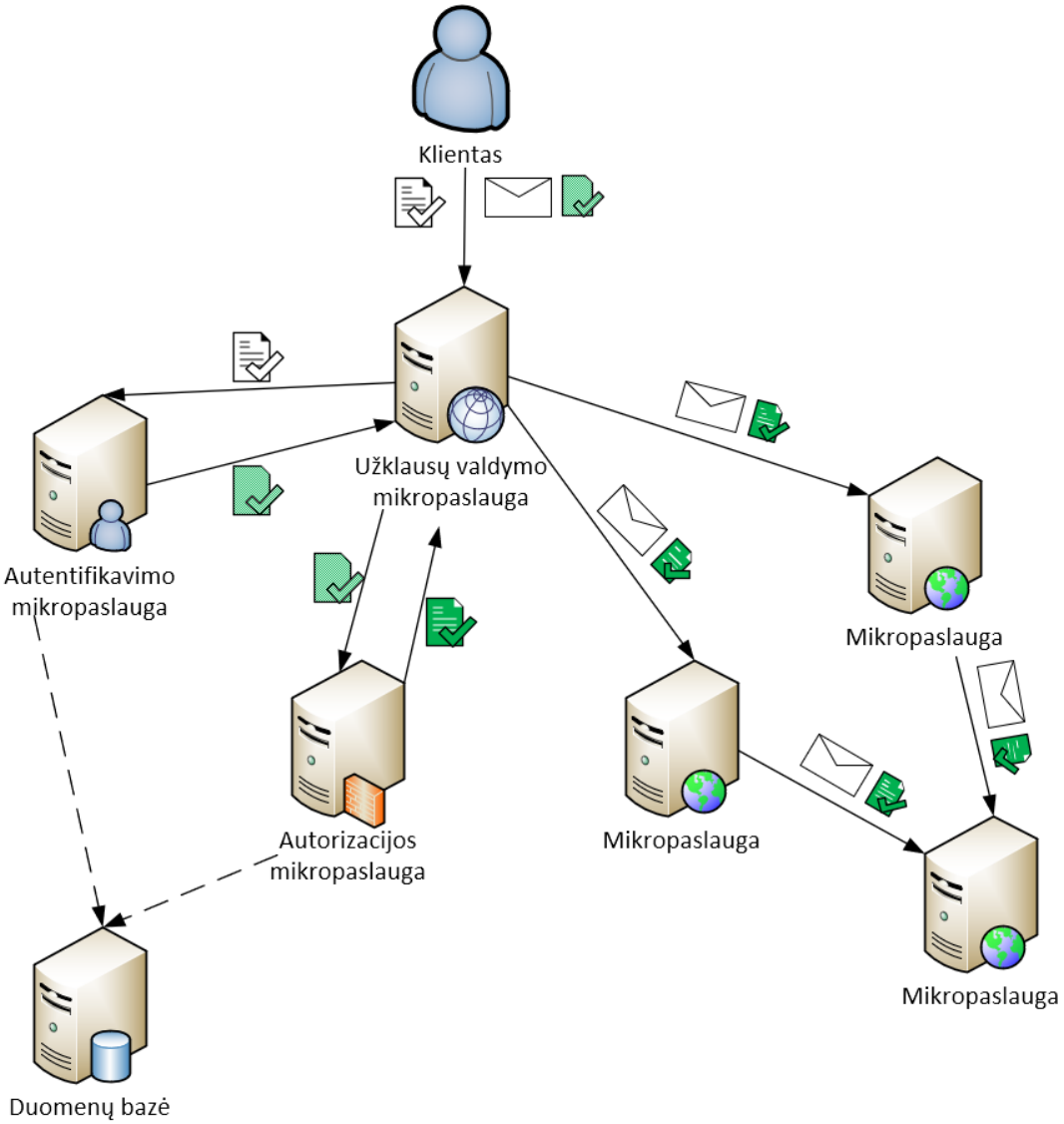
Centralizuota autentifikacija ir autorizacija užtikrina, kad klientas bus autentifikuojamas vieną kartą, o kiekviena jo užklausa bus autorizuojama vieno serviso. Tai reiškia, jog mikropaslaugoms, gavusioms vidinę užklausą liks tik validuoti pačią užklausą. Tai sumažina laiko kaštus, nes priešingu atveju kiekviena paslauga turėtų autentifikuoti ir autorizuoti naudotoją atskirai.

– Paprastesnis veiksmų registravimas

Nepaisant to, kad mikropaslaugų programų sistemoje gali būti didelė aibė, tačiau naudotojo autentifikacija ir autorizacija atliekama tik atitinkamuose servisuose. Tai reiškia, jog ne tik patogu registruoti kliento tapatybės nustatymo faktą, bet ir jam suteiktas prieigos teises. Papildomai galima pridėti unikalų užklausos identifikatorių, kuris bus egzistuos vidinio žetono viduje. Šis identifikatorius galėtų leisti susieti autorizacijos serviso veiksmų žurnalą su mikropaslaugų veiksmų žurnalais ir taip suteikti galimybę atsekti galimybę ne tik kada naudotojai buvo autentifikuoti ar autorizuoti, bet ir kokius konkrečius veiksmus atliko jų užklausos.

Bene esminis trūkumas yra greitaveika. Dėl papildomų prieigos valdymo laiko kaštų, reikalingų kiekvienai kliento užklausai, gali susidaryti sąlyginai didelis uždelsimo laikas, ypač jeigu klientas per trumpą laiko tarpą siunčia daug užklausų. Tačiau šią problemą galima dalinai išspręsti būdais, pavaizduotais **pav. 2.2**. Pirmasis jų - autentifikavimo paslaugai nesiųsti visos užklausos, o tik esminę jos dalį, susijusią su kliento tapatybe. Sekantis – skaidyti prieigos valdymo paslaugą į autentifikacijos ir autorizacijos paslaugas. Tokiu būdu bus tolygiai paskirstoma apkrova, reikalinga prieigos valdymui. Kalbant apie autorizavimo politiką, galima suteikti naudotojui teisę atlikti tokią pačią užklausą, jeigu tokia pati užklausa buvo sėkmingai autorizuota nepraėjus atitinkamam laiko tarpui.

Taip pat galima didinti užklausos galiojimo laiką, atsižvelgiant į panaudojimo atvejį. Reikėtų įvertinti tiek vidutinį bendrą užklausos apdorojimo laiką programų sistemoje, tiek įprastinį naudotojų užklausų kiekį per atitinkamą laiko tarpą. Taigi, šiais būdais įmanoma padidinti greitaveiką, tačiau tai daryti reikia atsargiai, nes abu šie sprendimai daro neigiamą įtaką bendram programų sistemos saugumui.

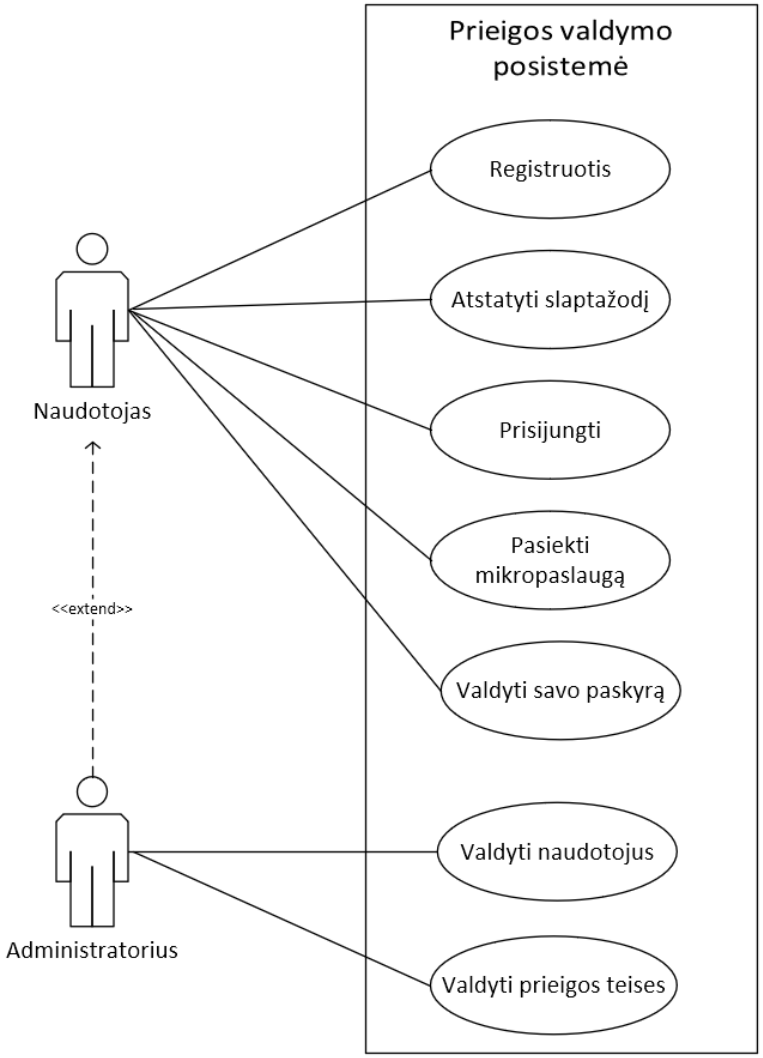


pav. 2.2 Siūlomo prieigos valdymo metodo mikropaslaugų architektūroje koncepcija

2.2. Prieigos valdymo metodo funkcijos

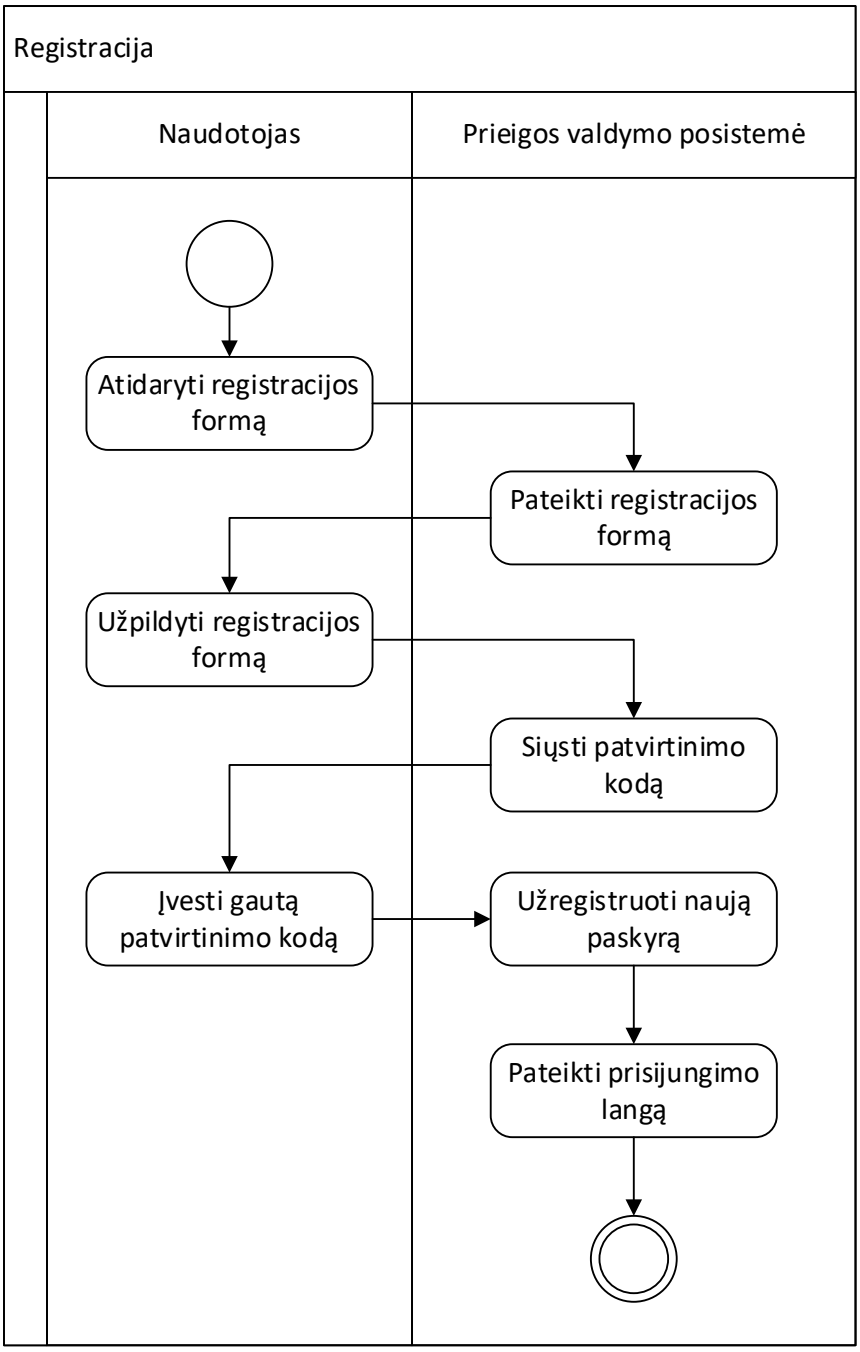
Kaip pavaizduota **pav. 2.2**, prieigos valdymo metodas tėra viena didesnės mikropaslaugų architektūros dalis. Todėl šiame skyriuje bus specifikuojamos tik prieigos valdymo metodo funkcionalumas, kuris pavaizduotas **pav. 2.3**. Paprastas naudotojas galės atlikti keturis pagrindinius veiksmus – užsiregistruoti, atstatyti prarastą slaptažodį, prisijungti, valdyti savo paskyros nustatymus ir pasiekti mikropaslaugą. Papildomai administratorius galės atlikti visas naudotojo funkcijas ir papildomai valdyti naudotojus bei jų prieigos teises. Toliau šiame skyrelyje detalizuojami naudotojo panaudojimo atvejai. Svarbu atkreipti dėmesį, kad šio skyrelio veiklos diagramose yra atvaizduojamas pagrindinis kelias darant prielaidą, kad visi veiksmai yra atliekami sėkmingai. Turima omeny, kad sistema reaguos į klaidas (pvz.: naudotojui pateikus neteisingus duomenis, žetoną

ir kt.), tačiau norint neapsunkinti veiklos diagramų tokie veiksmai kaip klaidos aptikimas, jos užregistravimas ir klaidos pranešimo pateikimas naudotojui ir tos veiklos dalies pakartojimas nebus atspindėti šio skyrelio diagramose.



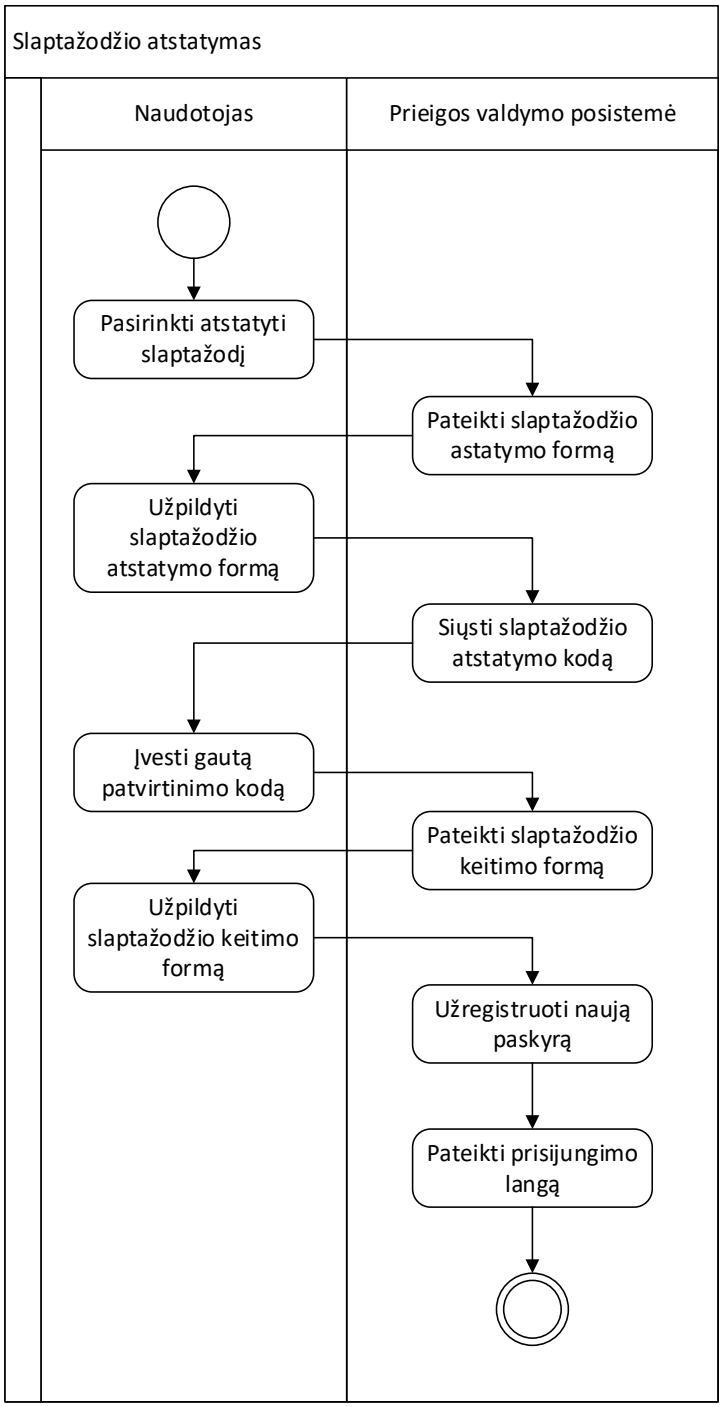
pav. 2.3 Prieigos valdymo posistemės panaudojimo atvejų diagrama

Registracijos metu naudotojas, kaip nurodyta **pav. 2.4**, turės užpildyti registracijos formą su reikiamais duomenimis ir ją pateikti prieigos valdymo posistemei. Jeigu registracijos formoje nurodytų unikalių identifikatorių (pvz.: elektroninio pašto adreso, telefono numerio ir kt.) nėra randama tarp egzistuojančių naudotojų, tuomet prieigos valdymo posistemė siunčia patvirtinimo kodą į bent vieną iš nurodytų komunikacijos identifikatorių. Konkretus komunikacijos kanalas šiuo atveju nėra svarbus, tačiau jis privalo būti specifikuotas realizacijos projekte. Naudotojui sėkmingai suvedus patvirtinimo kodą, paskyra su nurodytais duomenimis yra sėkmingai sukuriama sistemoje ir naudotojui pateikiamas prisijungimo langas.



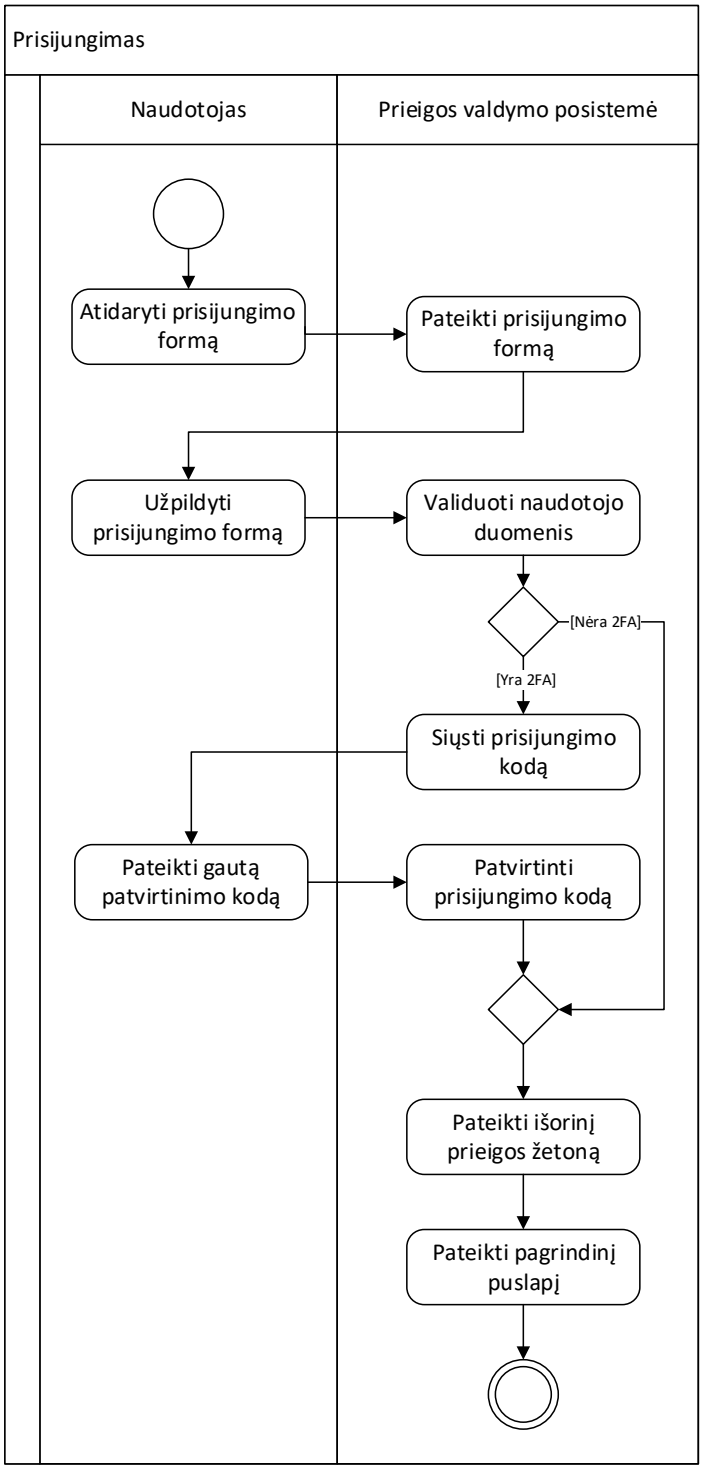
pav. 2.4 Registracijos veiklos diagrama

Slaptažodžio atstatymo metu naudotojui taip pat teks naudoti patvirtinimo kodą, norint susikurti naują slaptažodį, kaip kad pavaizduota **pav. 2.5.** Slaptažodžio atstatymui taip pat populiaru naudoti kontrolinius klausimus, kuriuos susikuria pats naudotojas individualiai, tačiau tai priverčia naudotoją juos prisiminti arba išsisaugoti. Šiuos klausimus praradus arba pamiršus, vis tiek teks naudotis kitomis autentifikavimo priemonėmis, todėl šios idėjos buvo atsisakyta. Atstatymo kodai gali būti siunčiami įvairiais išoriniais kanalais, tokiais kaip el. paštas, autentifikavimo programėlės (pvz.: *Google Authenticator*), trumposios žinutės ir t.t. Kanalas, kuris bus naudojamas atstatymo kodams priklauso nuo konkrečios realizacijos reikalavimų.



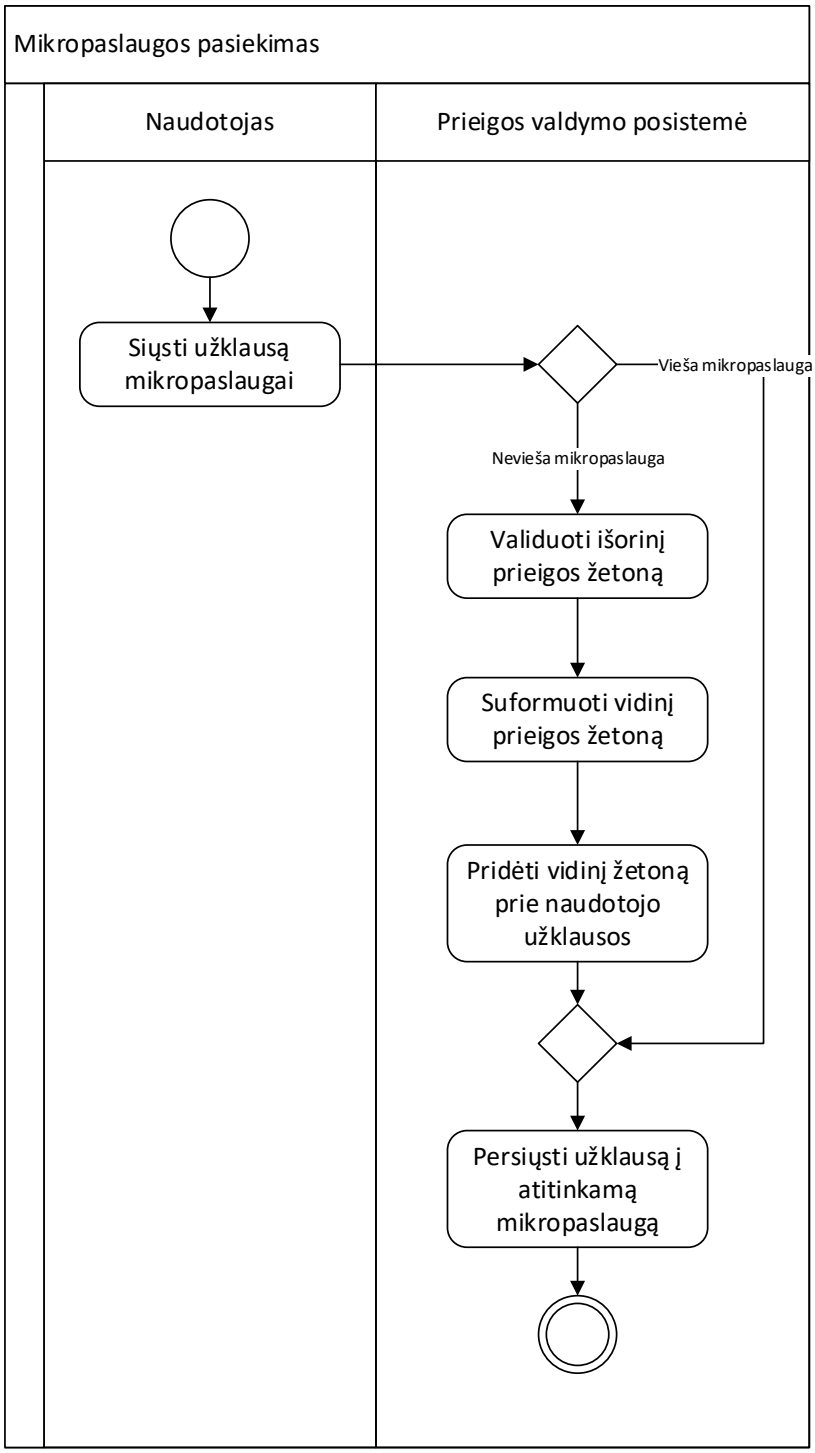
pav. 2.5 Slaptažodžio atstatymo veiklos diagrama

Naudotojas, kuris paskyros valdymu metu yra pasirinkęs aktyvuoti 2 Faktorių Autentifikaciją (**pav. 2.6**), prisijungimo metu papildomai turės pateikti prisijungimo kodą, kurį jam suteiks prieigos valdymo posistemė. Tai yra papildomas apsaugos būdas nuo neautorizuoto prisijungimo prie naudotojo paskyros, kuris šiais laikais laikomas įprastiniu standartu. Svarbu paminėti, kad tik registruoti naudotojai gali prisijungti prie sistemos. Išorinis prieigos žetonas yra perduodamas naudotojui, kuris jį ir valdys. Šis žetonas turi bazinę informaciją, reikalingą naudotojo identitetui nustatyti bei pasirinktinai turės bazinę informaciją, kuri neišsamiai apibūdina naudotojo privilegijas sistemoje.



pav. 2.6 Prisijungimo veiklos diagrama

Naudotojas, norėdamas pasiekti mikropaslaugą siunčia užklausą kurią pirmiausiai apdoroja prieigos valdymo posistemė, kaip kad pavaizduota **pav. 2.7**. Jeigu mikropaslauga yra vieša ir ji nereikalauja autorizacijos, tuomet užklausa yra tiesiog persiunčiama į ją. Priešingu atveju, kai autorizacija yra reikalinga, prieigos valdymo posistemė validuoja išorinį prieigos žetoną, kurį naudotojas gavo prisijungimo metu. Tuomet suformuojamas vidinis prieigos žetonas ir jis pridedamas prie naudotojo užklausos, kuri galiausiai persiunčiama toliau. Būtent šį vidinį žetoną kitos mikropaslaugos ir naudos, norėdamos autorizuoti naudotojo veiksmus.



pav. 2.7 Mikropaslaugos pasiekimo veiklos diagrama

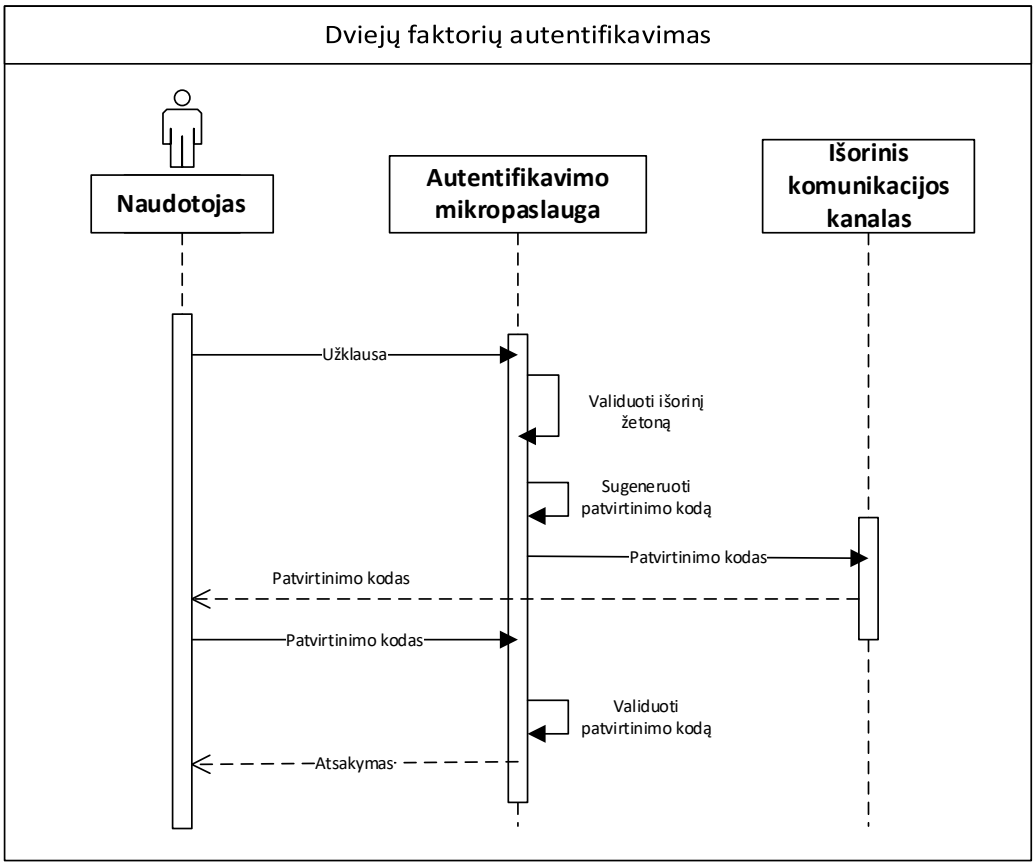
2.3. Autentifikacijos mikropaslaugos funkcijos

Kaip pavaizduota konceptualiame modelyje (**pav. 2.2**), autentifikacijos funkcijų atlikimu rūpinsis atskira mikropaslauga. Į šį funkcijų sąrašą įeina:

- Naujo naudotojo kūrimas
- Paskyros valdymas
- Slaptažodžio atstatymas
- Išorinio žetono išdavimas (Prisijungimas)

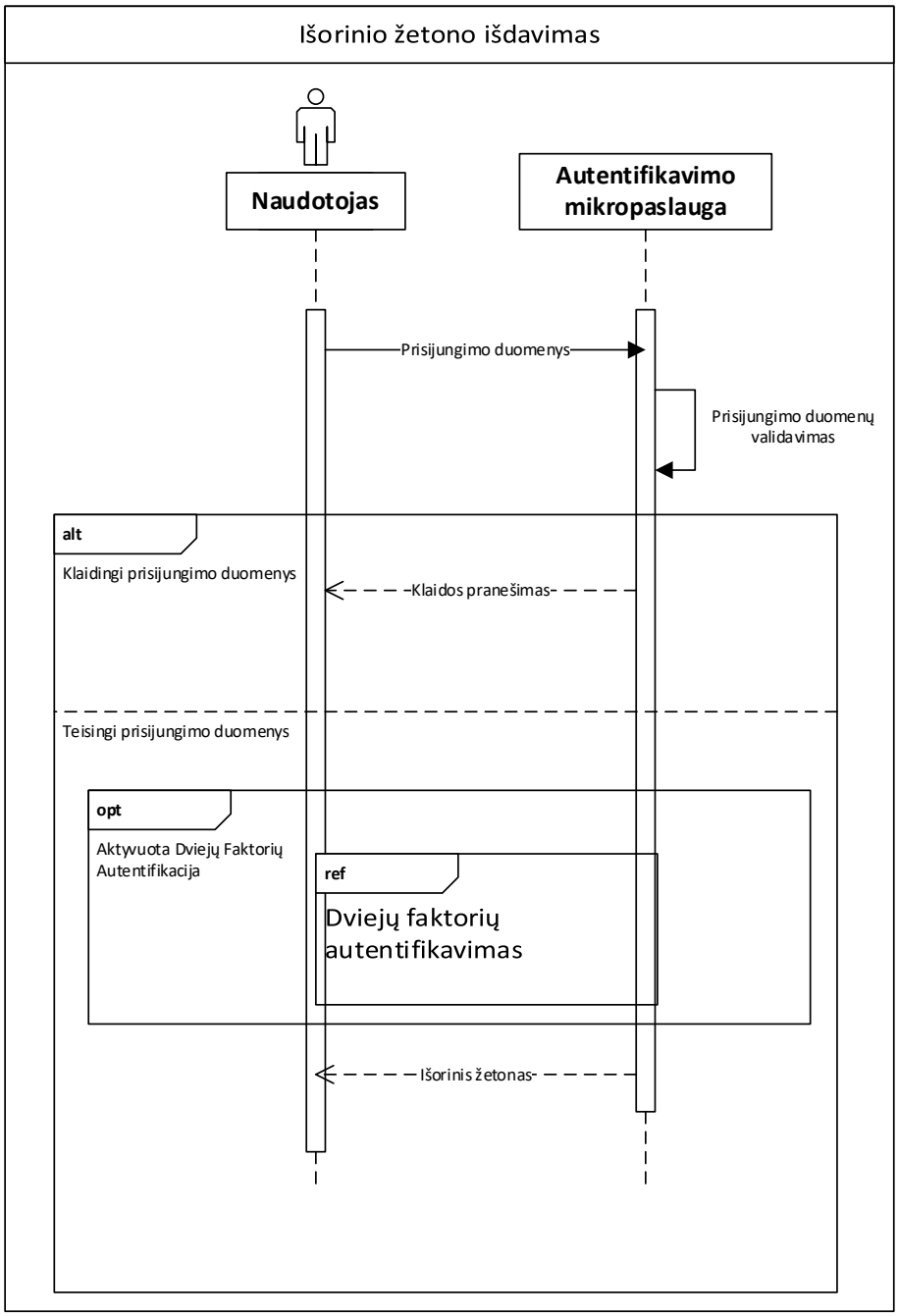
- Naudotojų valdymas
- Prieigos teisių valdymas

Visiems šiems veiksmams, išskyrus išorinio žetono išdavimui, gali būti naudojamas dviejų faktorių autentifikavimas, kurio sekų diagrama pavaizduota **pav. 2.8.** Išorinis komunikacijos kanalas priklauso nuo konkrečios realizacijos, tačiau tai gali būti elektroninio pašto sistema, trumpųjų žinučių paslauga ir kt. Šiuo išoriniu kanalu yra siunčiamas patvirtinimo kodas, kurį naudotojas pateikia autentifikavimo paslaugai. Tai papildomas autentifikavimo žingsnis, kuris padeda apsisaugoti nuo neautorizuotos prieigos. Įsitikinus, kad sugeneruotas ir naudotojo pateiktas patvirtinimo kodai sutampa, naudotojui grąžinamas atsakymas priklausomai nuo užklausos, kurią jis pateikė. Tai reiškia, kad kiekvieno panaudos atvejo metu atsakymas naudotojui bus skirtingas. Įvykus nenumatytam atvejui bet kuriame žingsnyje, naudotojui bus rodomi klaidos pranešimai, kurie priklausys nuo konkrečios realizacijos.



pav. 2.8 Dviejų faktorių autentifikavimo sekų diagrama

Kaip dviejų faktorių autentifikavimas naudojamas išorinio žetono išdavimo metu pavaizduota **pav. 2.9**. Norint gauti išorinį žetoną, privalu pateikti teisingus prisijungimo duomenys. Jeigu naudotojo paskyrai yra aktyvus dviejų faktorių autentifikavimas, tuomet sekama žingsniais, nurodytais **pav. 2.8**. Sėkmės atveju naudotojui yra grąžinamas išorinis žetonas, su kuriuo naudotojas vėliau galės pasiekti norimus resursus. Kitų funkcijų atžvilgių autentifikavimo sistema iš esmės laikosi tokio pačio proceso su dviejų faktorių autentifikavimu, todėl jų atskirai projekto specifikacijoje detalizuoti nereikia. Tačiau administratoriaus funkcijos, kurios atliekamos naudojantis autentifikavimo mikropaslauga, yra detalizuojamos skyrelyje 2.3. Taip pat paprastumo dėlei šio skyrelio sekų diagramose nėra vaizduojamas užklausų valdytojo mikropaslauga, per kurią vyksta komunikacija tarp naudotojo ir visų kitų mikropaslaugų, nes jis šiuo atveju tėra tarpininkas, kuris neturi įtakos bendram autentifikavimo procesui.

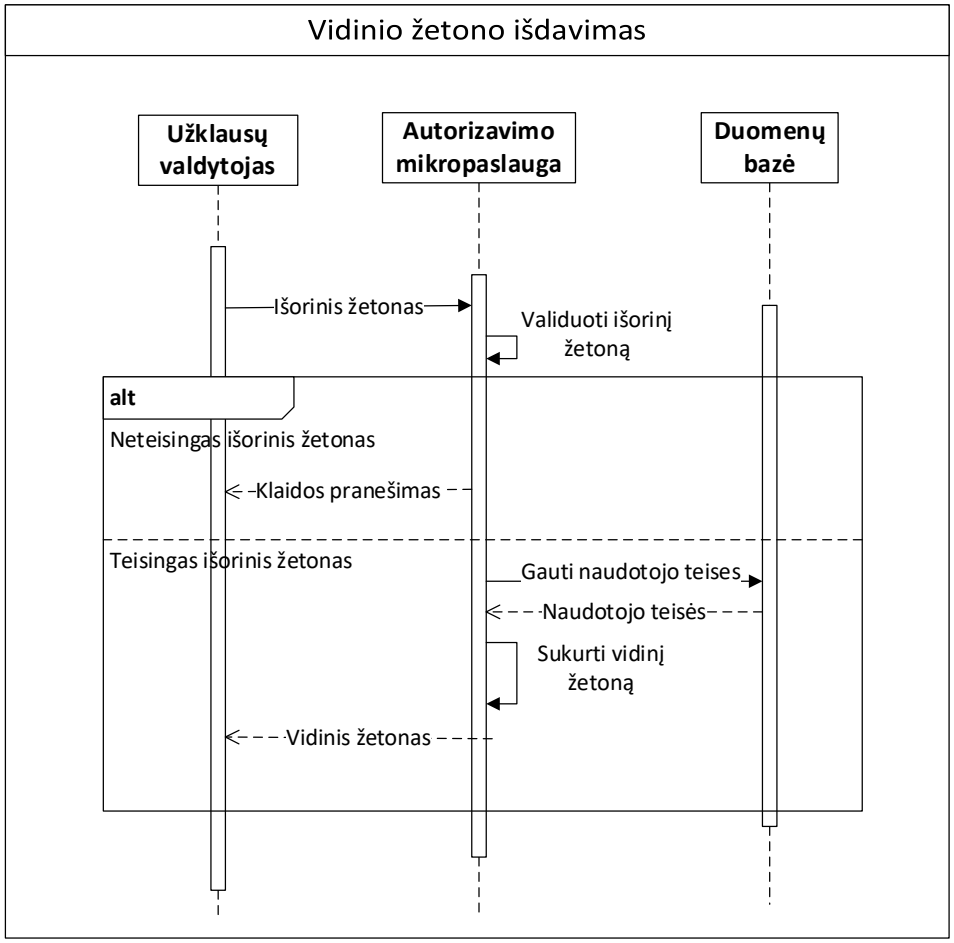


pav. 2.9 Išorinio žetono išdavimo sekų diagrama

2.4. Autorizacijos mikropaslaugos funkcijos

Autorizacijos mikropaslauga yra išimtai atsakinga tik už **pav. 2.3** pavaizduotą mikropaslaugos pasiekimo panaudojimo atvejį. Tai reiškia, jog autorizacijos mikropaslaugos paskirtis yra išduoti vidinį žetoną, kurį vėlesniuose užklausos gyvavimo etapuose naudos kitos sistemos mikropaslaugos, kaip pavaizduota **pav. 2.10.** Jeigu nepavyksta validuoti išorinio žetono, kurį naudotojui išdavė autentifikavimo mikropaslauga, tuomet autorizavimo mikropaslauga atsako užklausų valdytojui su klaidos pranešimu. Priešingu atveju, kai išorinis žetonas validuojamas sėkmingai, tuomet yra siunčiamas pranešimas su sugeneruotu vidiniu žetonu kuris turi detalią informaciją apie visas naudotojo turimas teises.

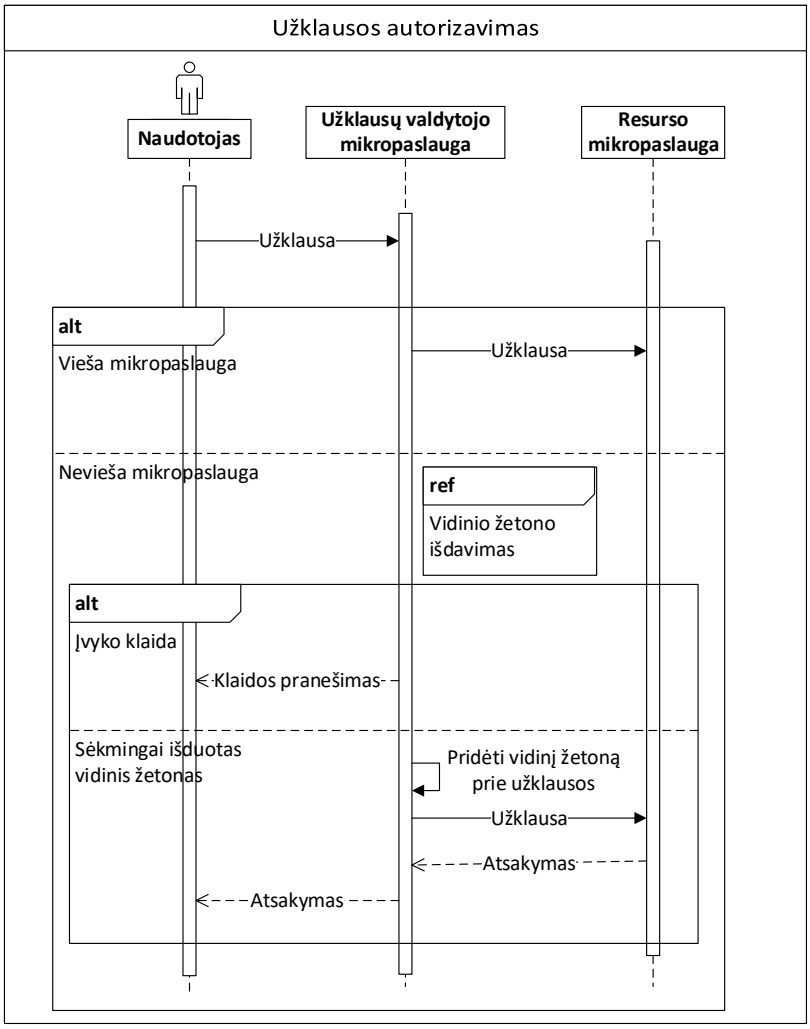
Vidinio žetono kūrimo metu galima įterpti ne tik informaciją apie naudotojo privilegijas, gautas iš duomenų bazės, bet ir apibendrintą sistemos informaciją, kuri būtų tolimesniame užklausos autorizavime. Tarkime, jeigu naudotojas paprašo el. pašto mikropaslaugos išsiųsti visiems naudotojams el. laiškus, autorizacijos mikropaslauga gali papildyti teisių sąrašą apribojimu atlikti tik tam tikrą kiekį veiksmų per atitinkamą laiko tarpą. Šis apribojimas gali varijuoti nuo, tarkime, bendro sistemos užimtumo, paros laikos ir kitų išorinių aplinkybių. Toks apribojimas, kuris valdo atliekamų veiksmų greitaveiką privalo priklausyti autorizacijos mikropaslaugai, nes tas pats apribojimas gali būti naudojamas keliose resursų mikropaslaugose. Todėl tokia funkcija yra labai palanki, kai sistema atlieka tokias funkcijas, kurioms daro įtaką išoriniai faktoriai. O žetono papildymas šiais apribojimais leidžia lanksčiai išspręsti šią problemą.



pav. 2.10 Vidinio žetono išdavimo sekų diagrama

Tačiau vidinio žetono išdavimas tėra žingsnis bendrame užklausos autorizavimo procese, kuris pavaizduotas **pav. 2.11**. Taip yra todėl, nes tarp naudotojo ir autorizavimo mikropaslaugos yra užklausų valdytojas, kuris taip pat atlieka svarbų vaidmenį. Dėl efektyvumo padidinimo atsakomybė nukreipti užklausas, kurioms nereikia autorizacijos tiesiai į resursus tenka būtent užklausų valdytojui. Priešingu atveju tektų papildomai siųsti užklausą į autorizacijos serverį, kur būtų visuomet išduodamas vidinis žetonas, todėl padidėtų bendras užklausos apdorojimo laikas. Tačiau ar toks sprendimas pasiteisintų priklauso nuo realizacijos, nes nors resursas ir viešas, bet jam vis tiek turėtų būti nurodomi papildomi apribojimai, kurie nepriklauso nuo naudotojo ir juos sukuria autorizacijos mikropaslauga. Todėl viešų mikropaslaugų sąrašo įtraukimas į užklausų valdytojo logiką yra nebūtinas, o rekomenduotinas.

Sekantis veiksmas iš **pav. 2.11**, kurį atlieka užklausų valdytojas yra sėkmingai sugeneruoto vidinio žetono pridėjimas prie naudotojo užklausos. Vėlesniuose etapuose būtent šis vidinis žetonas bus naudojamas autorizuojant naudotoją konkrečioms funkcijoms. Turint pilną užklausą, užklausų valdytojo mikropaslauga siunčia atitinkamam resursui. Po kurio laiko resursas pateikia atsakymą, kurį užklausos valdytojas perduoda naudotojui. Akivaizdu, kad užklausų valdytojas gali tapti greitaveikos lėtintoju, todėl labai svarbu realizacijos metu užtikrinti šios mikropaslaugos darbo asinchroniškumą. Tai reiškia, jog kiekvienas užklausos atsakymo laukimas negali blokuoti kitų užklausų apdorojimo.



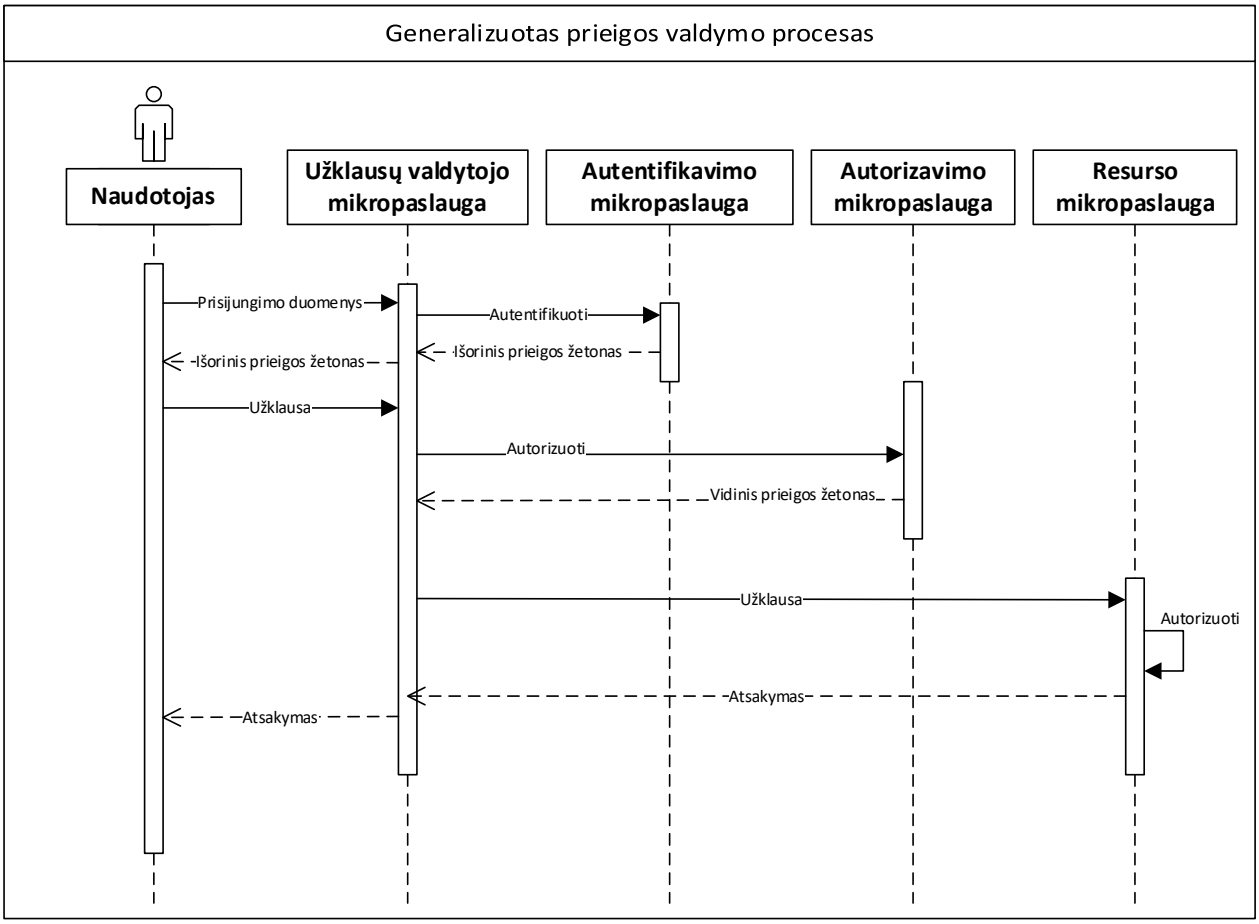
pav. 2.11 Užklausos autorizavimo sekų diagrama

2.5. Bendro mikropaslaugų prieigos valdymo proceso detalizacija

Kitose šiame skyriuje esančiose sekų diagramose (**pav. 2.8**, **pav. 2.9**, **pav. 2.10**, **pav. 2.11**) yra detalizuoti autentifikavimo ir autorizacijos procesai izoliacijoje. Todėl **pav. 2.12** vaizduojamas generalizuotas mikropaslaugų prieigos valdymo procesas. Šioje diagramoje pavaizduotas užklausos gyvavimo ciklas, kuriame dalyvauja naudotojas ir užklausų valdytojo, autentifikavimo, autorizavimo, bei resurso mikropaslaugos. Svarbu atkreipti dėmesį, kad šioje diagramoje vaizduojamas sėkmingas scenarijus, o įvairūs nenumatyti ar klaidų atvejai neįtraukiami. Jie klaidos gali būti apdorojamos skirtingai, remiantis konkrečios sistemos atveju, kuris bus detalizuotas trečiajame skyriuje.

Šiame prieigos metode svarbų vaidmenį atlieka užklausų valdytojas, kuris yra tarpininkas tarp naudotojo ir sistemos, kuri grindžiama mikropaslaugų architektūra. Papildomas žingsnis, kurį atlieka užklausų valdytojas, tai autorizavimo metu sugeneruoto vidinio žetono pridėjimas prie užklausos ir tolimesnis jos siuntimas į reikiamą resurso mikropaslaugą.

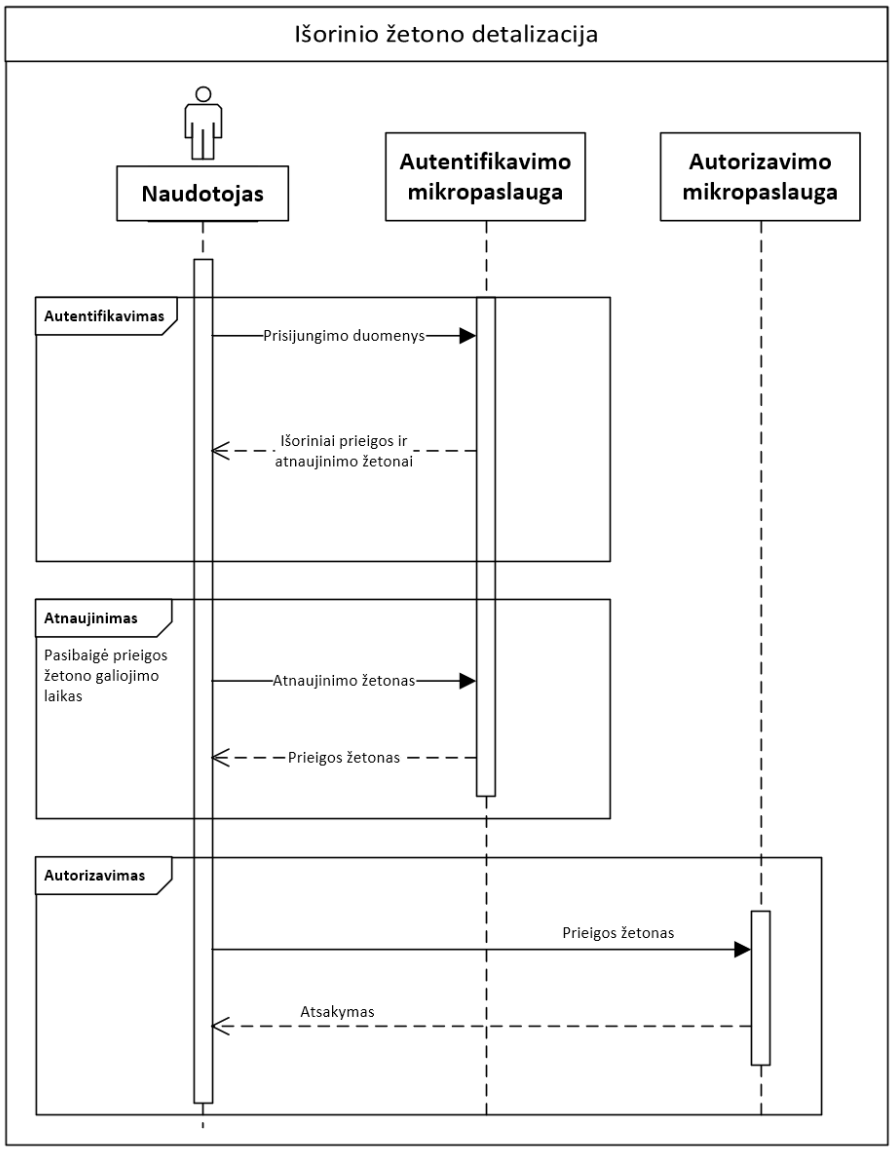
Autentifikavimo metu išduodamas išorinis žetonas yra supaprastintas informacijos agregatas, kuris yra atiduodamas naudotojo valdymui. Naudotojas privalo šį žetoną siųsti su vėlesnėmis užklausomis, kad autorizavimo mikropaslauga galėtų sėkmingai identifikuoti kieno vardu užklausa buvo išsiųsta. Autorizavimo mikropaslauga sukuria tokį prieigos žetoną, kuris turi detalią informaciją apie naudotoją, kuri yra būtina užklausai sėkmingai įgyvendinti resurse. Galiausiai resurso mikropaslauga autorizuoja naudotoją ir remiantis vidiniu žetonu ir įvykdo naudotojo užklausą.



pav. 2.12 Generalizuoto mikropaslaugų prieigos valdymo proceso sekų diagrama

2.6. Prieigos valdyme naudojamų žetonų specifikacija

Remiantis atlikta analize, šio prieigos valdymo metodo mikropaslaugų architektūroje atveju geriausiai pritaikomas JWT žetonas. Esminis privalumas – būsenos nebuvimas. Todėl prieigos valdymo proceso metu išduotas vidinis ar išorinis žetonas bus aktyvus tol, kol jis pats nenustos galioti. Todėl yra labai svarbu apibrėžti skirtingų žetonų galiojimo laikus. Sekantis svarbus aspektas yra išorinio žetono specifikavimas. Nepaisant to, kad išorinis žetonas iš esmės yra viena esybė, ją galima skaidyti į keletą nepriklausomų dalių. JWT atveju išorinį žetoną galėtų sudaryti prieigos ir atnaujinimo žetonai. Išorinis atnaujinimo žetonas būtų siunčiamas autentifikavimo mikropaslaugai su tikslu gauti naują prieigos žetoną, kai senasis baigia galioti. Tuo tarpu išorinis prieigos žetonas būtų siunčiamas į sistemą autorizacijos tikslais. Šis procesas pavaizduotas **pav. 2.13**, tačiau vengiant diagramos komplikavimo, neatvaizduojamas užklausų valdytojo vaidmuo šiame procese, nes jis lieka tik kaip tarpininkas ir papildomos logikos šiuo atveju neturi. Šioje diagramoje pavaizduota, kaip būtų galima atskirti išorinius prieigos ir atnaujinimo žetonus bei naudoti skirtingais tikslais.



pav. 2.13 Išorinio prieigos žetono detalizacijos sekų diagrama

Remiantis šiuo procesu, prieigos valdymo metode iš viso naudojami trys JWT žetonai- išoriniai atnaujinimo ir prieigos bei vidinis prieigos. Išoriniai žetonai atiduodami naudotojui, tad turi didesnę

galimybę būti pažeistiems. Todėl išoriniai žetonai turėtų naudoti ilgesnius raktus ir parašą. Tačiau vidinis žetonas turėtų būti naudojamas tarp perimetre esančių mikropaslaugų su trumpu galiojimo laiku, todėl atsižvelgiant į greitaveiką, galima naudoti trumpesnį raktą ir parašą. Todėl JWT žetono parašo algoritmu pasirinktas *EdDSA* elipsinės kreivės algoritmas, kuris iš pasižymi greitaveika [12]. Išoriniams žetonams naudojamas *Ed448* (448 bitų), o vidiniam – *Ed25519* (255 bitų) elipsinės kreivės parašo algoritmo versijos (**lentelė 2.1**). Žetonų galiojimo laikas priklauso nuo konkrečios realizacijos ir privalo būti lengvai keičiamas (per konfigūraciją). Tačiau galima apibrėžti, kad vidinis prieigos žetonas turi galioti trumpiausiai (vienos užklausos gyvavimo metu), o išorinis atnaujinimo žetonas – ilgiausiai.

lentelė 2.1 JWT Žetonų specifikacija

Pritaikymas	**Tipas**	**EdDSA versija**	**Galiojimo laikas**	**Paskirtis**
Išorinis	Prieigos	Ed448	Vidutinis	Autorizavimui
Išorinis	Atnaujinimo	Ed448	Ilgas	Prieigos atnaujinimui
Vidinis	Prieigos	Ed25519	Trumpas	Resurso užklausos įvykdymui

Informaciją, kuri privalo būti naudojama skirtinguose žetonuose, vaizduoja **lentelė 2.2**. Lentelėje frazės nurodo konkrečias reikšmes, kurios turi būti atitinkamuose žetonų laukuose, pliusas simbolizuoja lauko ir reikšmės buvimą, o minusas simbolizuoja, kad laukas neturi reikšmės atitinkamam žetonui ir neturėtų būti į jį įtrauktas. Apart standartinių *iss, sub, aud, exp, iat, jti* JWT žetono laukų [13], papildomai pridedami *kid, role* ir *claims* laukai. Rakto identifikatorius reikalingas nustatyti, kuriuo simetriniu raktu žetoną reikia validuoti. Kadangi skirtingais laiko momentais to pačio JWT žetonai gali būti išduodami skirtingais raktais, šis laukas užtikrins, kad žetonai bus validuojami su tuo raktu, kuriuo ir buvo išduotas žetonas. Tačiau tam reikia papildomai sekti raktų identifikatorių. Nepaisant to, ši strategija leis sistemoje rotuoti raktus (pvz. pakeisti galimai pažeistą raktą) nesukeliant klientams didelių nepatogumų – vienu metu klientams bus išduodami žetonai su nauju simetriniu raktu, o tuo tarpu jau išduoti žetonai bus validuojami su senuoju iki tol, kol visi žetonai nustos galioti. Tai taip pat padidina sistemos saugumą, nes atsiranda galimybė naudoti aibę raktų ir juos nuolatos rotuoti, o tai apsunkina galimybę piktavaliams kriptografinės analizės būdais analizuojant žetoną nuspėti slaptąjį raktą, kurį naudoja serveris. Prieigos valdymas yra lankstesnis, jei naudotojas gali turėti keletą rolių. Šios rolės nėra reikalingos išoriniam atnaujinimo ar vidiniam prieigos žetonams, tačiau būtinos išoriniam prieigos žetonui. Pagal subjekto roles, resurso mikropaslauga žinos bazines subjekto galimybes ir jomis remiantis leis arba uždraus naudotojui atlikti tam tikrus veiksmus sistemoje. Leidimų laukas privalomas vidiniam prieigos žetonui, nes jame bus specifikuota detali autorizacijos informacija susijusi su subjekto galimybėmis kiekvienoje mikropaslaugoje. Šią informaciją iš dalies galima pateikti ir išoriniame žetone, jeigu to reikalauja panaudojimo atvejis. Taip pat išskiriamas sąlygų užklausoms laukas, kuriame yra detalizuojamos išorinės sistemos sąlygos užklausai, kurios nepriklauso tiesiogiai nuo naudotojo. Tai, pavyzdžiui, galėtų būti greitaveikos apribojimai tam tikrai mikropaslaugai dėl ir taip didelio sistemos užimtumo ar pan. Svarbu, kad ši informacija būtų centralizuotai valdoma, nes teoriniame lygmenyje, mikropaslaugos yra izoliuotos ir tarpusavyje viena apie kitos būseną neturėtų nieko žinoti.

lentelė 2.2 Naudojami JWT žetonų laukai

Laukas	Pavadinimas		Išorinis prieigos	Išorinis atnaujinimo	Vidinis prieigos
iss	Issuer	Žetono išdavėjas	MicroAC:Authentication Service	MicroAC:Authentication Service	MicroAC:Authorization Service
sub	Subject	Žetono subjektas	MicroAC:User	MicroAC:User	MicroAC:Request
aud	Audience	Žetono auditorija	MicroAC:Authorization Service	MicroAC:Authorization Service	MicroAC:Service
exp	Expiration Time	Galiojimo laikas	+	+	+
iat	Issued at	Išdavimo laikas	+	+	+
jti	Žetono ID (JWT ID)	Žetono identifikatorius	+	+	+
kid	Key ID	Rakto identifikatorius	+	+	+
sroles	Subject Roles	Subjekto rolės	+	-	-
claims	Subject Claims	Subjekto leidimai	+	-	+
cnd	Conditions	Sąlygos užklausai	-	-	+

Norint užtikrinti sistemos saugumą, privalu apibrėžti žetonų validavimo taisykles. Jos remiasi gerosiomis praktikomis, užkerta kelią piktavaliams pasinaudoti tam tikromis silpnybėmis bei apibrėžia aiškius validavimo proceso reikalavimus programinės įrangos kūrėjams, kurie integruos siūlomą metodą į savo programų sistemą. Taisyklėms yra svarbi tikrinimo tvarka tik greitaveikos atžvilgiu:

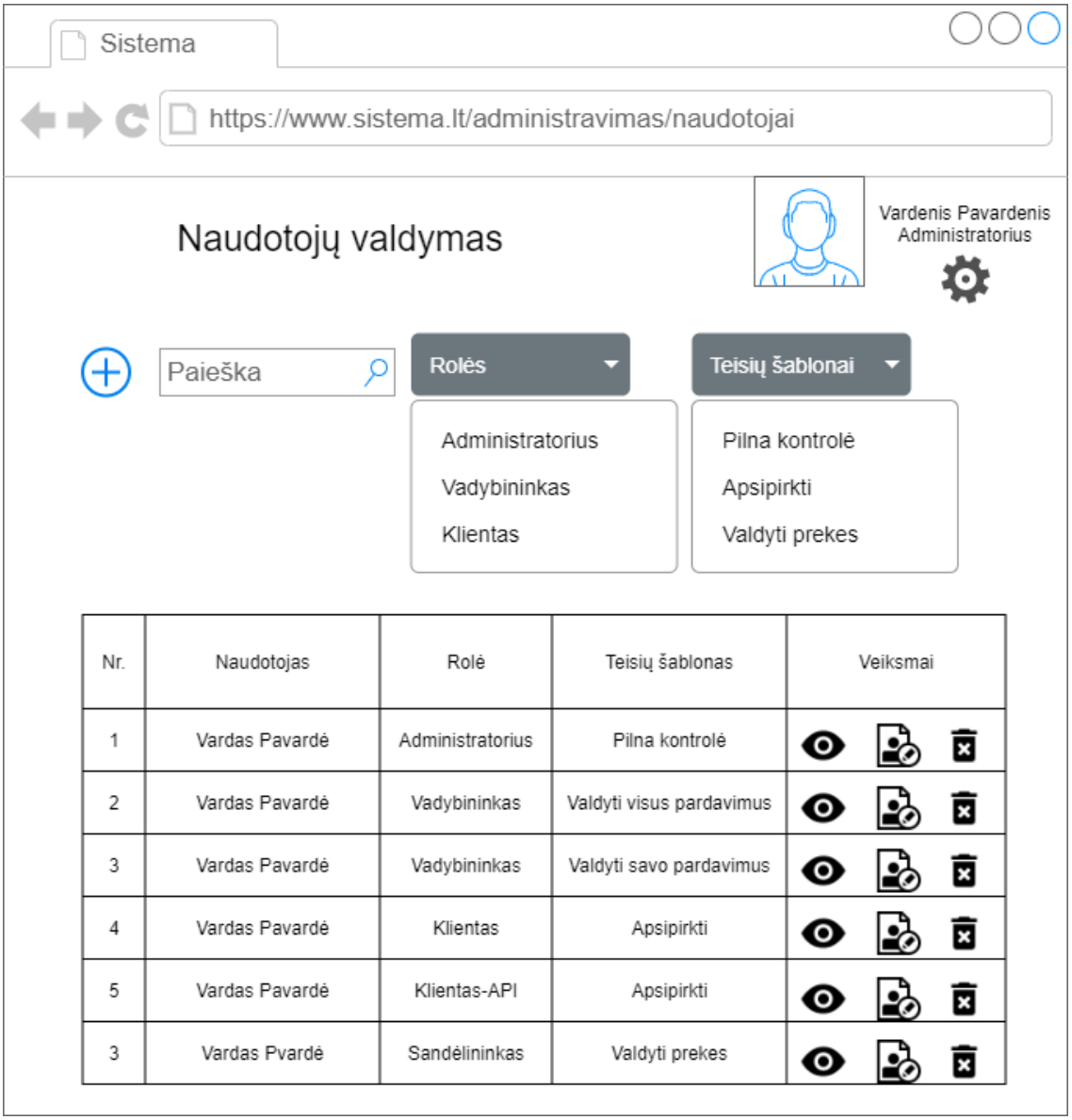
1. Atlikti JWT validavimo veiksmus, nurodytus RFC specifikacijoje [13], 7.2 skyrelyje.
2. Patikrinti, ar žetono antraštės algoritmo lauke nurodytas atitinkamas *EdDSA* algoritmas, kaip nurodyta **lentelė 2.1**.
3. Patikrinti, ar žetono parašas yra teisingas, pasitelkiant raktą, kurio identifikatorius yra nurodytas žetone.
4. Patikrinti, ar žetone nurodyti subjekto, auditorijos ir išdavėjo laukai yra tokie, kokie nurodyti **lentelė 2.2**.
5. Autentifikavimo ir autorizavimo mikropaslaugose patikrinti, ar naudotojo identifikatorius sutampa su nurodytu duomenų bazėje.
6. Jeigu realizuotas žetono atšaukimas, autentifikavimo ir autorizavimo mikropaslaugose patikrinti, ar žetono galiojimo identifikatorius sutampa su nurodytu duomenų bazėje.

2.7. Administratoriaus vaidmuo prieigos valdymo posistemėje

Pagal panaudojimo atvejų diagramą, pavaizduotą **pav. 2.3**, administratorius gali valdyti naudotojus ir jų teises. Į naudotojų valdymo funkcionalumą įeina šie veiksmai:

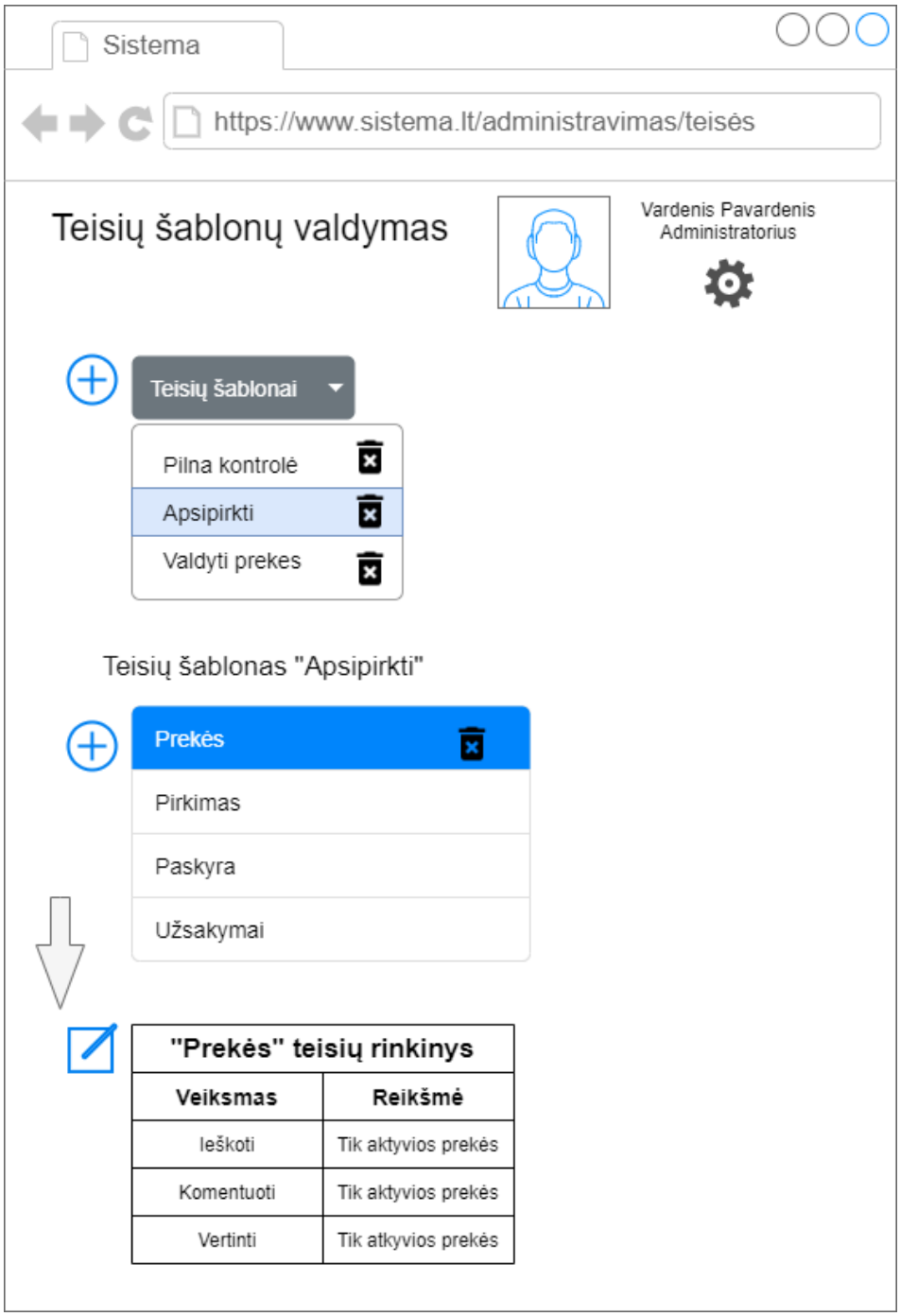
- Naudotojų sąrašo peržiūrėjimas
- Naujo naudotojo sukūrimas
- Teisių priskyrimas naudotojui
- Naudotojo ištrynimas

Paveiksle **pav. 2.14** pavaizduota pavyzdinis naudotojų valdymo tinklapis, kuriame administratoriaus galės atlikti išvardintas funkcijas. Naudotojai vaizduojami lentelės principu, o eilutės įrašą sudaro naudotojo vardas ir pavardė, jo rolė bei pritaikytas teisių šablono pavadinimas. Konkrečios rolės ir teisių šablonų pavadinimai priklauso nuo konkrečios sistemos atvejo ir šiame kontekste nėra svarbūs. Tačiau atkreiptinas dėmesys, jog naudotojo rolė ir jam priskirtos teisės nėra tiesiogiai susijusios – t.y. tas pačias roles turintys naudotojai gali turėti skirtingas teises ir atvirkščiai – tas pačias teises turintys naudotojai gali turėti skirtingas roles. Tokia leidimų struktūra yra lankstus, rolėmis grįstas prieigos valdymo metodas su papildomu plėtiniu – atskiromis teisėmis kiekvienai rolei. Šios teisės detalizuoja leidžiamus naudotojo veiksmus sistemoje ir išsprendžia problemą, kai tą pačią rolę turintys darbuotojai privalo galėti atlikti funkcijas skirtingais apribojimais. Tarkime, pavyzdyje nurodyta vadybininko rolė, kuri turi du teisių šablonus – valdyti visus arba tik savo pardavimus. Šioje situacijoje galima numanyti, kad priklausomai nuo vadybininko kasdieninių funkcijų, jis gali arba valdyti tik savo, arba visų vadybininkų užregistruotus pardavimų įrašus. Šiuo atveju abu vadybininkai privalo galėti atlikti tuos pačius veiksmus, tačiau vienas iš skirtumų, tarkime, gali būti tai, jog vienam vadybininkui mikropaslaugos ta pati funkcija gražins tik jo įrašus, o kitam – visus. Todėl ši struktūra leis mikropaslaugoms realizuoti lankstų prieigos valdymo metodą, o administratoriau beliks teisingai nustatyti teises naudotojams.



pav. 2.14 Naudotojų valdymo žiniatinklio puslapio pavyzdys

Iš pagrindinio naudotojų valdymo puslapio, pavaizduotame **pav. 2.14**, administratorius taip pat galės šią lentelę filtruoti pagal paieškos frazę, rolę ir (arba) teisių šabloną. Taip pat administratorius turės galimybę atidaryti kiekvieną naudotojo paskyros įrašą bei peržiūrėti jo detales. Be abejo, administratorius galės ir redaguoti tam tikrą naudotojo paskyros informaciją – parinkti teisių šabloną, rolę arba ištrinti (užblokuoti) paskyrą. Svarbu atkreipt dėmesį, kad „ištrynimo“ metu naudotojas nebus tiesiogiai ištrinamas, bet bus tiesiog užblokuojamas – paskyrai nebebus galima prisijungti prie sistemos. Ši technika vadinama minkštuoju ištrynimu (angl. *soft delete*) ir ji išsprendžia tokias problemas, kaip tiesioginės sąsajos tarp duomenų – ištrynus naudotoją greičiausiai reikėtų ištrinti ir visus įrašus, susijusius su juo, tarkime jo atliktus užsakymus, kurie yra istoriškai svarbi informacija sistemoje. Taip pat ši technika veikia kaip saugumo mechanizmas, kai įrašas buvo ištrintas per klaidą ir jį reikia atstatyti ištrintą įrašą, o šiuo atveju tai padaryti paprasta. Tačiau susidaro problema dėl perteklinių duomenų, todėl patartina šiuos minkštai ištrintus įrašus galutinai ištrinti po kiekvienos duomenų bazės atsarginės kopijos padarymo.



pav. 2.15 Teisių šablonų valdymo žiniatinklio puslapio pavyzdys

Naudotojų teisės – tai leidimų rinkinys, kuris apibrėžia sistemos funkcijas, kuriomis naudotojas gali naudotis . Į naudotojų teisių valdymą įeina:

- Teisių šablonų valdymas
- Teisių sąrašo peržiūrėjimas
- Naujų teisių sukūrimas
- Esamų teisių ištrynimas

Administratorius valdo naudotojų teisių šablonus. Pavyzdiniame tinklalapyje **pav. 2.15** pavaizduota, kad pirmąjį sąrašą sudaro teisių šablonų rinkinys. Šį rinkinį administratorius gali keisti – redaguoti,

pridėti naujus įrašus arba ištrinti esamus. Sekantis sąrašas apibūdina teisių rinkinius, kurie įeina į pasirinktą šabloną. Vienas teisių rinkinys yra skirtas vienai mikropaslaugai arba posistemei. Pavyzdžiui, jei el. prekybos sistemos modulius galima išskaidyti į prekių, pirkimų, paskyrų ir užsakymų mikropaslaugas, tuomet turėtų būti sukurtos ir atitinkami teisių rinkiniai. Teisių rinkiniai turi būti administruojami atskirai, o tai padarius, galima valdyti šablonus kurie remiasi šiais teisių rinkiniais. Kiekvienai posistemei rekomenduotina išskirti po atskirą teisių rinkinį lankstumo tikslais. Tokiu būdų teisės skirtinguose sistemos moduliuose (mikropaslaugose) nebus tarpusavyje priklausomi, todėl galės būti valdomi izoliuotai. Tai reiškia, jog pakeitus naudotojo teises prekių mikropaslaugai, kiti teisių rinkiniai liks nepakitę, todėl mikropaslaugos veiks kaip veikusios.

Kiekvienas teisių rinkinys yra sudarytas iš sąrašo veiksmų ir atitinkamų reikšmių, kurios apibūdina konkretų leidimą. Šis sąrašas yra konfigūruojamas kiekvienam šablonui atskirai. Svarbu paminėti, kad suteikiant privilegijas naudotojams vadovaujamasi mažiausių privilegijų principu, kai yra daroma prielaida kad naudotojai gali atlikti tik nurodytus veiksmus, o visų kitų veiksmų atveju daroma prielaida, kad naudotojas jų atlikti neturi teisės. Toks prieigos valdymo metodas iš dalies apsaugo nuo kontekstinių leidimų atlikti funkcijas arba netyčinio per didelių naudotojo leidimų sistemoje. Tačiau tik iš dalies todėl, nes teoriškai administratorius gali priskirti šablonui visas teises, ir šį šabloną priskirti visiems naudotojams. Todėl administratoriaus funkcija yra labai svarbi – jis turi suprasti valdomų naudotojų poreikius su atliekamomis funkcijomis ir išmanyti, kokių teisių jiems reikia atliekant kasdieninius veiksmus ir nepriskirti naudotojams daugiau teisių, negu jiems reikia.

Šablone esantis teisių rinkinys yra priskiriamas kiekvienam naudotojui. Kai naudotojas kreipiasi į resurso mikropaslaugą, autorizavimo mikropaslauga išduoda vidinį prieigos žetoną, kuriame yra šie teisių rinkiniai. Kiekviena resurso mikropaslauga gavusi užklausą privalo validuoti vidinį prieigos žetoną ir atlikti funkcijas sistemoje remiantis šiame žetone esančia informacija. Svarbu paminėti, kad mikropaslauga turi galimybę būti visiškai vieša (t.y. nereikalaujanti jokios autorizacijos) arba turėti tam tikras funkcijas, kurioms taip pat nereikia jokių prieigos teisių. Tik tokiu atveju mikropaslauga gali ignoruoti vidinį prieigos žetoną ir atlikti funkciją be jokio autorizavimo.

2.8. Prieigos valdymo metodo duomenų bazės specifikacija

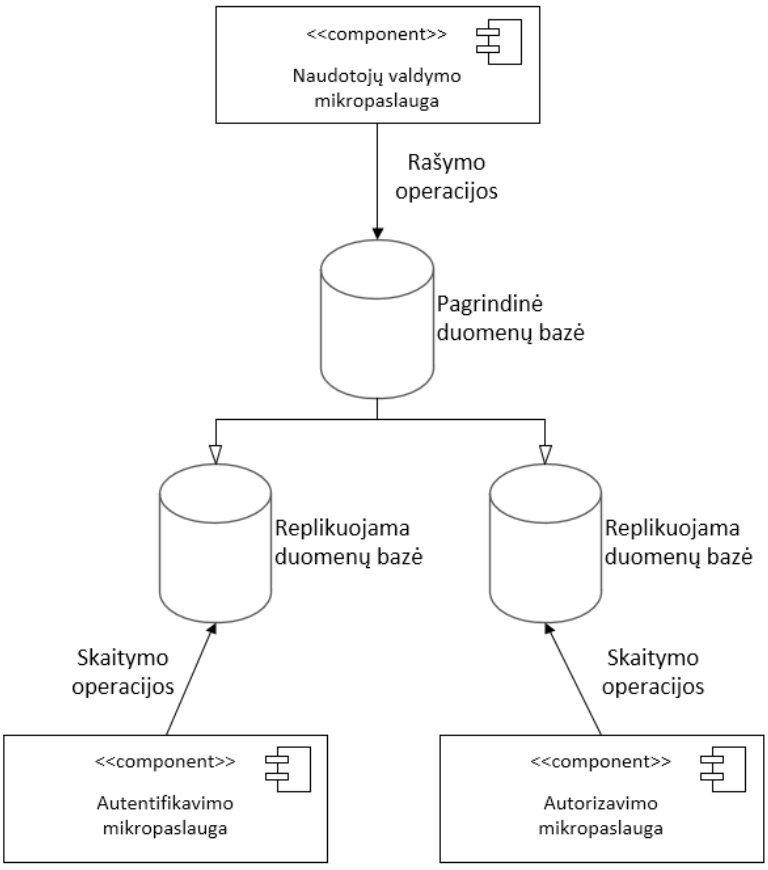
Siūlomas prieigos valdymo metodas yra atsakingas už naudotojų duomenų valdymą, kurie yra saugomi duomenų bazėje. Prototipo realizacijai pasirinktas Microsoft SQL duomenų bazės serveris. Jis yra bene lengviausiai suderinamas su kitomis prototipe naudojamomis Microsoft technologijomis. Todėl ši duomenų bazė buvo sklandžiai integruota į bendrą prieigos valdymo procesą. Microsoft SQL reliacinė duomenų bazė turi šiuos *ACID* transakcijos privalumus:

- Atomiškumas – sėkmingai įvyksta visi transakcijos veiksmai, arba nei vienas;
- Stabilumas – transakcija prasideda ir baigiasi įprastoje sistemos būsenoje;
- Izoliacija – transakcijos veiksmai yra izoliuoti nuo pradinės būsenos. Įvykus transakcijai, duomenų būsena pasikeičia iškart;
- Ilgalaikiškumas – įvykusių transakcijų rezultatai išlieka ir savaime negali pakisti.

Šios savybės užtikrina duomenų saugyklos operacijų patikimumą ir duomenų būsenos integralumą. Tačiau neigiamai atsiliepia greitaveikai, lyginant su nereliacinėmis duomenų bazėmis. Pavyzdžiui, nereliacinėse duomenų bazėse neretai atsisakoma duomenų pastovaus vientisumo, paskirstant apdorojimo srautus skirtingiems duomenų bazės mazgams. Tokia metodika prieigos valdymo

mechanizmui netiktų, nes nesant užtikrintam duomenų vientisumui, naudotojas gali įgyti teises, kurių nebeturėtų turėti ir realizuoti neautorizuotus veiksmus vien dėl duomenų bazės sinchronizavimo proceso netobulumo. Todėl renkantis duomenų bazę prototipui nuspręsta naudotojų duomenis patikėti Microsoft SQL duomenų bazei.

Microsoft SQL duomenų bazė turi esminį trūkumą – greitaveika ir plečiamumas. Nereliacinės duomenų bazės yra puikiai pritaikytos horizontaliam plėtimui – sistemoje esančių mazgų didinimui. Tačiau Microsoft SQL duomenų bazė nusileidžia šiuo aspektu, nes jos horizontalus plėtimas yra sunkesnis uždavinys, o vertikalus plėtimas tam tikrais atvejais tampa sąlyginai brangus arba net neįmanomas, kas liečia procesoriaus pajėgumų, disko dydžio ar operatyviosios atminties kiekio didinimą. Šiai problemai yra taikomas gana populiarus sprendimas – pirminės ir antrinės duomenų bazių (angl. *master-slave*) architektūra **pav. 2.16**. Šioje architektūroje pagrindinė duomenų bazė atliktų duomenų rašymo ir skaitymo operacijas, kurias nurodo naudotojų valdymo mikropaslauga. Replikuojamomis duomenų bazėmis naudotųsi tik autentifikavimo ir autorizavimo mikropaslaugos, kurioms pilnai pakanka tik skaitymo operacijų duomenų bazėje, kad sėkmingai išduotų ar valiuotų prieigos žetonus. Pritaikius šią duomenų bazės plėtimo sprendimą, reliacinė duomenų bazė iš esmės išpildo mikropaslaugų architektūros keliamus reikalavimus greitaveikai, plečiamumui, patikimumui ir prieinamumui.



pav. 2.16 Prieigos valdymo duomenų bazių plėtimo koncepcija

Prieigos valdymo metodo mikropaslaugų architektūroje projekto išvados

1. Suprojektuotas prieigos valdymo metodas, kuris klientui pateikia supaprastintą autentifikavimo žetoną, o vidinėms mikropaslaugoms – autorizavimo. Tai leidžia paslėpti tam tikras su autorizavimu susijusias detales kurios gali kisti nuo kliento, pritaikant išorinius ir vidinius prieigos žetonus.
2. Mikropaslaugų architektūros prieigos metodas naudoja žetonus, kurie neturi būsenos. Tai leidžia centralizuoti prieigos valdymą į autentifikavimo ir autorizavimo paslaugas, kurių išduotais žetonais pasitiki resursų mikropaslaugos. Taip užtikrinama, kad tarp mikropaslaugų leidimai naudotojams bus lengvai valdomi.
3. Prieigos valdymo strategija pagrįsta naudotojų teisių šablonais, kurie turi teisių rinkinius su leidimais konkretiems veiksmams atlikti. Toks minimalių privilegijų (atributų) modelis užtikrina didesnį sistemos saugumą bei didesnį prieigos valdymo lankstumą naudotojų atžvilgiu.
4. Prieigos valdymo metodui mikropaslaugų architektūroje reikia naudotojo sąsajos, kurioje būtų galima administruoti naudotojus ir jų teises. Jos pritaikymas leis prieigos valdymo sprendimu naudotis paprasčiau, todėl jis bus lengviau pritaikomas įvairiose sistemose.
5. Žetonų saugumui padidinti, autentifikavimo ir autorizavimo mikropaslaugos turėtų turėti galimybę vienu metu turėti galiojančius žetonus, pasirašytus skirtingais raktais, todėl realizuojant sistemą vertėtų atsižvelgti į raktų paskirstymo ir rotacijos strategijas.
6. Siūloma užklausų valdytojo servisui persiųsti autorizavimo ir autentifikavimo mikropaslaugoms tik reikalingą informaciją, o ne visą užklausą. Tai turėtų padidinti prieigos valdymo greitaveiką. Svarbu paminėti, kad suprojektuoto sprendimo mikropaslaugos yra tarpusavy izoliuotos, todėl gali būti plečiamos horizontaliai.

3. Prieigos valdymo metodo mikropaslaugų architektūroje prototipas

3.1. Įrankiai prieigos valdymo metodo prototipui

Prieigos valdymo metodo mikropaslaugų architektūroje realizacijai pasirinktas technologijų paketas, kuris yra orientuotas į Microsoft siūlomus įrankius. Microsoft paskutiniu metu koncentruojasi į nuo platformos nepriklausančių programų kūrimą bei debesų kompiuteriją. Šios dvi kryptys leido šiai kompanijai sukurti patogius įrankius, su kuriais galima efektyviai vystyti programų sistemas pasitelkiant debesų kompiuterijos infrastruktūrą.

Azure Service Fabric – tai platforma, kuri skirta efektyviam mikropaslaugomis grindžiamų aplikacijų kūrimui, diegimui ir priežiūrai debesų kompiuterijos infrastruktūroje. Tai nemokamas ir universalus įrankis, kuris gali būti naudojamas įvairiose aplinkose, pvz.: su Azure, AWS debesijos paslaugų tiekėjais arba įdiegtas asmeniniame kompiuteryje. Azure Service Fabric platformą sudaro 3 pagrindinės dalys:

– Infrastruktūra

Azure Service Fabric SDK suteikia galimybę įdiegti mazgus ir sukurti klasterius pasirinktoje įrangoje, kartu su jų priežiūra ir valdymu funkcionalumą. Vienas klasteris gali būti sudarytis iš vieno ar kelių mazgų, kurie turėtų būti tarpusavyje susiję įdiegtų aplikacijų atžvilgiu ir sudaryti paskirstytą programų sistemą. Ši platforma palaiko dviejų tipų mikropaslaugas – su būsena (tarpusavy susiję mazgai) ir be būsenos (tarpusavy nenusiję mazgai). Taip pat infrastruktūroje pateikiamos papildomos paslaugos, įskaitant DNS, žurnalo registravimo, srauto paskirstymo, įgaliotojo serverio (angl. *proxy*) ir t.t.

– Valdymo įrankis

Azure Service Fabric Explorer yra įrankis, kuris leidžia patogiai valdyti programinės įrangos kūrėjo sukurtus klasterius, mazgus ir juose įdiegtas mikropaslaugas. Į valdymą įeina aplikacijų diegimas, būsenos peržiūra, veiksmų žurnalo stebėjimas ir t.t. Tai patogi sąsaja, kuri leidžia suprasti, kaip veikia klasteryje įdiegta paskirstyta programų sistema.

– Programavimo bibliotekos

Prie Azure Service Fabric SDK architektūros kartu pateikiami ir programavimo kalbų bibliotekų rinkiniai. Šios bibliotekos programuotojams palengvina mikropaslaugų sąsają su *Service Fabric* infrastruktūra. Pavyzdžiui, *.NET* platformoje yra pateikiamos bibliotekos, kurios leidžia mikropaslaugoms paleisti žiniatinklio programas šioje infrastruktūroje, kurti įrašus žurnalo paslaugoje, gauti informaciją apie savo mazgą, naudotis DNS paslauga ir kreiptis į kitas mikropaslaugas.

Azure Service Fabric platforma yra puikiai pritaikyta C# programavimo kalbai. Tai bene pagrindinė programavimo technologija, kuri sklandžiai integruojasi į šią debesijos platformą. *.NET* yra Microsoft vystoma atvirojo kodo ekosistema, kuri apima bibliotekų rinkinius, vykdymo aplinkas, kompiliatorius, įvairias kalbas ir t.t. O programas galima diegti ir vykdyti ne tik Windows, bet ir Linux sistemose. Šiam metodui pasirinkta *.NET* 5 versija, kuri išleista 2020 metais. Ji sujungia tradicinį .NET karkasą su nuo platformos nepriklausančiu *.Net Core* karkasu. Taip pat *Visual Studio* pateikia šabloninius projektus C# programavimo kalbai, kuriuose yra realizuota bazinė integracija su *Azure Service Fabric* platforma.

3.2. Prieigos valdymo metodo prototipo API funkcijų specifikacija

Prieigos valdymo metodo prototipas mikropaslaugų architektūroje turi 7 mikropaslaugas, kurios veikia bendroje visumoje ir sudaro prieigos valdymo mechanizmą, kuriuo gali naudotis kiti programinės įrangos kūrėjai. Mikropaslaugos turi laikytis REST API šabloninių principų nuoseklumui ir aiškumui palaikyti, kaip vaizduoja **lentelė 3.1**. Į šį sąrašą neįtraukta naudotojų valdymo mikropaslauga. Tai reiškia, jog prototipas neturės tiesioginės grafinės sąsajos, o naudotojai bus valdomi automatizuotų scenarijų pagalba.

Užklausų valdytojo iš funkcinės pusės mikropaslauga yra atsakinga už užklausų peradresavimą visoms kitoms mikropaslaugoms. Papildomai, užklausų valdytojas gali atlikti ir tokius veiksmus kaip žurnalo vedimas, užklausų paskirstymas, statistikos vedimas ir pan. Iš naudotojo pusės ši mikropaslauga yra prieigos prie sistemos taškas, todėl jai pakanka vieno URL adreso.

Autentifikavimo mikropaslauga atlieka dvi funkcijas – prijungia naudotoją prie sistemos išduodant jam išorinį prieigos ir atnaujinimo žetonus. Šiuos žetonus mikropaslauga suteikia tik naudotojui pateikus teisingą prisijungimo vardą ir slaptažodį. Prototipo realizacijoje daroma prielaida, kad naudotojo prisijungimo vardas yra jo el. paštas, yra unikalus. Taip pat autentifikavimo mikropaslauga gali atnaujinti prieigos žetoną, naudotojui pateikus atnaujinimo žetoną. Taigi, kai autentifikavimo mikropaslauga gauna galiojantį atnaujinimo žetoną, ji išduoda naują prieigos ir atnaujinimo žetonus, kurie leis naudotojui toliau kreiptis į resurso mikropaslaugas ir sėkmingai naudotis programų sistema.

lentelė 3.1 Mikropaslaugų API funkcijų specifikacija

Mikropaslauga	**Nr.**	**URL Adresas**	**HTTP Metodas**	**Įvestis**	**Funkcija**
Užklausų valdytojo	1	/RequestManager	Bet kuris	Kliento užklausa	Užklausos vidinio žetono gavimas, auditavimas, persiuntimas
Autentifikavimo	2	/ Authentication /Login	POST	Kliento prisijungimo vardas ir slaptažodis	Išorinio prieigos žetono išdavimas
	3	/ Authentication /Refresh	POST	Atnaujinimo žetonas	Išorinio prieigos žetono atnaujinimas
Autorizavimo	4	/Authorization	POST	Išorinis prieigos žetonas	Vidinio prieigos žetono išdavimas
Prekių	5	/	GET	Nėra	Gauti prekes
	6	/{id}	GET	Prekės identifikatorius	Gauti prekę
	7	/	POST	Nauja prekė	Sukurti prekę
	8	/{id}	PUT	Esamos prekės informacija	Atnaujinti prekę, kreipiasi į Nr. 12
	9	/{id}	DELETE	Prekės identifikatorius	Ištrinti prekę, kreipiasi į Nr. 12
Krepšelių	10	/	POST	Naujas krepšelis	Sukurti krepšelį, kreipiasi į Nr. 5
	11	/{id}	GET	Krepšelio identifikatorius	Gauti krepšelį, kreipiasi į Nr. 6
	12.	/{id}/products	POST	Krepšelio identifikatorius, krepšelio prekė	Pridėti prekę į krepšelį

	13	/{id}	DELETE	Krepšelio identifikatorius	Ištrinti krepšelį
	14	/{id}/products/{id}	DELETE	Krepšelio ir prekės identifikatorius	Pašalinti prekę iš krepšelio
Siuntų	15	/	GET	Nėra	Gauti siuntas
	16	/{id}	GET	Siuntos identifikatorius	Gauti siuntą
	17	/	POST	Nauja siunta	Sukurti siuntą, kreipiasi į Nr. 5
	18	/{id}	PUT	Esamos siuntos informacija	Atnaujinti siuntą, kreipiasi į Nr. 6
	19	/{id}	DELETE	Siuntos identifikatorius	Ištrinti siuntą, kreipiasi į Nr. 5
Užsakymų	20	/	GET	Nėra	Gauti užsakymą
	21	/{id}	GET	Užsakymo identifikatorius	Gauti užsakymus
	22	/{id}	DELETE	Užsakymo identifikatorius	Ištrinti užsakymą.
	23	/{id}/shipment	PUT	Užsakymo identifikatorius ir siuntos duomenys	Pateikti siuntos duomenis, kreipiasi į Nr. 18
	24	/{id}/payment	PUT	Užsakymo identifikatorius ir mokėjimo duomenys	Pateikti mokėjimo duomenis, kreipiasi į Nr. 18
	25	/{id}	PUT	Užsakymo identifikatorius	Pateikti užsakymą, kreipiasi į Nr. 5, 18

Prekių, krepšelių, siuntų ir užsakymų mikropaslaugos skirtos imituoti verslo posistemę. Šios mikropaslaugos komunikuoja tarpusavyje ir taip generuoja vidinį srautą, kuris atspindi realios sistemos darbą. Kiekviena resurso funkcija yra apsaugota papildomu autorizavimu, kuris patikrina vidinį prieigos žetoną, kuris siunčiamas HTTP užklausos antraštėje „*MicroAC-JWT*“. Realias resursų mikropaslaugas, jų URL adresus, jų funkcijas bei reikiamus leidimus apibrėžia sistemos kūrėjai, kurie integruoja siūlomą prieigos valdymo metodą į savo programų sistemą. Kūrėjams reikės papildomai sukonfigūruoti užklausų valdytoją, kad šis priimtų užklausas į įvairias resursų mikropaslaugas. Taip pat pačias mikropaslaugas reikės realizuoti taip, kad jos priimtų vidinį prieigos žetoną ir jį validuotų su tuo pačiu raktu, kuris buvo naudojamas išdavimo procedūroje. Realizavus šiuos integracijos aspektus, programinės įrangos kūrėjai galės pilnai naudotis siūlomu mikropaslaugomis grįstu prieigos valdymo metodu.

Užklausų valdytojo, autentifikavimo, autorizavimo ir resursų mikropaslaugos atlieka papildomą užklausos žymėjimą. Šį žymėjimą galima įjungti ir išjungti, nurodant atitinkamą parametrą mikropaslaugos konfigūracijoje. Žymėjimas atliekamas užklausos apdorojimo pradžioje ir pabaigoje. Tai reiškia, jog kiekviena mikropaslauga kiekvieną užklausą pažymės mažiausiai du kartus – iškart ją gavus ir prieš pat išsiunčiant atsakymą. Žymėjimo procesas – tai metaduomenų pridėjimas į *HTTP* užklausos „*MicroAC-Timestamp*“ antraštę. Pridedami duomenys – data, laikas, mikropaslaugos pavadinimas, užklausos būsena, atliekamas veiksmas. Ši informacija gali būti naudojama kaip pėdsakai, skirti suprasti, kokie konkretūs veiksmai buvo atliekami užklausos apdorojimo metu.

3.3. Prieigos valdymo metodo prototipe naudojami API žetonai

API žetonų struktūra turėtų atitikti keliamus reikalavimus, kuriuos apibendrina **lentelė 2.2**, tačiau prototipo atžvilgiu tai nėra būtina. Prieigos valdymo metode naudojami trys žetonų tipai – išorinis prieigos (**pav. 3.1**), išorinis atnaujinimo (**pav. 3.2**) bei vidinis prieigos (**pav. 3.1**). Išorinis prieigos žetonas yra išduodamas pateikus teisingus naudotojo prisijungimo duomenis arba galiojantį atnaujinimo žetoną. Antruoju atveju kartu išduodamas ir naujas išorinis atnaujinimo žetonas. Vidinis prieigos žetonas išduodamas tik pateikus galiojantį išorinį prieigos žetoną. Likusioje programų sistemoje, resurso mikropaslaugos atlieka veiksmus tik tuo atveju, kai pateiktas galiojantis vidinis prieigos žetonas ir jame nurodyti subjekto leidimai atitinka leidimus, kurie reikalingi atlikti kviečiamai funkcijai.


```
{
  "jti": "3a6e7ee1-7c07-41ca-8204-c3e54716ccaa",
  "iss": "MicroAC:AuthenticationService",
  "aud": "MicroAC:AuthorizationService",
  "sub": "MicroAC:User",
  "uid": "29ebe694-f193-499a-95d0-63414342718d",
  "sroles": [
    "Client",
    "Free User"
  ],
  "nbf": 1639311558,
  "exp": 1640175558,
  "iat": 1639311558
}
```


pav. 3.1 Išorinio prieigos žetono pavyzdys

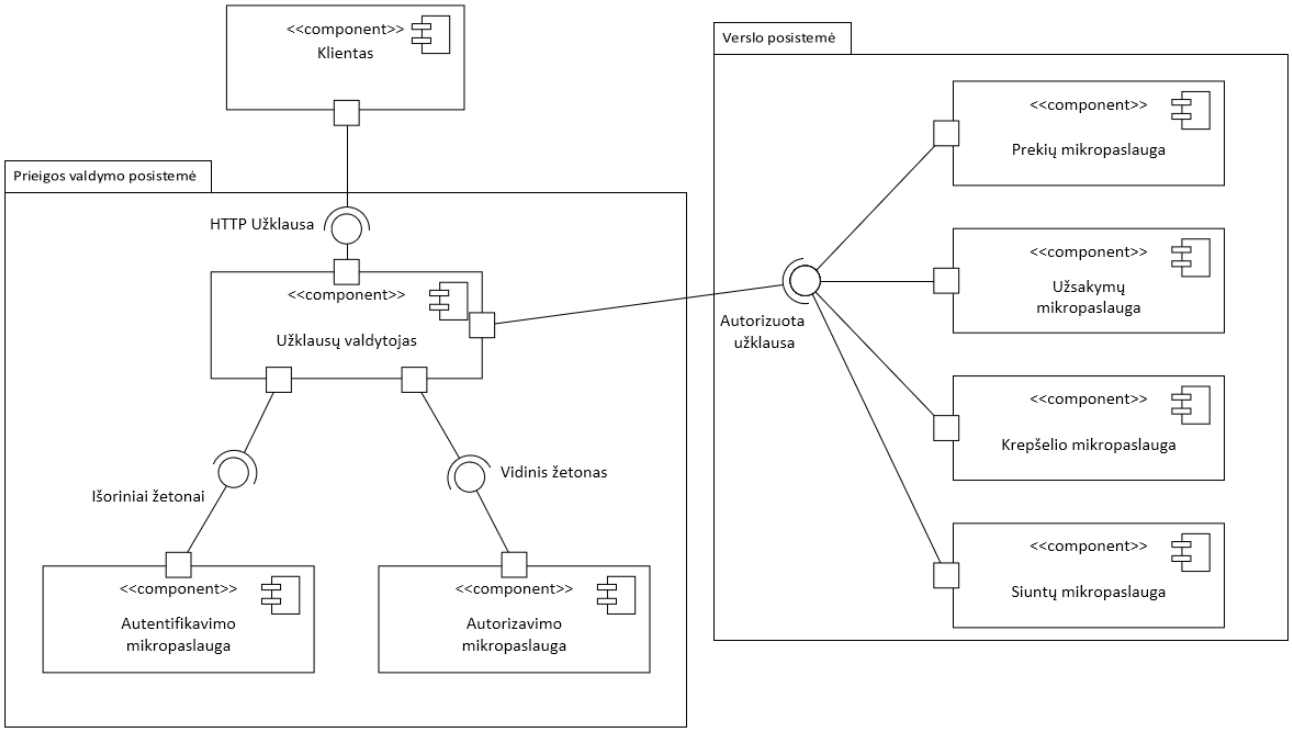

```
{
  "jti": "ea70bd10-eab4-4b83-aac8-953ddf6990b9",
  "iss": "MicroAC:AuthenticationService",
  "aud": "MicroAC:AuthenticationService",
  "sub": "MicroAC:User",
  "uid": "bfec3586-16d5-4803-82a0-c331eb405c5d",
  "nbf": 1637100968,
  "exp": 1637964968,
  "iat": 1637100968
}
```


pav. 3.2 Išorinio atnaujinimo žetono pavyzdys

pav. 3.3 Vidinio prieigos žetono pavyzdys

3.4. Prieigos valdymo metodo prototipo mikropaslaugų programinio kodo specifikacija

Siūlomame prieigos valdymo metodas reikalauja trijų pagrindinių mikropaslaugų – užklausų valdytojo, autentifikavimo ir autorizavimo. Klientas kreipiasi tik vieną iš jų – užklausų valdytoją. Jeigu klientas nori autentifikuotis, jis kreipiasi į užklausų valdytoją, kuris užklausą persiunčia į autentifikavimo mikropaslaugą. Ji išduoda reikiamus išorinius žetonus ir juos grąžina užklausų valdytojui, o užklausų valdytojas – grąžina juos klientui. Užklausų valdytojas persiunčia užklausas ir į kitus prašomus resursus, tačiau jeigu jie reikalauja autorizacijos – papildomai siunčiama užklausa į autorizavimo mikropaslaugą vidiniam prieigos žetonui gauti. Užklausa, kurią norime autorizuoti, privalo būti autentifikuota – t.y. antraštėje turėti galiojantį išorinį prieigos žetoną. Gavės atsakymą, užklausų valdytojas toliau persiunčia užklausą su papildomu autorizavimo žetonu toliau, į verslo posistemę. Resursų mikropaslaugos savarankiškai validuoja vidinius prieigos žetonus ir autonomiškai sprendžia, ar autorizuoti kliento veiksmą.

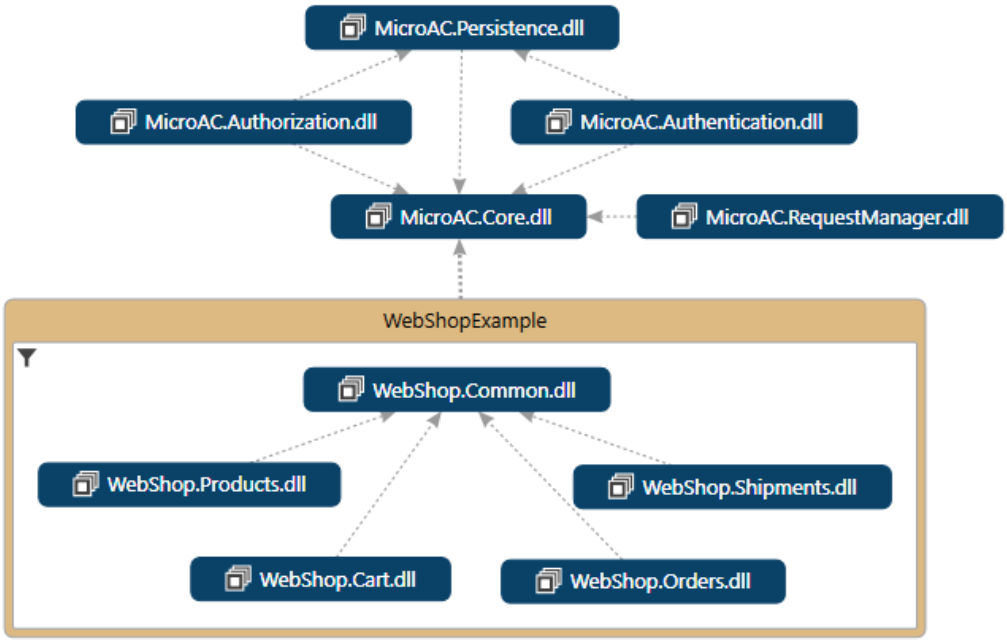


pav. 3.4 Prieigos valdymo metodo komponentų diagrama

Tokia prieigos valdymo posistemės atskirtis nuo likusios verslo posistemės ir kliento, vaizduojama **pav. 3.4**, turi esminį bruožą – naudojimo patogumas. Patogumas atsiskleidžia tuo, jog nei klientas, nei resurso paslaugos nežino, koks yra pilnas prieigos valdymo ir žetonų išdavimo procesas. Klientas tik žino apie išorinį prieigos žetoną, kuriuo rūpinasi pats, tačiau jam neaktualus vidinis prieigos žetonas. Tuo tarpu resursų mikropaslaugos autorizacijai naudoja tik vidinį prieigos žetoną. Vadinasi tiek klientai, tiek resursų mikropaslaugos išvengia sudėtingų žetonų valdymo schemų, kurios neretai būna būdingos didelio mąsto paskirstytose programų sistemose. Kadangi autorizavimo, autentifikavimo ir užklausų valdytojo mikropaslaugos atlieka esminius informacijų mainus, svarbiausias aspektas integracijai į realią verslo sistemą belieka konfigūracija. Užklausų valdytojui privaloma nurodyti pasiekiamų resursų paslaugų sąrašą, įskaitant ir požymius, ar paslaugai reikalinga autentifikacija ir/ar autorizacija. Sekantis konfigūracijos žingsnis – žetonų pasirašymo raktai. Jie reikalingi autorizavimo, autentifikavimo ir visoms resursų mikropaslaugoms, kurioms reikalinga autorizacija. Žetonai išduodami autorizavimo ar autorizavimo mikropaslaugose privalo būti ir validuojami tuo pačiu kriptografiniu simetriniu raktu, su kuriuo buvo išduoti.

Siūlomo prieigos metodo prototipe realizuotų programinių bibliotekų artefaktų ryšiai vaizduojami diagramoje **pav. 3.5**. Diagramoje esantys artefaktai atspindi diegimo failų visumą, kuri reikalinga norint šią programų sistemą įdiegti į klasterį ir jo mazgus. *MicroAC* vardų srityje yra tik tos mikropaslaugos, kurios yra esminės prieigos valdymo. *WebShop* vardų srityje yra klasės, kurios simuliuoja verslo posistemės darbo veiklą. Jų Komunikacijai tarpusavyje buvo sukurta *WebShop.Common* biblioteka, kuri realizuoja komunikaciją tarp resurso mikropaslaugų. Autorizavimo ir autentifikavimo mikropaslaugos kreipiasi į duomenų *Persistence* biblioteką, kuri turi bendrauja su duomenų baze. Visos mikropaslaugos priklauso nuo bendro branduolio *Core* artefakto, kuris savyje talpina sąsajas, klases ir modelius, kurie yra bendrai naudojami tarp skirtingų mikropaslaugų. Šia biblioteka gali naudotis ir resursų *WebShop* mikropaslaugos. Vystant realią aplikaciją, nekiltų problemų branduolio biblioteką supakuoti ir publikuoti viešuose arba privačiuose paketų šaltiniuose, pvz.: *NuGet Package Feed*. Kalbant apie konkrečias klases, branduolio biblioteka

turi: žetonų išdavimo ir validavimo klases; duomenų perdavimo modelius; žymų registravimo metodus ir kt. Sprendimas kelti bendras sąsajas ir klases į atskirą programinį paketą leidžia kurti mažiau kompoziciniais ryšiais apipintą kodą, o tai leis ateityje kodą lengviau plėsti, keisti ir testuoti.



pav. 3.5 Prieigos valdymo prototipo artefaktai ir jų tarpusavio ryšiai

Prototipo apimtis pagal kodo eilučių kiekį pateikta **pav. 1.1**. Prototipui realizuoti reikėjo 8 tūkst. kodo eilučių, iš kurių pagrindiniai prototipo projektai parašyti *C#* kalba turi 6 tūkst. eilučių. *F#* kalba daugiausiai naudojama greitaveikos testuose, o *Cucumber* – integracinių testų specifikacijai. Tuo tarpu *Typescript* kalba naudojama pavyzdinio tinklalapio sukūrimui.

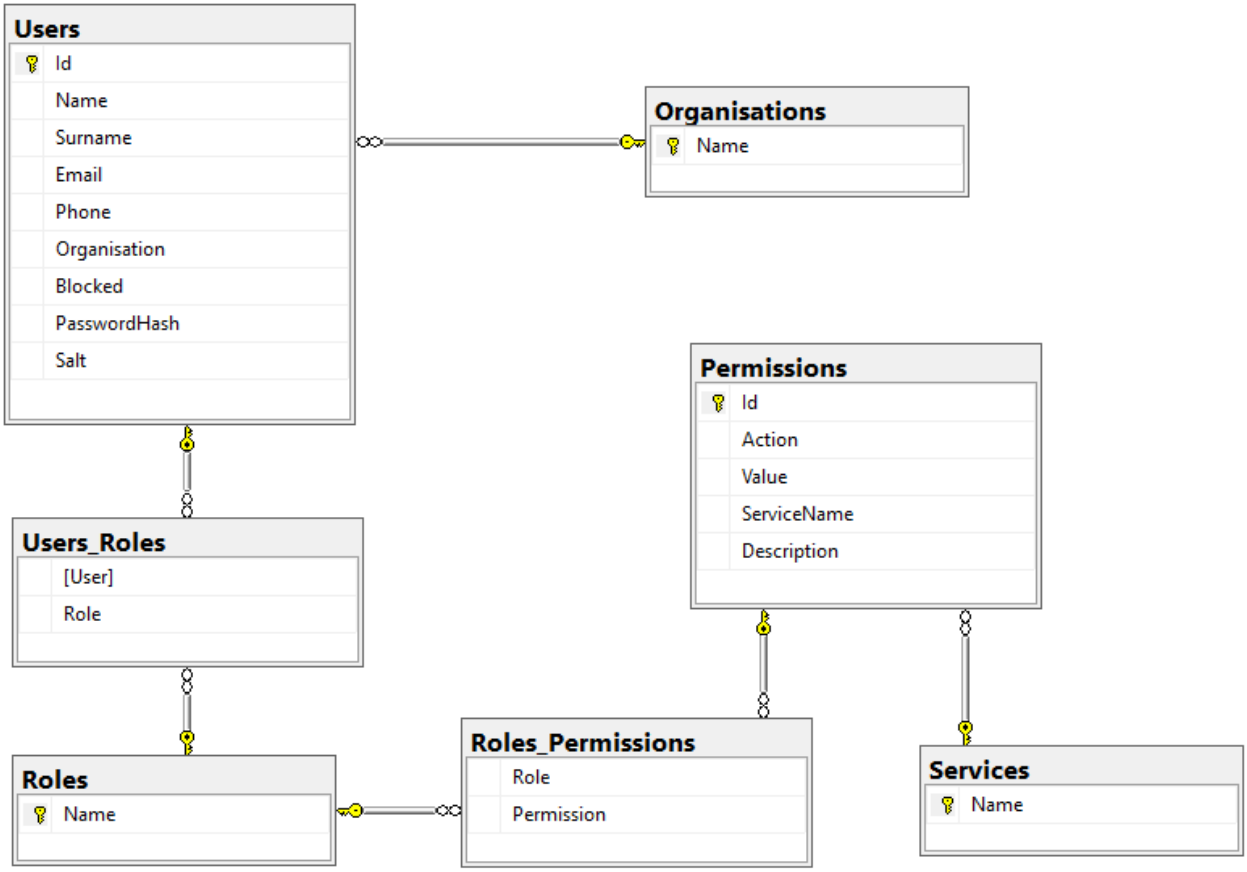
Language	files	blank	comment	code
C#	102	1010	400	6309
F#	12	172	11	838
TypeScript	15	64	11	457
SQL	11	15	30	203
PowerShell	1	67	51	178
Cucumber	5	24	0	149
SUM:	146	1352	503	8134

pav. 3.6 Prototipo apimtis pagal kodo eilučių kiekį

3.4.1. **Prieigos valdymo metodo prototipo duomenų bazės schemos specifikacija**

Diagramoje **pav. 3.7** vaizduojama prieigos valdymo metodo duomenų schemos struktūra, įskaitant lenteles, stulpelius bei lentelių tarpusavio ryšius. Svarbu atkreipti dėmesį, jog ši schema yra tik prototipas, kuris atspindi koncepciją. Šioje schemoje specialiai trūksta papildomų laukų, lentelių, identifikatorių. Toks sprendimas pasirinktas paprastumo sumetimais – prototipui papildomos vertės

nepridės daug įvairių laukų ir lentelių, kurie savyje turės sąlyginai daugiau naudingos informacijos. Toks duomenų schemos apkrovimas gali nukreipti dėmesį nuo esmės – tarpusavio sąsajų tarp leidimų, rolių ir naudotojų. Taip pat paprastumo kai kurios lentelės vietoj tikrojo unikalaus identifikatoriaus turi leidimą. Tai palengvina prototipo demonstraciją.



pav. 3.7 Prieigos valdymo metodo duomenų bazės schema

Taigi, duomenų bazėje turi būti lentelės, kurios saugo šią informaciją:

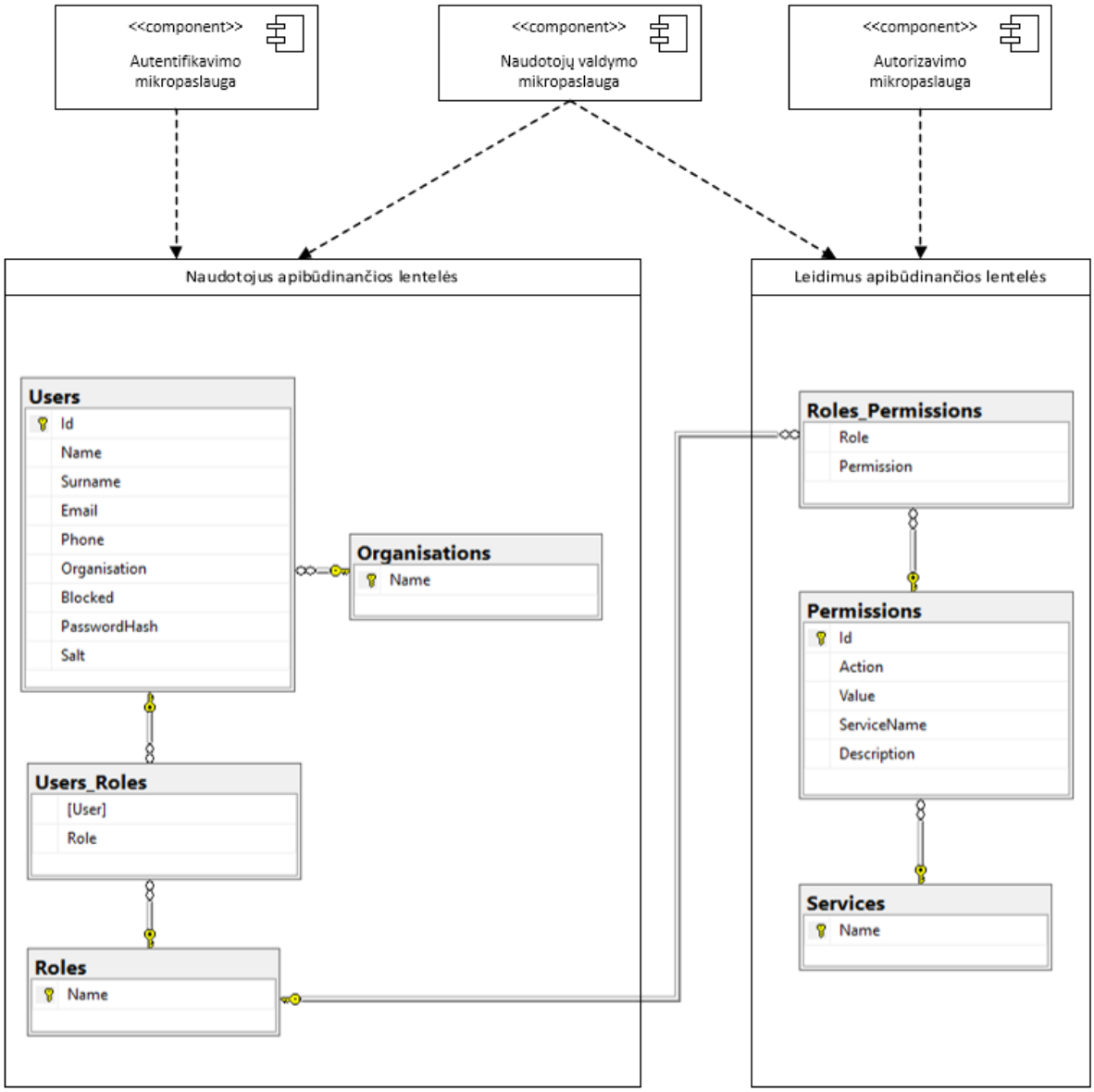
1. Naudotojai (*Users*) – naudotojų esybę apibūdinantys duomenys, įskaitant naudotojo identifikatorių ir prisijungimo laukelius.
2. Organizacijos (*Organisations*) – pavyzdinė lentelė, kuri papildo naudotojų informaciją organizacijomis. Keli naudotojai gali priklausyti tai pačiai organizacijai.
3. Naudotojų Rolės (*Users_Roles*) – tarpinė lentelė, apibūdinanti „daug su daug“ ryšį tarp naudotojų ir rolių. Tai reiškia, jog naudotojai gali ne tik turėti keletą rolių, bet ir tarpusavy jas dalintis.
4. Rolės (*Roles*) – galimų naudotojų rolių sąrašas. Šioje lentelėje specifikuojama konkreti informacija, kuri apibūdina rolę. Rolė – tai leidimų rinkinys, leidžiantis sistemos naudotojui atlikti veiksmus
5. Rolių Leidimai (*Roles_Permissions*) – tarpinė lentelė, apibūdinanti „daug su daug“ ryšį tarp rolių ir leidimų. Tai reiškia, viena rolė gali ne tik turėti keletą leidimų, bet ir keli skirtingi leidimai gali turėti tą pačią rolę.
6. Leidimai (*Permissions*) – lentelė, kaupianti duomenis, kurie susiję su konkrečiais leidimais. Leidimas – tai teisė atlikti veiksmus sistemoje. Kiekvieną leidimą apibūdina veiksmas (pvz.: sukurti, peržiūrėti), paslauga (pvz.: prekių, krepšelio, užsakymų) ir reikšmė (pvz.: leisti, drausti,

tik savo). Pavyzdžiui, viena įrašo eilutė gali suformuluoti tokį leidimą: „peržiūrėti tik savo krepšelį“.

7. Paslaugos (*Services*) – sąrašas, kuris nurodo, kokios paslaugos yra teikiamos sistemoje. Kiekvienas leidimas yra skirtas konkrečiai paslaugai. Šiuo atveju paslaugos yra suprantamos kaip mikropaslaugų visuma, kurios sudaro paskirstytą programų sistemą debesų kompiuterijoje, kurios prieiga yra valdoma siūlomu metodu.

3.4.2. **Mikropaslaugų sąsaja su duomenų bazės lentelėmis**

Duomenys, kuriuos naudoja siūlomo prieigos metodo mikropaslaugos, gali būti suskirstyti į dvi grupes – apibūdinantys naudotojus arba leidimus. Esminė informacija, apibūdinanti naudotojus, yra paskyros laukeliai ir naudotojo rolės. Ši informacija yra esminė autentifikavimo mikropaslaugai, nes išoriniuose žetonuose yra naudojama būtent ši informacija, kaip pavaizduota **pav. 3.1** ir **pav. 3.2**. Taigi, autentifikavimo užklausos metu ši mikropaslauga kreipiasi į duomenų bazę ir skaito informaciją iš lentelių, kurios apibūdina naudotojus, kaip vaizduojama **pav. 3.8**.



pav. 3.8 Mikropaslaugų priklausomybė nuo duomenų bazės lentelių

Lygiai tokia pati situacija ir su autorizavimo mikropaslauga. Užklausos metu, užklausų valdytojas kreipiasi į autorizavimo mikropaslaugą, o ši išduoda vidinį prieigos žetoną, pavaizduotą **pav. 3.3**. Žetono išdavimui reikalinga su leidimais susijusi informacija, įskaitant naudotojų-rolių, leidimų ir paslaugų lenteles, kaip vaizduojama **pav. 3.8.** Autorizavimo mikropaslauga, gaudama šią informaciją gali sėkmingai išduoti vidinį prieigos žetoną tolesniam užklausos autorizavimui. Svarbu atkreipti dėmesį, kad prototipe autorizavimo mikropaslauga išduoda žetoną remiantis tik rolių sąrašu, kuris nurodytas autentifikavimo žetone. Jeigu būtų poreikis autentifikavimo žetone įrašyti tam tikrus viešus leidimus, ši galimybė yra. Be abejo, būtų pažeista duomenų bazės lentelių izoliavimas tarp grupių, tačiau tai neturi didelės įtakos, jeigu naudojamas siūlomas duomenų bazių replikavimo sprendimas **pav. 2.16**.

Naudotojų valdymo mikropaslaugos funkcijoms atlikti yra būtina rašymo prieiga prie visų duomenų bazės lentelių. Naudotojų valdymo mikropaslaugos esminė funkcija yra valdyti ir keisti duomenis, todėl ši mikropaslauga privalo naudoti pagrindinę duomenų bazę, jeigu realizuojama duomenų bazių replikacija. Todėl **pav. 3.8** vaizduojama situacija, kai naudotojų valdymo mikropaslauga naudoja tas pačias lenteles, kaip ir autentifikavimo ir autorizavimo mikropaslaugos. Tačiau esminis skirtumas – naudotojų valdymo mikropaslauga atlieka ne tik skaitymo, bet ir rašymo operacijas, tuo tarpu autorizavimo ir autentifikavimo mikropaslaugos atlieka tik skaitymo operacijas duomenų bazėje.

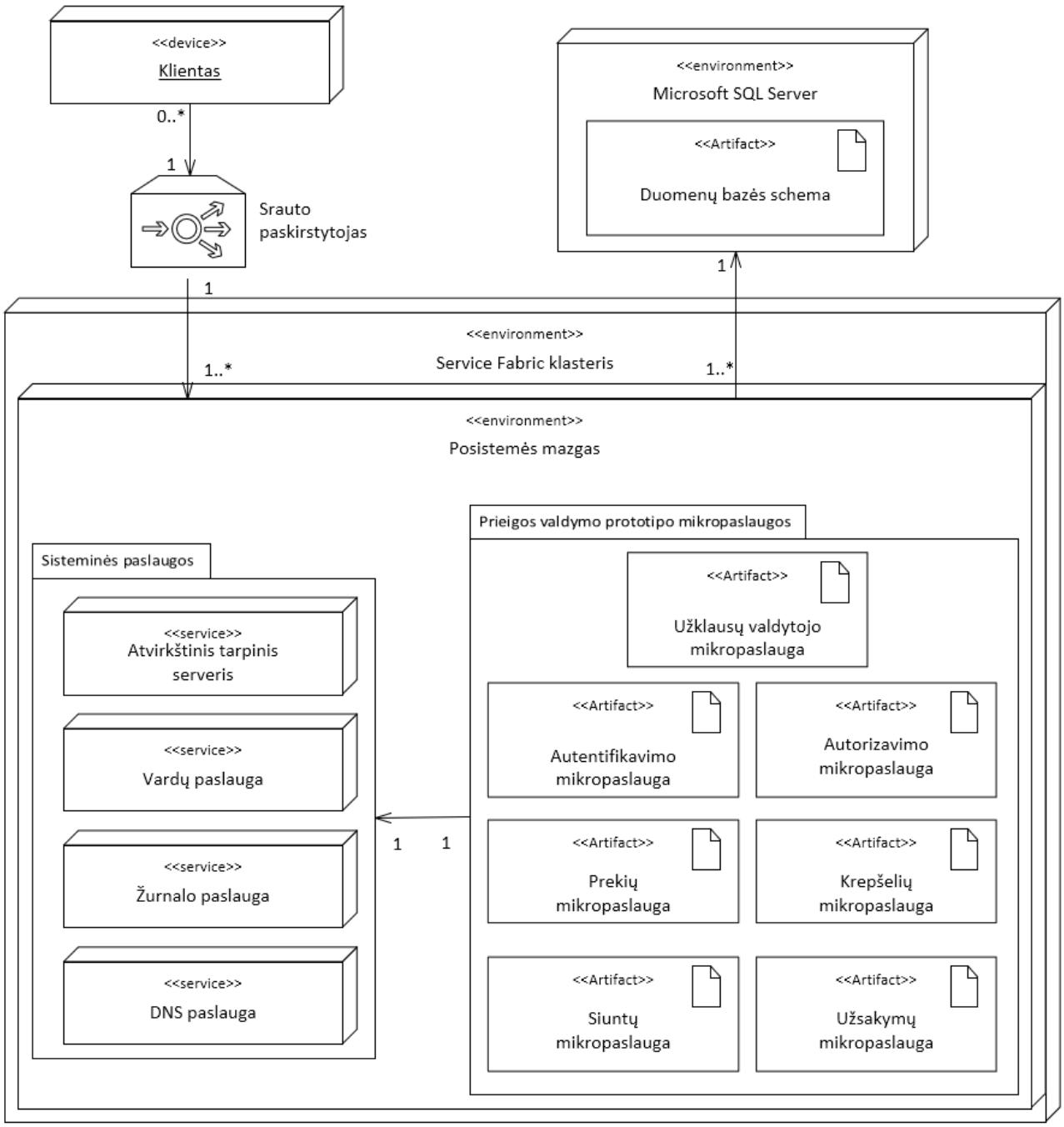
Apibendrinant, kuriant prieigos valdymo mikropaslaugas buvo atsižvelgta į duomenų bazių struktūrą ir jos tikslinį panaudojimą. Prieigos metodo prototipas realizuotas taip, jog autorizavimo ir autentifikavimo mikropaslaugos atliktų tik duomenų skaitymo operacijas duomenų bazėje. Todėl prototipas pilnai atitinka projekte detalizuota duomenų bazių replikavimo modelį. Tai reiškia, kad nors prototipe duomenų bazių replikavimas nėra realizuotas, tačiau jam yra realizuoti visi reikalavimai, kad jį būtų galima pritaikyti. Jeigu būtų realizuotas siūlomas duomenų bazių replikavimo mechanizmas, apibrėžiamas **pav. 2.16**. – prototipo greitaveikos tyrimo rezultatai turi galimybę pagerėti, esant dideliems užklausų srautams.

3.5. Prieigos valdymo metodo prototipo mikropaslaugų architektūroje diegimo diagrama

Prieigos valdymo prototipas mikropaslaugų architektūroje diegiamas į *Service Fabric* klasterį, kaip vaizduojama **pav. 3.9**. Klasteris iš ne mažiau nei vieno mazgo, tačiau jų kiekis nėra ribojimas. Jis priklauso nuo plečiamumo ir greitaveikos reikalavimų. Klasteryje egzistuoja dvejų tipų mikropaslaugos – sisteminės ir prototipo. Vaizduojamos prototipo mikropaslaugos skirtos pademonstruoti prieigos valdymo metodo mechanizmą realizacijoje, tuo tarpu pagrindinės sisteminės paslaugos yra DNS, žurnalo, vardų ir HTTP vartų. Klasteryje yra naudojamas specifinis paslaugų vardų valdymo mechanizmas – apart to, kad kiekviena mikropaslauga turi savo IP adresą ir DNS vardą, jai papildomai yra suteikiamas *Service Fabric* pavadinimas (pasiekiamas vardų paslaugoje). Todėl žurnalo ir DNS paslaugų tarpusavio sąveikia leidžia naudotis atvirkštiniu tarpiniu serveriu, kuris leidžia per vieną unifikuotą tašką pasiekti bet kurią mikropaslaugą. Šiuo atveju mikropaslaugoms esančioms tame pačiame klasteryje nereikia saugoti papildomos informacijos, kurios reikia pasiekti kitas paslaugas – pakanka žinoti tik paslaugos pavadinimą, kas labai palengvina bendrą komunikacijos valdymą tarp mikropaslaugų.

Taip pat diegimo diagramoje, pavaizduotoje **pav. 3.9**, nurodoma, kad duomenų bazė yra diegiama į atskirą aplinką, nesusijusią su klasteriu. Prototipo atveju, reikšmės neturi kur konkrečiai bus diegiamas duomenų bazės serveris. Tačiau greitaveikos atžvilgiu, geriausia būtų, jeigu tinklo schemoje šie komponentai nebūtų nutolę vienas nuo kito. Kalbant apie klientą, tarp jo ir

mikropaslaugų turi būti realizuotas srauto paskirstytojas. Tačiau nėra svarbu, ar srauto paskirstytojas yra lokalizuotas klasterio viduje ar išorėje. Srauto paskirstytojo vaidmenį teoriškai gali atlikti ir mazge esantis atvirkštinis tarpinis serveris. Ši sisteminė klasterio paslauga yra ypatinga, nes ji turi sąsajas ir su kituose mazguose esančiomis mikropaslaugomis. Tai reiškia, kad viename mazge esantis atvirkštinis tarpinis serveris gali pasiekti mikropaslaugas, esančias kituose mazguose. Tai sąlygoja sisteminės mikropaslaugos, kurios realizuotos su būsena. Tarp to pačio tipo paslaugų, šia būsena yra dalinamasi skirtinguose mazguose. Svarbu atkreipti dėmesį, kad prototipe realizuotos mikropaslaugos neturi minėtos būsenos, dėl to jos nėra priklausomos nuo *Service Fabric* klasterio infrastruktūros ir visos mikropaslaugų replikos yra vienodos.



pav. 3.9 Prieigos valdymo metodo diegimo diagrama mikropaslaugų architektūroje

Prieigos valdymo prototipo mikropaslaugų architektūroje išvados

1. Sukurtas prieigos valdymo metodo prototipas turi tris esmines mikropaslaugas – užklausų valdytojo, autorizavimo ir autentifikavimo. Pavyko pilnai atskirti jų funkcijas ir įrodyti, jog jos siūlomas prieigos valdymo metodo procesas yra praktiškai realizuojamas;
2. Pasirinkta Service Fabric platforma suteikia reikiamą infrastruktūrą, kuri turi visus esminius debesų kompiuterijos privalumus – paskirstytos programų sistemos valdymas, diegimas, komunikacija ir plėtimas. Ši platforma suteikia galimybę šią infrastruktūrą diegti ir į debesų kompiuteriją;
3. Sukurtos mikropaslaugos vykdo tik skaitymo operacijas duomenų bazėje, o esminės lentelės yra suskirstytos į dvi tarpusavyje mažai priklausomas dalis. Tai reiškia, jog siūlomo metodo greitaveiką galima pagerinti replikuojant duomenų bazes;
4. Sukurta bendrojo kodo biblioteka, kurioje dalinamasi bendromis sąsajomis, klasėmis ir metodais. Ši biblioteka galėtų būti perduodama klientams arba programinės įrangos kūrėjams su tikslu pagretinti siūlomo prieigos metodo integraciją;
5. Realizuotas žymų mechanizmas leidžia lengvai stebėti, kas, kada ir kokius veiksmus atliko su konkrečia užklausa. Tai leidžia geriau suprasti, kaip siūlomas metodas reaguoja į konkrečias sąlygas ir identifikuoti greitaveikos problemas.

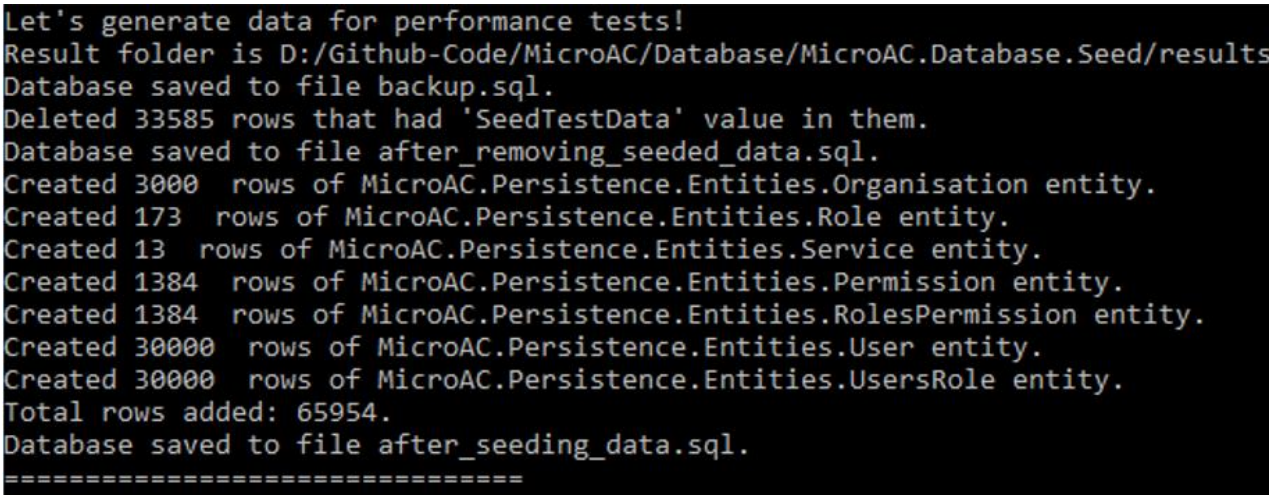
4. Prieigos valdymo mikropaslaugų architektūroje prototipo tyrimas

4.1. Prieigos valdymo prototipo tyrimo metodas ir įrankiai

Sukurtam prieigos valdymo metodo prototipo analizei pasirinktas greitaveikos tyrimas. Tyrimo tikslas yra išnagrinėti, kaip siūlomas prieigos valdymo metodas skirtingais režimais apdoroja įvairaus dydžio apkrovas. Tikslui pasiekti prieigos valdymo prototipui papildomai sukurta:

- Komunikacijos klasterio viduje optimizacija;
- Prototipo decentralizuotas darbo režimas be užklausų valdytojo;
- Automatizuotas duomenų bazės užpildymo įrankis;
- Aparatinės įrangos stebėjimo įrankis;
- 24 integraciniai testai prieigos valdymo ir verslo posistemėms;
- Greitaveikos testai su žymų analizės logika.

Norint įvertinti prieigos valdymo metodo greitaveiką, sistemos duomenų bazėje privalo būti pakankamai duomenų, kurie apibūdina naudotojus ir jų teises programų sistemoje. Kadangi naudotojų gali būti daug, nuspręsta suprogramuoti įrankį, kuris automatiškai užpildytų duomenų bazę su nurodytu kiekiu naudotojų. Atsižvelgiant į testų apkrovos dydžius ir tarpinių eksperimentų rezultatus, nuspręsta tarp visų greitaveikos testų turėti pastovų 30 tūkstančių naudotojų skaičių ir su jais susijusių duomenų. Kiekvienam naudotojui yra priskiriama viena rolė, o vienai rolei – penki leidimai. Duomenų generavimo metu užpildomos visos lentelės, įskaitant tarpines, kurios pavaizduotos **pav. 3.7**. Duomenų pildymo programos rezultatai pateikti **pav. 4.1**. Šis įrankis sukurtas su *F#* programavimo kalba ir priklauso *MicroAC.Database.Seed* programinio kodo paketui.



pav. 4.1 Automatizuoto duomenų bazės uždildymo įrankio vykdymo rezultatų pavyzdys.

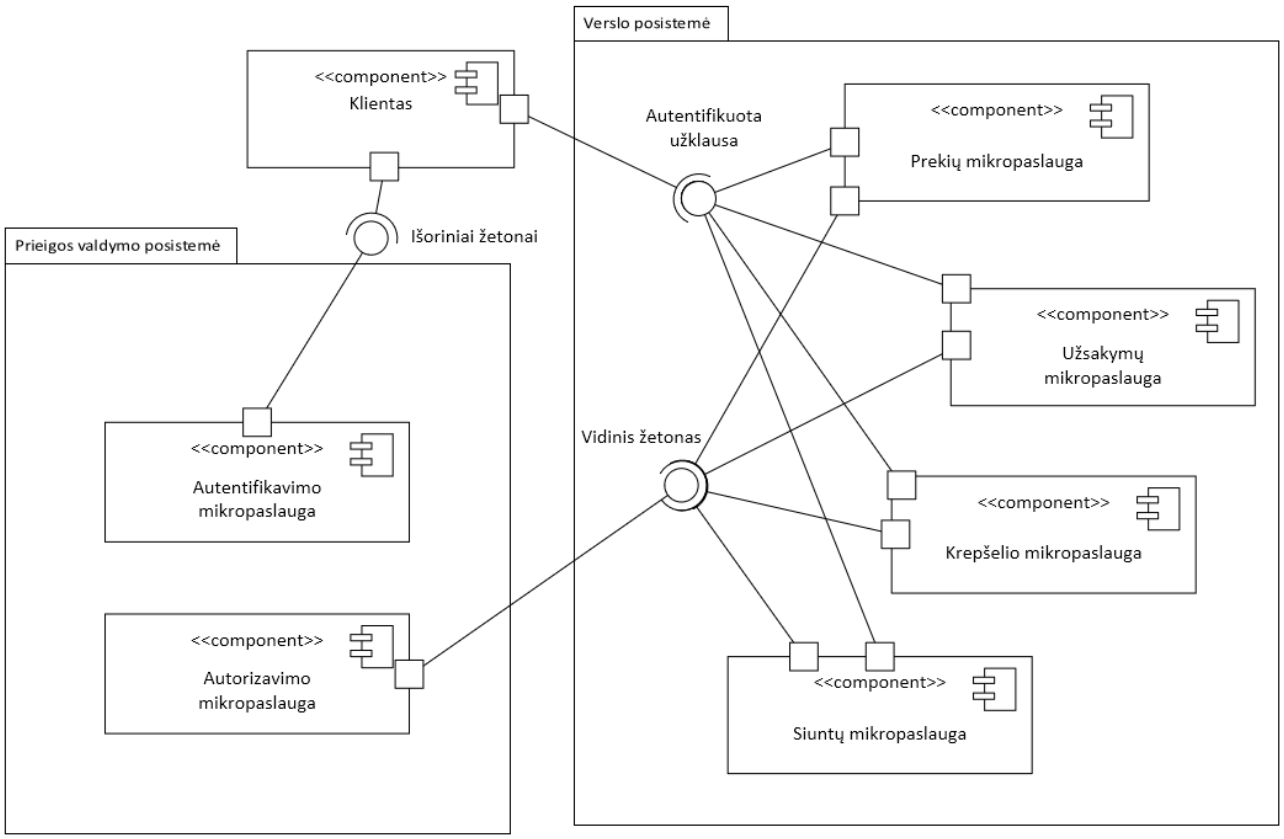
Norint suprasti prieigos valdymo prototipo naudojamus išteklius greitaveikos testų metu reikia surinkti informaciją apie darbinės stoties aparatinę įrangą. Todėl nuspręsta suprogramuoti įrankį, kuris kas sekundę fiksuoja procesoriaus bei operatyviosios atminties apkrovas ir jas spausdina į ekraną ir failą. Šis įrankis sukurtas su *C#* programavimo kalba ir priklauso *MicroAC.Performance.Tests.Monitor* programinio kodo paketui.

	A	B	C	D
1	Time	CPU	RAM MB	RAM %
2	16:16:44	0	13230	54
3	16:16:45	34	13227	54
4	16:16:46	10	13261	54
5	16:16:47	12	13238	54
6	16:16:48	28	13248	54
7	16:16:49	2	13238	54
8	16:16:50	16	13242	54
9	16:16:51	18	13248	54
10	16:16:52	14	13256	54
11	16:16:53	2	13244	54
12	16:16:54	4	13249	54
13	16:16:55	17	13237	54
14	16:16:56	8	13248	54
15	16:16:57	3	13238	54
16	16:16:58	6	13240	54
17	16:16:59	2	13227	54
18	16:17:00	6	13228	54

```
demo
Results file path: D:\Github-Code\MicroAC\Tests\Micr
oAC.Performance.Tests.Monitor\monitorResults\demo_20
22-05-05 16.16.44_monitorResults.csv
2022-05-05 16:16:44 | CPU: 0% | RAM:54%  - 13230Mb
2022-05-05 16:16:45 | CPU: 34% | RAM:54%  - 13227Mb
2022-05-05 16:16:46 | CPU: 10% | RAM:54%  - 13261Mb
2022-05-05 16:16:47 | CPU: 12% | RAM:54%  - 13238Mb
2022-05-05 16:16:48 | CPU: 28% | RAM:54%  - 13248Mb
2022-05-05 16:16:49 | CPU: 2% | RAM:54%  - 13238Mb
2022-05-05 16:16:50 | CPU: 16% | RAM:54%  - 13242Mb
2022-05-05 16:16:51 | CPU: 18% | RAM:54%  - 13248Mb
2022-05-05 16:16:52 | CPU: 14% | RAM:54%  - 13256Mb
2022-05-05 16:16:53 | CPU: 2% | RAM:54%  - 13244Mb
2022-05-05 16:16:54 | CPU: 4% | RAM:54%  - 13249Mb
2022-05-05 16:16:55 | CPU: 17% | RAM:54%  - 13237Mb
2022-05-05 16:16:56 | CPU: 8% | RAM:54%  - 13248Mb
2022-05-05 16:16:57 | CPU: 3% | RAM:54%  - 13238Mb
2022-05-05 16:16:58 | CPU: 6% | RAM:54%  - 13240Mb
2022-05-05 16:16:59 | CPU: 2% | RAM:54%  - 13227Mb
2022-05-05 16:17:00 | CPU: 6% | RAM:54%  - 13228Mb
Pausing execution.
Enter prefix for new results file:
second demo
```

pav. 4.2 Darbinės stoties aparatinės įrangos stebėjimo įrankio rezultatų pavyzdys.

Siūlomo prieigos metodo mikropaslaugų architektūroje ypatumų išryškinimui suprogramuotas antras prototipo darbo režimas, kuris paremtas decentralizuota autorizacija, kaip pavaizduota **pav. 4.3**. Siūlomas metodas, pavaizduotas **pav. 3.4**, remiasi centralizuota autorizacija, kurioje užklausų valdytojas gauna visas užklausas iš kliento ir, jeigu reikia, jas autorizuoja kviesdamas autorizacijos mikropaslaugą. Palyginimui sukurtas decentralizuotos autorizacijos režimas, kuriame nėra užklausų valdytojo ir kiekviena verslo mikropaslauga savarankiškai autorizuoja kiekvieną užklausą. Tai reiškia, jog jeigu viena kliento užklausa sugeneruos daugiau vidinių užklausų vidinėje verslo posistemėje, tuomet kiekviena mikropaslauga atskirai kreipsis į autorizavimo mikropaslaugą ir sugeneruos papildomą srautą, kuris centralizuoto autorizavimo metu yra išvengiamas. Kitaip tariant, centralizuoto autorizavimo metu kliento užklausa yra visuomet autorizuojama tik vieną kartą, o decentralizuoto – keletą, priklausomai nuo sugeneruotų vidinių užklausų kiekio. Šis režimas leidžia palyginti, ar greitaveikos atžvilgiu naudoti centralizuotą užklausų valdytoją yra efektyviau nei decentralizuotą, savarankišką autorizaciją.



pav. 4.3 Decentralizuoto prieigos valdymo metodo komponentų diagrama

Greitaveikos testavimas tikrina prototipo efektyvumą, kai naudojamas centralizuotas **pav. 3.4** ir decentralizuotas **pav. 4.3** veikimo režimai kartu tu optimizuota ir neoptimizuota komunikacija. Tyrimo metu stebima, kaip kiekvienam rėžimui pavyksta susidoroti su mažu, vidutiniu ir dideliu kiekiu nuolatos siunčiamų užklausų. Todėl tyrime vykdomi iš viso 12 greitaveikos testų kurie sudaryti iš 4 režimų su 3 skirtingomis apkrovomis. Greitaveikos testas suprogramuotas vykdyti nurodytą gijų skaičių, kurios nuolatos siunčia užklausas į visas esamas prototipo funkcijas, kurios aprašytos **lentelė 3.1**. Testuose naudojamas pastovus gijų skaičius, kuris priklauso nuo apkrovos:

- Maža apkrova vykdo 1 giją
- Vidutinė apkrova vykdo 3 gijas
- Didelė apkrova vykdo 6 gijas

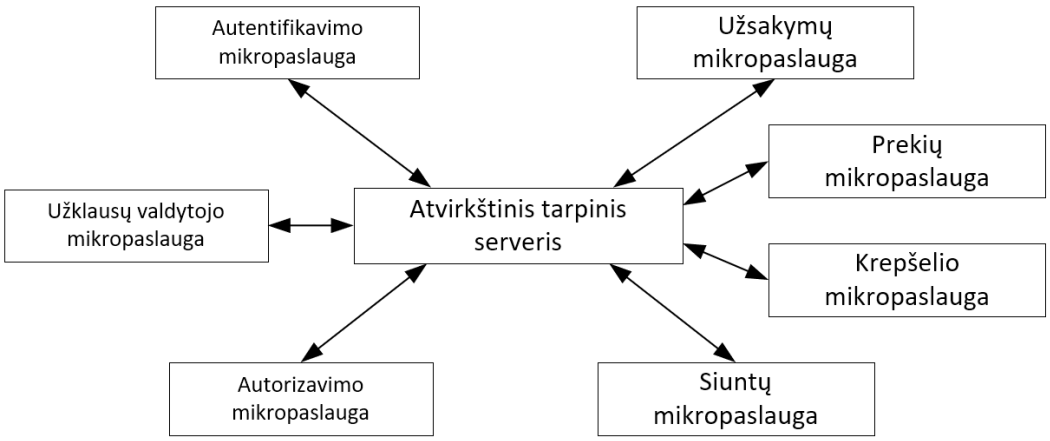
Gijų kiekis kiekvienai apkrovai parinktas pagal tarpinių eksperimentų rezultatus. Jų metu nustatyta, jog 6 gijos yra didžiausia apkrova, kurią gali apdoroti darbinėje stotyje įdiegtas prototipas. Giją galima interpretuoti kaip klientą, kuris be pertraukos nuosekliai siunčia užklausas ir laukia atsakymo. Ši strategija generuoja linijinę apkrovą programų sistemai, todėl toks testas simuliuoja stabilų užklausų srautą iš klientų. Pasirinkta testo trukmė – 20 minučių. Tarpinių eksperimentų metu pastebėta, jog tai yra pakankamas laiko tarpas, per kurį galima pastebėti anomalijas bei užtikrinti pastovius rezultatus. Prieš kiekvieną testą per 3 minutes yra perkraunamas *Service Fabric* klasteris, per 2 minutes vykdomi integraciniai testai ir paleidžiamas 1 minutės trukmės apšilimas, kuris generuoja kas sekundę generuoja vienos gijos apkrovą. Klasterio perkrovimas leidžia pašalinti nereikalingus duomenis iš atminties bei užtikrinti vienodas sąlygas kiekvieno testo vykdymo metu. Apšilimas reikalingas užtikrinti, jog gerokai ilgesni pirmųjų užklausų laikai neiškreips esminių

tyrimo duomenų, nes jo metu užkraunami įvairūs moduliai, vyksta procesoriaus ir atminties paruošimas intensyviam darbui. Apšilimo metu sugeneruoti duomenys nėra nei fiksuojami, nei įtraukiami į galutinius rezultatus. Taigi, bendra minimali tyrimo trukmė duomenims surinkti yra 312 minutės arba 5 val. ir 12 minučių (12 testų po 26 minutes).

Greitaveikos testų įrankis sukurtas *F#* programavimo kalba ir priklauso *MicroAC.Performance.Tests* programinio kodo paketui. Po kiekvienos įvykdytos užklausos yra surenkamos antraštėje esančios žymos, kurias sugeneravo kiekviena prie užklausos apdorojimo prisidėjusi mikropaslauga. Esminė informacija žymose yra: mikropaslaugos pavadinimas, mazgo numeris, kuriam priklauso mikropaslauga ir užklausos apdorojimo mikropaslaugoje pradžios bei pabaigos laikai. Po 20-ies minučių, pabaigus apdoroti visas užklausas, yra vykdomas žymų analizės algoritmas, kuris yra individualiai suprogramuotas tirti šio prototipo greitaveikai. Algoritmas įvairiais skerspjūviais apdoroja žymų duomenis ir gautus rezultatus spausdina į failą.

4.2. Prieigos valdymo metodo mikropaslaugų architektūroje komunikacijos optimizacija

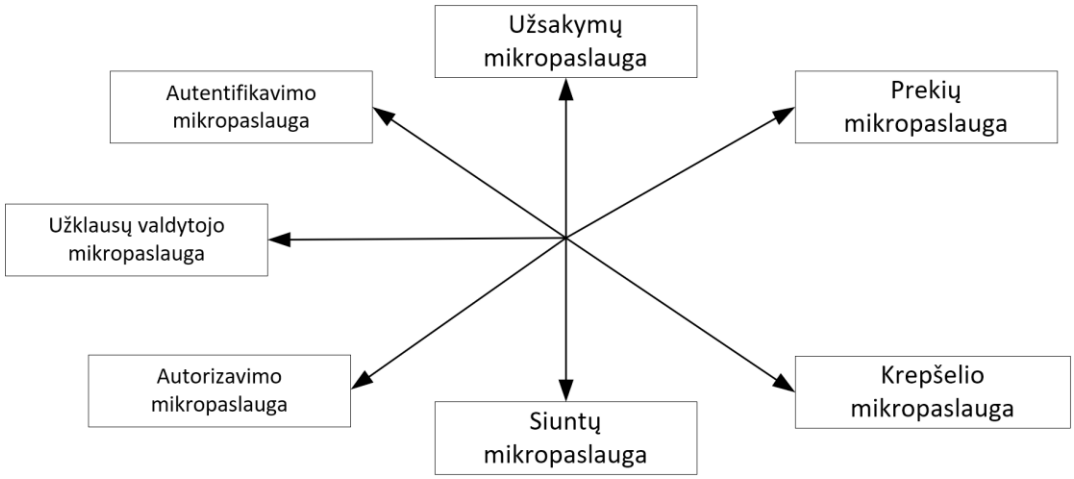
Prieš atliekant galutinį greitaveikos tyrimą, nuspręsta pagerinti prototipo realizaciją optimizuojant komunikaciją tarp mikropaslaugų klasterio viduje. Tarpinių eksperimentų metu pastebėta, jog mažą srauto didėjimą lydi nemažas užklausų apdorojimo laikų didėjimas. Išnagrinėjus situaciją buvo nustatyta, jog *Service Fabric* dokumentacijoje [14] rekomenduojamas naudoti atvirkštinis tarpinis serveris (angl. *Reverse Proxy*) turi nepaminėtų greitaveikos problemų. Mazge esantis atvirkštinis įgaliotasis serveris, vaizduojamas **pav. 4.4** realiu laiku persiunčia užklausą į mikropaslaugą, kurios pavadinimas yra nurodytas užklausos kelyje. Tuomet tarpinis atvirkštinis serveris persiunčia užklausą į reikiamą mikropaslaugą bei laukia atsakymo iš jos. Tačiau naudojant šį komunikacijos modelį buvo pastebėtas nelogiškai ilgas užklausų apdorojimo laikas.



pav. 4.4 Komunikacija tarp mikropaslaugų per atvirkštinį tarpinį serverį

Atsižvelgus į šią greitaveikos problemą nuspręsta sukurti mechanizmą, kuris, naudodamasis *Service Fabric* bibliotekomis, sudaro visų klasteryje esančių mikropaslaugų adresus, juos kešuoja ir naudoja visų testų metu. Tai leidžia išvengti kreipimosi į atvirkštinį tarpinį serverį kiekvienos užklausos metu, kaip vaizduojamas **pav. 4.5**. Taip sumažinamas bendras užklausų srautas klasteryje ir taupomi darbinės stoties resursai, tačiau kiekviena mikropaslauga turi naudoti specialiai sukurtą ir į mikropaslaugą integruotą programinę klasę. Ji tinkama lygiagrečiam užklausų vykdymui bei pašalina reikiamybę naudoti papildomą elementą klasteryje, kuris apkrauną tinklą. Šio komunikacijos modelio

realizacija yra *MicroAC.Core.Client* programinėje bibliotekoje, kurią naudoja prieigos valdymo, verslo logikos, integracinių ir greitaveikos testų programiniai paketai.



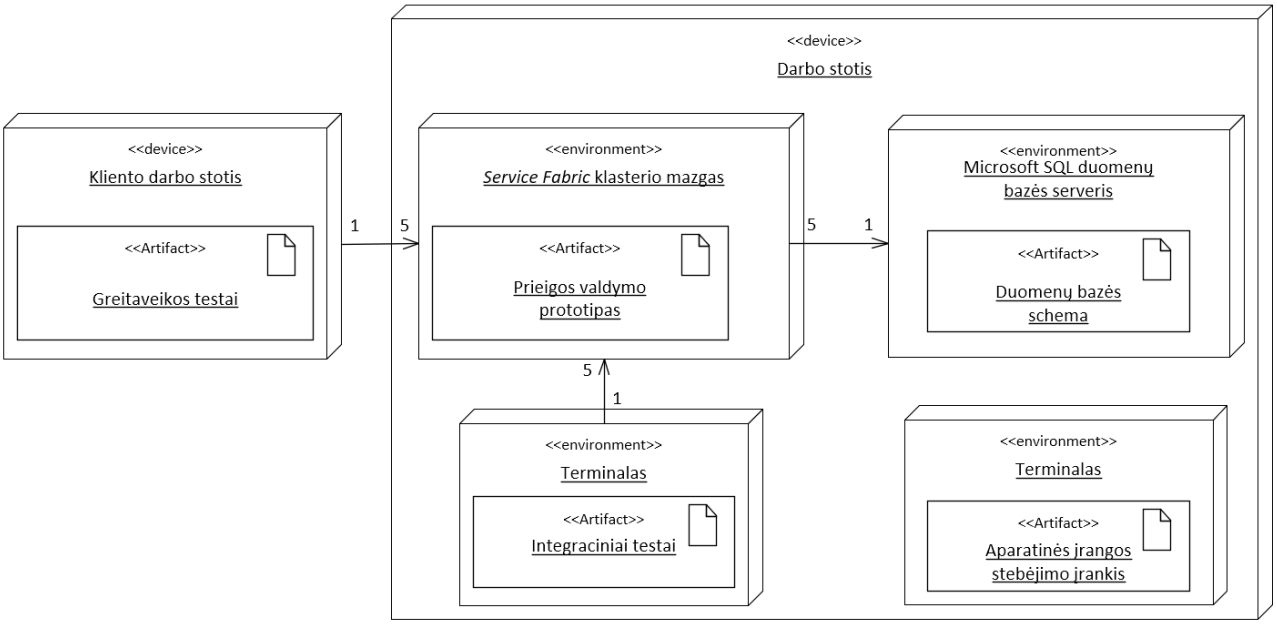
pav. 4.5 Tiesioginė komunikacija tarp mikropaslaugų

4.3. Prieigos valdymo metodo tyrimo aplinka mikropaslaugų architektūroje

Tyrimo metu greitaveikos testai siunčia užklausas iš kliento darbo stoties į darbo stotį, kurioje yra įdiegtas *Service Fabric* klasteris, kartu su visu prieigos valdymo metodo prototipu mikropaslaugų architektūroje. Tai vaizduojama **pav. 4.6**. Abi darbinės stotys yra sujungtos į tą patį vietinį tinklą. Prieigos valdymo prototipas taip pat naudoja duomenų bazę, kuri yra įdiegta toje pačioje darbo stotyje. Darbo stotyje taip pat veikia aparatinės įrangos stebėjimo įrankis, kuris surenka informaciją apie procesoriaus bei operatyviosios atminties naudojimą ir šią informacija rašo į failą. Taip pat prieš kiekvieną greitaveikos testą yra paleidžiami integraciniai testai, kurie užtikrina, jog klasteris buvo įdiegtas sėkmingai ir programų sistema yra pasiruošusi greitaveikos testams. Klasteris turi 5 mazgus, o kiekviename mazge yra įdiegta po vieną mikropaslaugos repliką. Tai reiškia, jog darbo stotyje vienu metu yra vykdomi 35 atskiri prototipo mikropaslaugų procesai (7 mikropaslaugos 5-iuose mazguose, kaip vaizduojama **pav. 3.9**).

Pagrindinė darbo stotis turi šiuos techninius parametrus:

- Windows 10 64 bitų operacinė sistema;
- *Intel Core* i5-6600K procesorius;
  - 4 branduoliai;
  - 3,50 GHz taktinis dažnis;
  - 6-oji karta, 2015 metų gamyba;
- 24 GB RAM, DDR-4 1330 MHz;
- *Service Fabric Runtime* 8.1.329 ir *SDK* 5.1.329;
- *Microsoft SQL Server* 2019 (15.0.2080.9).



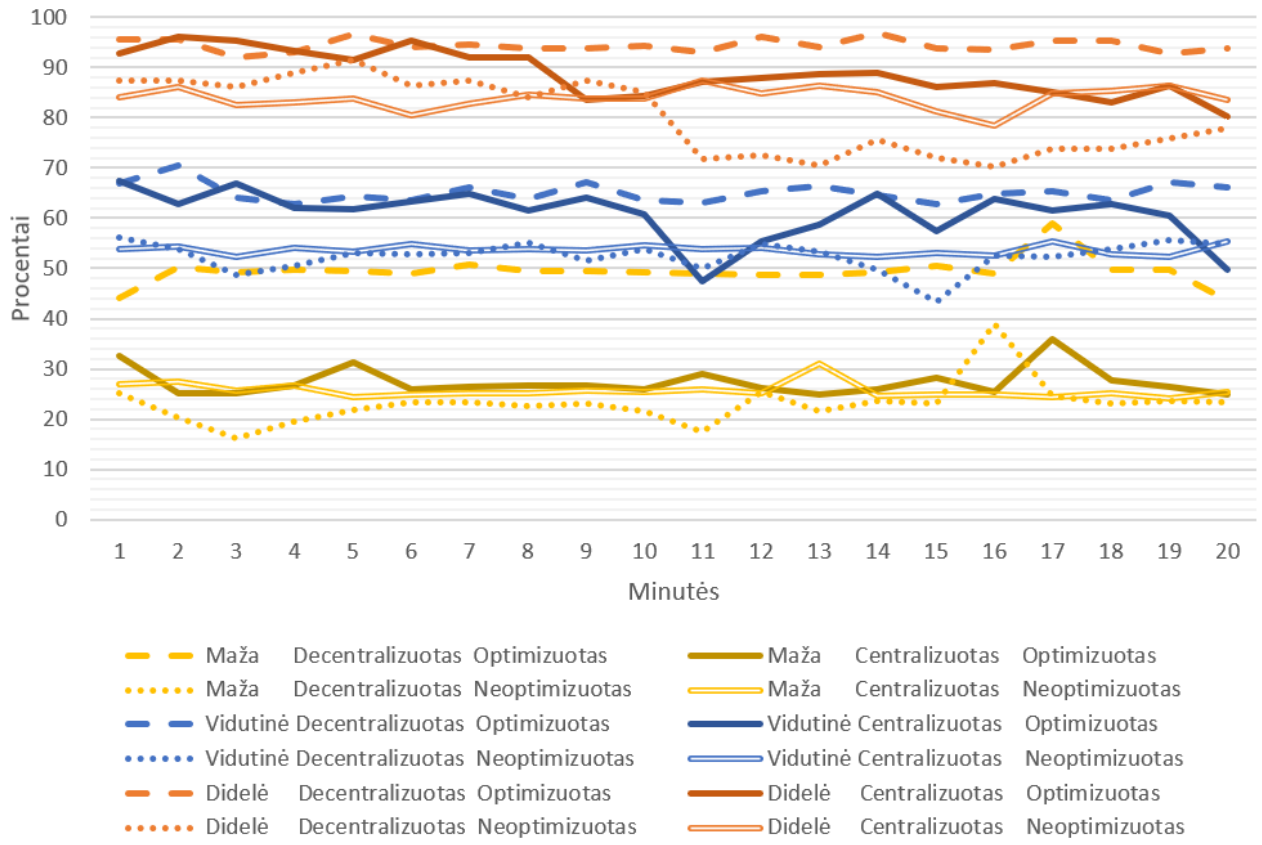
pav. 4.6 Prototipo greitaveikos tyrimo diegimo diagrama

4.4. Prieigos valdymo metodo mikropaslaugų architektūroje tyrimo rezultatai

Šiame skyrelyje pateiktos diagramos, kurios analizuoja visus 12 greitaveikos testus. Diagramos nubraižytos pagal duomenis, kurie surinkti testų vykdymo metu ir yra pateikti **0** priede. Diagramose kiekvienas testas turi savo unikalų pavadinimas formate „apkrova režimas optimizacija“, kuriame:

- Apkrova (maža, vidutinė, didelė) nusako gijų kiekį (1, 2, 3) kliento darbo stotyje, Ji žymima geltona, mėlyna ir raudona spalvomis;
- Režimas nusako, ar naudojama centralizuota (**pav. 3.4**) ar decentralizuota (**pav. 4.5**) autorizacija. Centralizuotas režimas žymimas vientisu kontūro užpildu, o decentralizuotas – atvirkščiai;
- Optimizacija nusako, ar naudojama vidinė klasterio optimizacija, kuri aptarta poskyryje 4.2. Diagramose ji išskiriama skirtingais užpildo tipais.

Procesoriaus apkrova **pav. 4.7** rodo, jog centralizuotas režimas naudoja mažiau procesoriaus resursų nei decentralizuotas. Su maža apkrova centralizuotas režimas veikia beveik dvigubai efektyviau nei decentralizuotas. Vidutinės ir didelės apkrovos atveju vidutiniškas skirtumas siekia 5% procesoriaus apkrovos tarp centralizuoto ir decentralizuoto rėžimų. Taip pat galima pastebėti, jog tiek vidutinės, tiek didelės pakrovos metu neaptimizuotas režimas naudoja vidutiniškai 6% mažiau procesoriaus resursų. Prieš darant išvadą, jog neoptimizuotas metodas veikia efektyviau, reikėtų atkreipti dėmesį į **pav. 4.8** ir **pav. 4.13**. Šie grafikai simbolizuoja prastesnį neoptimizuotos komunikacijos greitį. Atsižvelgiant į tai, jog neoptimizuota komunikacija sunaudojo daugiau operatyviosios atminties ir lėčiau apdorojo mažesnį kiekį užklausų, galima daryti prielaidą, jog mažesnis procesoriaus resursų panaudojimas reiškia ne efektyvesnį komunikacijos modelį, o priešingai – atvirkštinis tarpinis serveris ilgiau laukia užklausų atsakymų iš vidinių mikropaslaugų, todėl neefektyviai panaudoja procesoriaus resursus. Taip galima teigti dėl to, nes tiek centralizuoto, tiek neoptimizuoto režimo testų rezultatai buvo surinkti vienodomis tyrimo sąlygimos, įskaitant užklausų apkrovų intensyvumą.

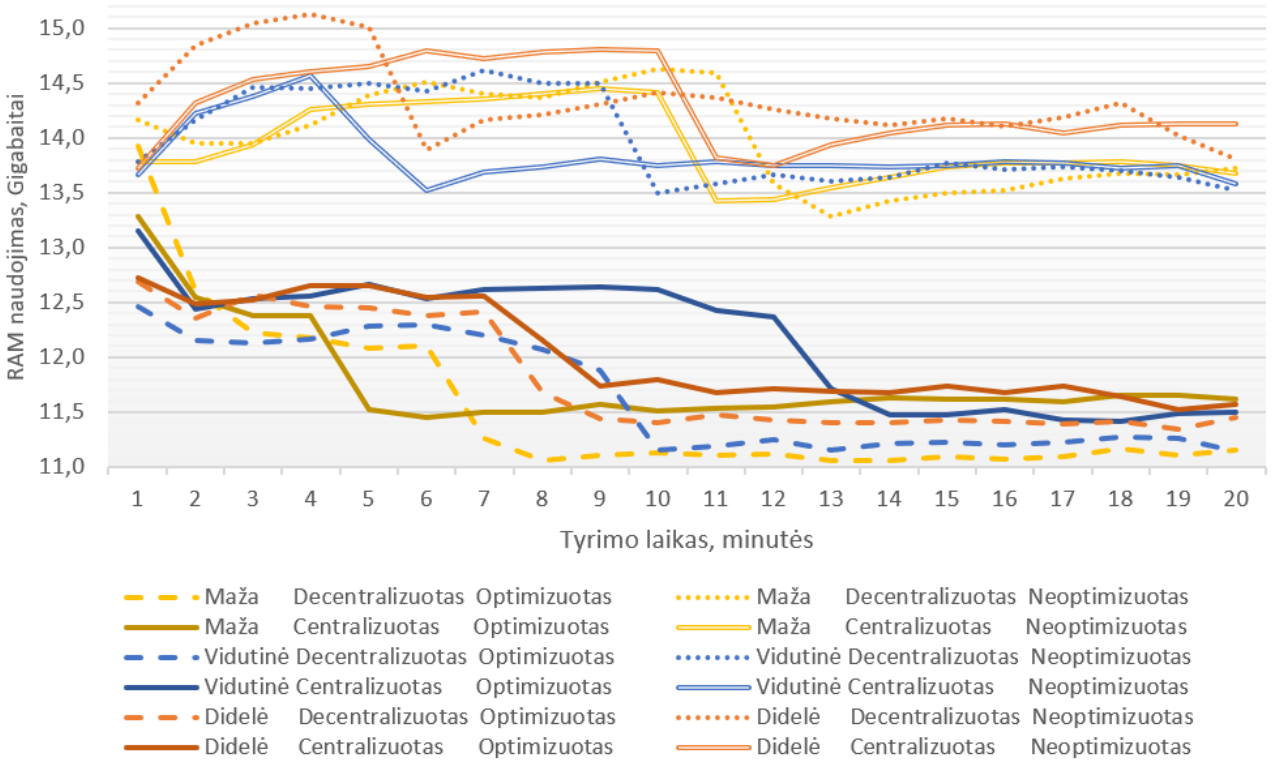


pav. 4.7 Prototipo procesoriaus naudojimas

Žvelgiant į operatyviosios atminties naudojimą **pav. 4.8**, matomas grafikas, kurio linijos leidžiasi žemyn. Pirmasis susikirtimo taškas yra ties antra minute, kai atminties naudojimas nukrenta apytiksliai iki 12GB. Toks spartus atminties naudojimo sumažėjimas, kuris matomas visuose testuose, galėtų reikšti atminties atsistatymą po intensyvaus apšilimo. Sekantis staigus kritimas, kuris įvyksta po 3 ar 6 minučių, veikiausiai simbolizuoja nenaudojamų sisteminių *Service Fabric* modulių iškrovimą iš atminties. Likęs laiko tarpas, kuris yra tarp 12 ir 7 minučių iki grafiko pabaigos yra tolygiai stabilus. Tai reiškia, kad visų testų atminties naudojimas darbinėje stotyje svyruoja 11 ir 11,8 GB rėžyje. Grafike nėra staigaus kylimo ir matomas bendras naudojimo mažėjimas laiko atžvilgiu. Tai rodo, kad prototipo realizacijoje nėra spragų atminties naudojime, kurios sukeltų anomalijų ir žymius atminties naudojimui šuolius. Tarpiniuose tyrimuose buvo identifikuotos atminties valdymo klaidos, kurios ištaisytos prieš atliekant galutinius greitaveikos testus. Taip pat svarbu paminėti, jog po klasterio išjungimo ir prieš jo paleidimą, atminties sunaudojimą siekė vidutiniškai 6 GB. Todėl galima teigti, jog prototipo veikimui reikia tarp 5 ir 6 GB atminties. Tai reiškia, jog kiekvienas iš 35 prototipo mikropaslaugų procesų naudoja iki 200 megabaitų atminties darbinės atminties optimizuoto režimo atveju.

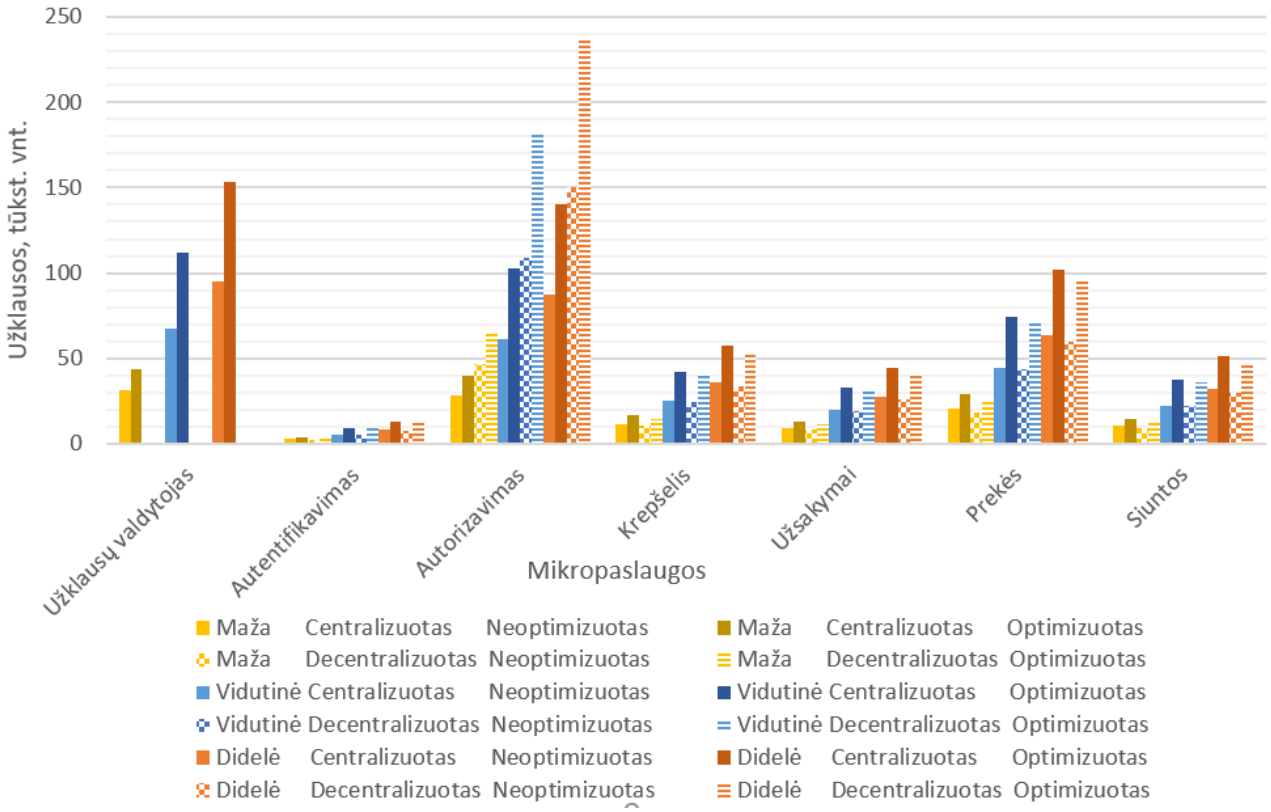
Tačiau neoptimizuotas režimas rodo kiek prastesnę situaciją nei optimizuotas. Testo pradžioje matomas staigus atminties šuolis neoptimizuotame režime vietoje sumažėjimo, kaip centralizuotame režime. Tarp 4 ir 10 minučių yra pasiekiamas pikas, kurį seka staigus nusileidimas nuo apytiksliai 14,5 GB iki 13,7 GB. Panašus nusileidimas matomas ir centralizuoto režimo atveju. Nuo 13 minutės, įpusėjus testui, matomas stabilus linijinis atminties naudojimas tiek centralizuoto, tiek neoptimizuoto režimo atveju. Apibendrinus grafiką, tendencija iš esmės yra vienoda, tačiau neoptimizuoto režimo

atveju ji yra pasistumėjusi kiek daugiau nei 2GB į viršų. Todėl galima drąsiai teigti, jog neoptimizuotas atvirkštinio tarpinio serverio naudojimas *Service Fabric* klastyreje pareikalavo trečdaliu daugiau operatyviosios atminties.



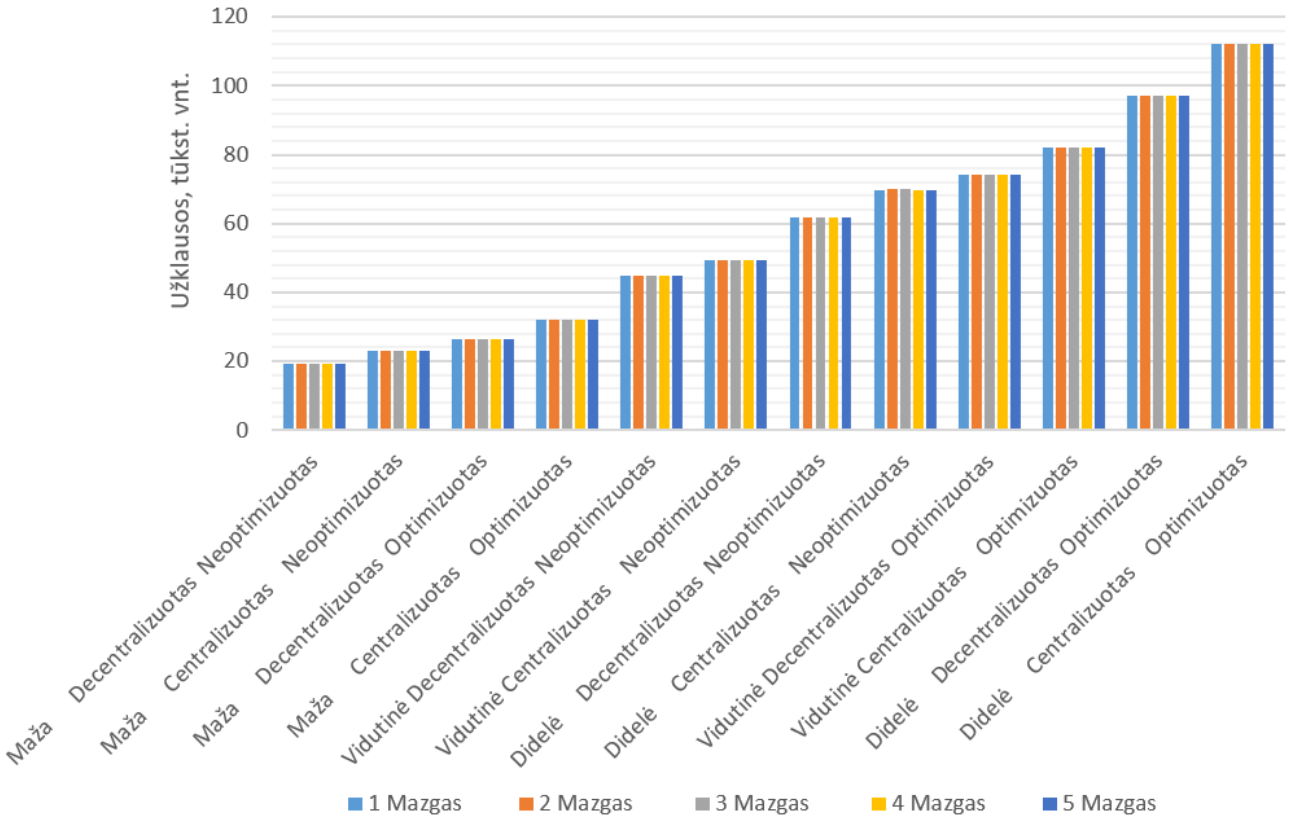
pav. 4.8 Operatyviosios atminties naudojimas

Užklausų pasiskirstymas mikropaslaugose **pav. 4.9** identifikuoja vieną iš esminių skirtumų tarp centralizuoto ir decentralizuoto rėžimų. Centralizuotas autorizacijos rėžimas naudoja užklausų valdytojo mikropaslaugą, o decentralizuotas – ne. Tačiau centralizuotas rėžimas šį skirtumą kompensuoja mažesniu užklausų kiekiu autorizavimo mikropaslaugai. Tačiau išaugęs autorizavimo užklausų kiekis decentralizuotame režime reiškia didesnį darbo stoties resursų sunaudojimą – priešingai nei užklausų valdytojas, autorizavimo mikropaslauga papildomai kreipiasi į duomenų bazę ir ieško duomenų, susijusių su naudotojo rolėmis ir leidimais. Šis skirtumas yra labai didelis – optimizuoto decentralizuoto režimo atveju autorizavimo mikropaslaugai vidutiniškai siunčiama 60% daugiau užklausų nei centralizuotame. Didelės apkrovos metu decentralizuotame režime autorizavimo mikropaslauga apdoroja beveik 100 tūkst. užklausų daugiau, nei centralizuotame, o tai sudaro beveik 40% visų režimo vidinių užklausų.



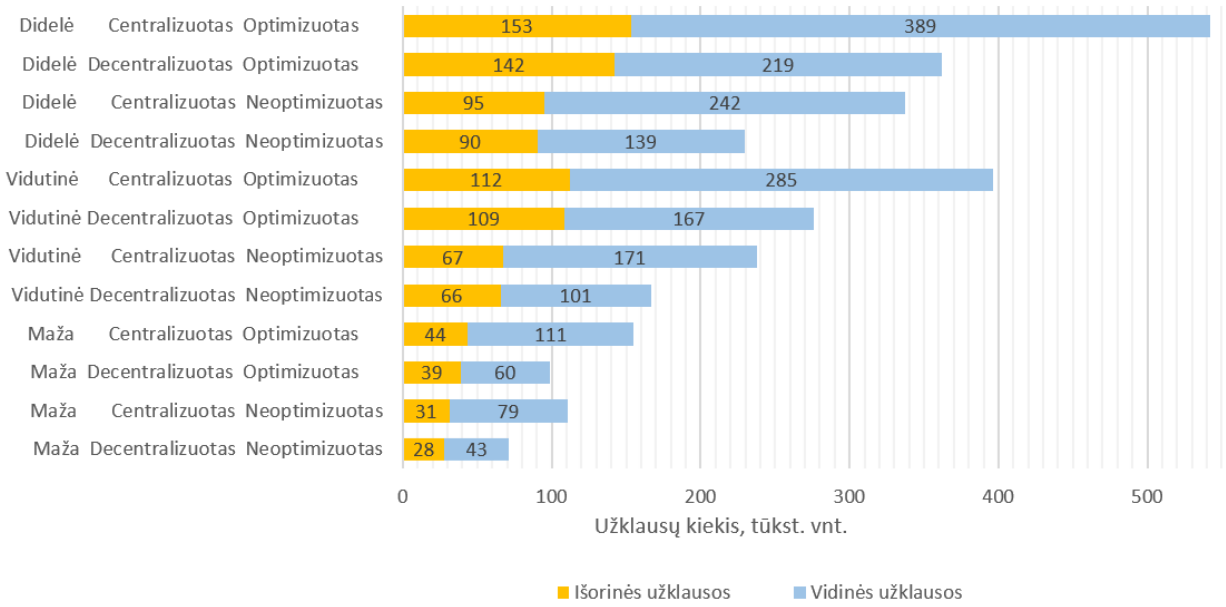
pav. 4.9 Užklausų pasiskirstymas mikropaslaugose

Užklausų pasiskirstymas mazgų atžvilgiu yra idealus, kaip vaizduojama **pav. 4.10**. Tai reiškia, jog vidinės komunikacijos klasteryje optimizacijos metu pritaikytas srauto paskirstymo algoritmas pasiteisino. Algoritmas turi vieną pagrindinę taisyklę – kiekvienai siunčiamai užklausai pateikti sekantį mikropaslaugos adresą žiedinės iteracijos (angl. *Circular Iteration*) principu. Šiam algoritmui pritaikytas monitorius kritinei sekcijai apsaugoti nuo kelių gijų, kurios vienu metu bando atlikti žiedinę iteraciją. Tokia komunikacijos optimizacija užtikrina tolygų užklausų pasiskirstymą tarp mazgų, todėl tyrimo metu išvengta situacijos, kai dėl neproporcingo srauto mikropaslaugų resursai būtų neefektyviai išnaudojami. Todėl darytina išvada, jog visuose režimuose užklausos yra tolygiai paskirstomos mazguose, todėl šiuo atžvilgiu realizuota komunikacijos optimizacija pilnai patenkina atvirkštiniam tarpiniam serveriui keliamus užklausų paskirstymo reikalavimus.



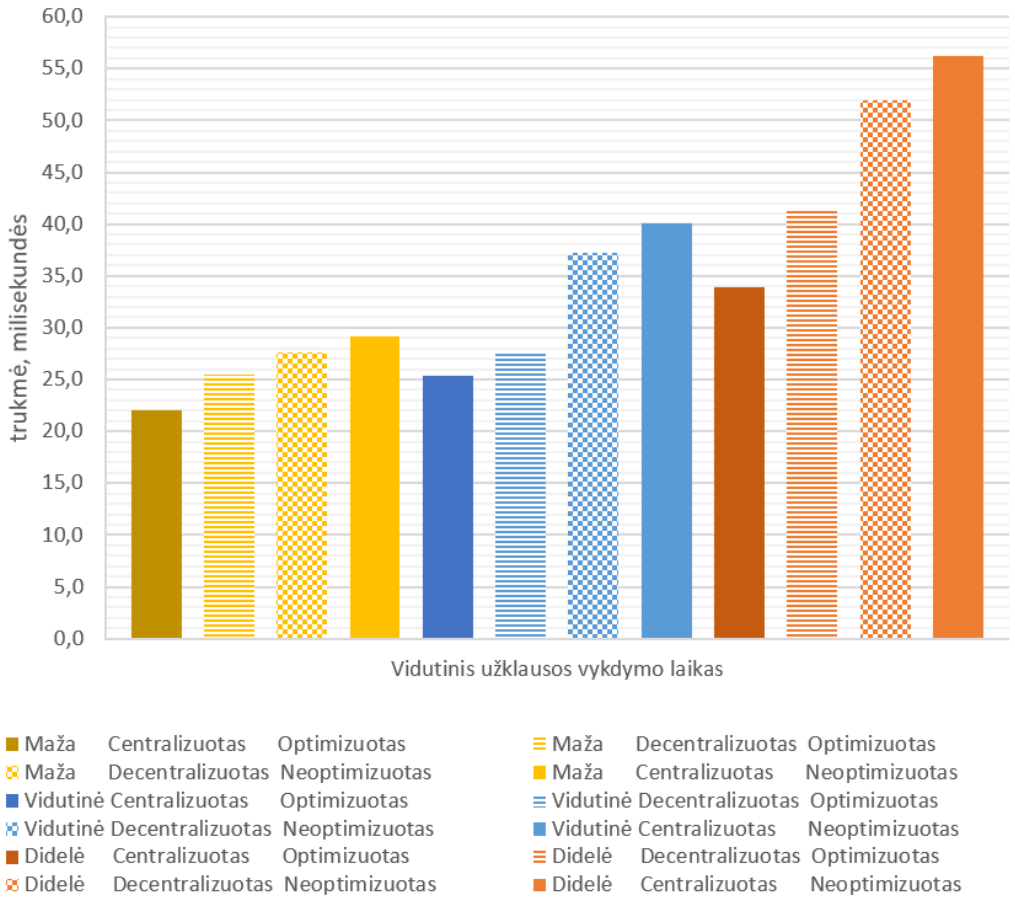
pav. 4.10 Užklausų pasiskirstymas mazguose

Vertinant bendrus išorinių ir vidinių užklausų kiekius **pav. 4.11**, centralizuotas rėžimas apdorojo 13%, 3% ir 8% daugiau išorinių užklausų su atitinkamai maža, vidutine ir didele apkrovomis. Nors centralizuotas režimas bendrai apdorojo 8% daugiau išorinių užklausų nei decentralizuotas, tačiau vidinių užklausų centralizuotame režime buvo siunčiama 50% daugiau. To priežastis, kuri atsispindi **pav. 4.9** – užklausų valdytojo mikropaslauga, kuri privalo apdoroti visas išorines užklausas. Todėl vidinis tinklas yra apkraunamas papildomu užklausų kiekiu, kuris lygus išorinių užklausų kiekiui. Tačiau ši apkrova centralizuotam režimui netrukdo apdoroti užklausas greičiau sunaudojant mažiau darbo stoties resursų, nes užklausų valdytojo pagrindinė atsakomybė yra laukti atsakymo iš kitų mikropaslaugų, o šis veiksmas laiko ir procesoriaus resursų prasme yra pigesnis nei autorizavimo mikropaslaugos užklausos vykdymas.



pav. 4.11 Bendri užklausų kiekiai

Bendri išorinių užklausų vykdymo laikai rodo, jog optimizuotame decentralizuotame režime užklausa vidutiniškai buvo vykdoma ilgiau, nei centralizuotame. Ši tendencija sutampa su užklausų pasiskirstymų mikropaslaugose (**pav. 4.9**) ir procesoriaus apkrova (**pav. 4.7**) bei atspindi mažesnį decentralizuoto režimo efektyvumą. Pagrindinė priežastis – perteklinis užklausų siuntimas į autorizavimo mikropaslaugą. Vykdymo trukmių aritmetiniai vidurkiai vaizduojami **pav. 4.12**, jie sudaryti iš 975 tūkst. išorinių užklausų. Optimizuotas centralizuotas režimas mažos, vidutinės ir didelės apkrovos metu apdoroja užklausas atitinkamai 15%, 9% ir 22% greičiau. Todėl galima teigti, jog centralizuotas optimizuotas režimas apdoroja užklausas 15% greičiau nei decentralizuotas, su tokiu pačiu užklausų apkrovos intensyvumu. Tuo tarpu neoptimizuotas režimas užklausas vykdo 16%, 31% ir 30% lėčiau su maža, vidutine ir didele apkrovomis, o vidutiniškai – 26%. Didžiausias atotrūkis yra užfiksuotas tarp centralizuoto optimizuoto ir neoptimizuoto režimų. Natūralus kandidatas lėčiausiam režimui yra decentralizuotas neoptimizuotas, tačiau jis yra prieš paskutinėje vietoje. To priežastis – centralizuotas režimas generuoja daugiau vidinių užklausų, nei decentralizuotas, o neoptimizuotame režime šios užklausos apdorojamos vidutiniškai 8%, 11% ir 12% lėčiau. Apibendrinant bendrus užklausų vykdymo laikus darytina išvada, jog prieigos valdymo metodo centralizavimas ir optimizavimas gerokai padidino prototipo efektyvumą laiko atžvilgiu.



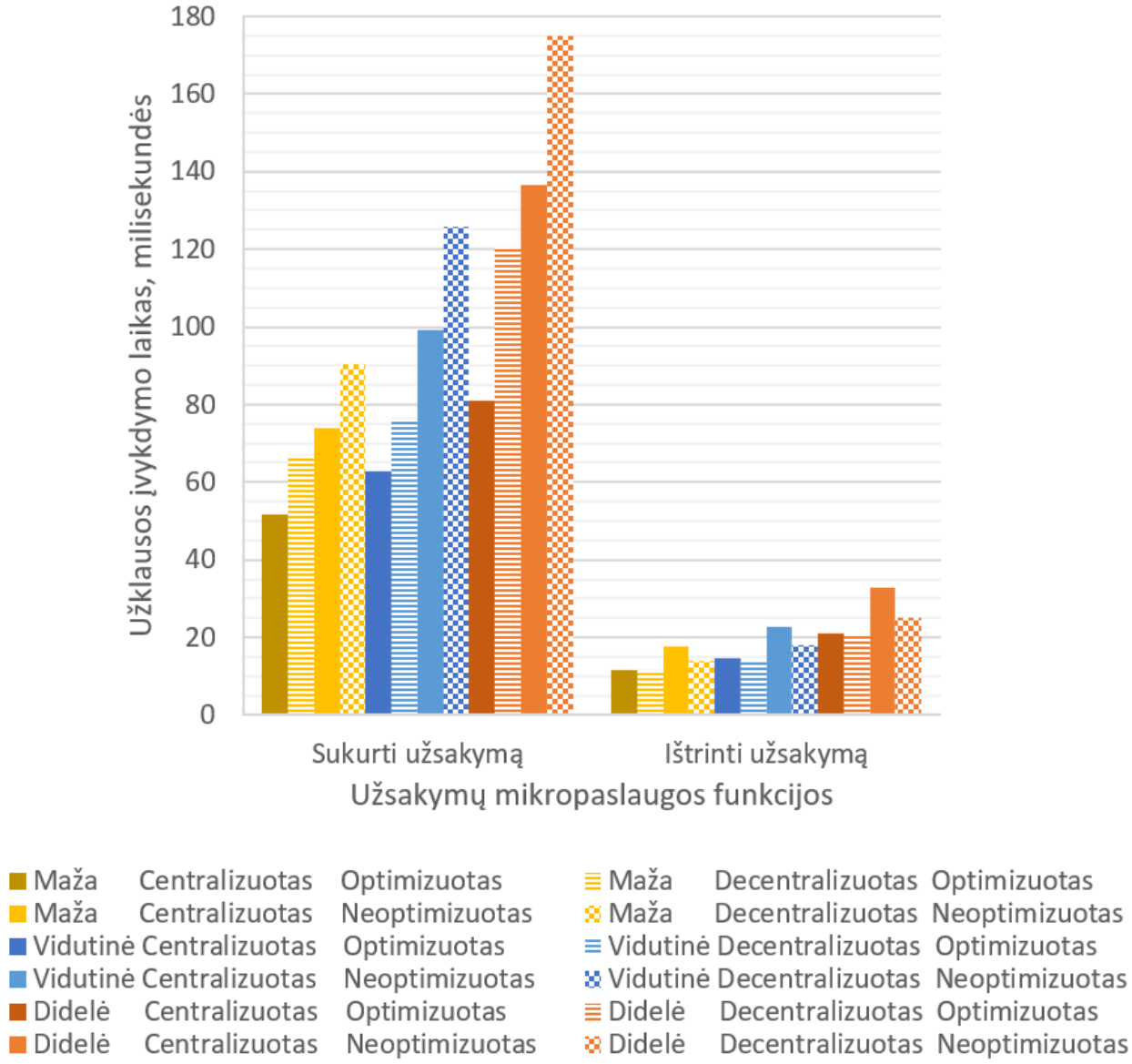
pav. 4.12 Vidutinė išorinių užklausų įvykdymo trukmė

Tačiau centralizuotas režimas ne visais atvejais yra efektyvesnis už decentralizuotą. Vidutinės išorinių užklausų apdorojimo trukmės užsakymų mikropaslaugos funkcijose yra vaizduojamos **pav. 4.13**. Šis skerspjūvis nusako, kiek laiko vidutiniškai kliento užklausa buvo apdorojama, įskaitant užklausų valdytojo ir autorizavimo mikropaslaugos laiko kaštus. Diagramoje parinktos dvi užsakymų funkcijos:

- Sukurti užsakymą – kreipiasi į tris kitas mikropaslaugų funkcijas;
- Ištrinti užsakymą – nesikreipia į kitas mikropaslaugas.

Jeigu funkcija nesikreipia į kitas mikropaslaugas, optimizuotas decentralizuotas režimas tokiu atveju efektyvesnis beveik 1%. Vieno procento skirtumas slypi užklausų valdytojo mikropaslaugos nenaudojime. Tačiau jeigu funkcija kreipiasi į kitas mikropaslaugas, optimizuotas centralizuotas režimas kliento užklausas apdoroja vidutiniškai 31% greičiau nei decentralizuotas. Tai reiškia, jog užklausų valdytojo naudojimas pasiteisina tik tuo atveju, jeigu verslo posistemėje esančios mikropaslaugos tarpusavyje siunčia autorizuotas vidines užklausas. Užsakymo sukūrimo ir pateikimo funkcijų atveju (**pav. 4.13**) centralizuotas režimas yra greitesnis 28%, 20% ir 45% su atitinkamai maža, vidutine ir didele apkrovomis. Todėl optimizuotas centralizuotas režimas yra 31% efektyvesnis su funkcijomis kurios generuoja vidines užklausas, tačiau 1% lėtesnis su funkcijomis, kurios negeneruoja vidinių užklausų. Taip pat neoptimizuotas režimas yra vidutiniškai 34% lėtesnis nei optimizuotas. Šis skirtumas yra beveik 20% mažesnis nei **pav. 4.12**, nes ten vaizduojami apibendrinti užklausų laikai, o šiame grafike (**pav. 4.13**) parinktos funkcijos turi didžiausią atotrūkį su tikslu įrodyti, jog siūlomo metodo efektyvumas iš esmės priklauso nuo to, ar mikropaslaugos verslo

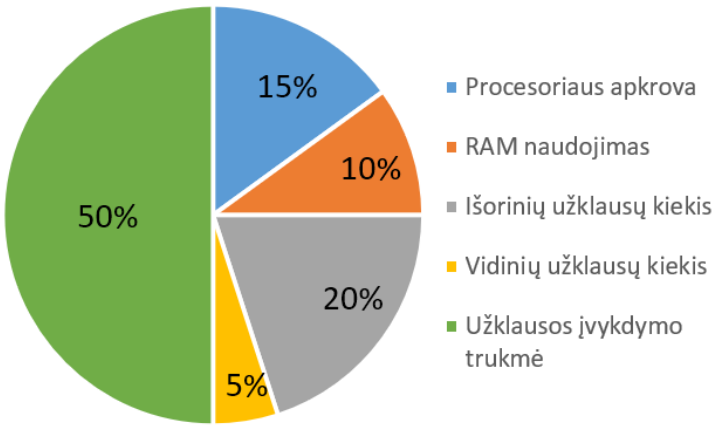
posistemėje siunčia vidines užklausas, kurios reikalauja autorizacijos. Jeigu taip, matomas 31% efektyvumo pagerėjimas, o jeigu ne – 1% suprastėjimas užklausos vykdymo laike. Šie rezultatai koreliuoja ir su darbinės atminties naudojime **pav. 4.8**, kai neoptimizuotas režimas naudoja apytksliai trečdaliu daugiau atminties.



pav. 4.13 Užklausų apdorojimo trukmė užsakymų mikropaslaugos funkcijose

4.5. Prieigos valdymo metodo tyrimo variacijų palyginimas

Poskyryje 4.4 išanalizuoti tyrimo rezultatai yra pateikti įvairiais skerspjūviais. Juose yra matomos įvairios tendencijos, kurios įvardija testuotų prototipo variacijų pranašumus ir trūkumus. Tačiau norint palyginti bendrą režimo efektyvumą, privalu parinkti esminius kriterijus ir juos tarpusavyje palyginti. Todėl šiame poskyryje yra pateikiami apibendrinti įverčiai kiekvienai testo variacijai su tikslu nustatyti, kuri iš jų bendrame kontekste buvo efektyviausia. Šiam tikslui pasiekti yra pasirinkti šie kriterijai kartu su jų svertiniais koeficientais, kurie vaizduojami **pav. 4.14**.



pav. 4.14 Prieigos valdymo metodo efektyvumo kriterijai ir jų svertiniai koeficientai

Pirmieji du kriterijai – procesoriaus ir atminties naudojimas apibūdina aparatinės įrangos naudojimą. Kartu jų įtaka galutiniam įverčiui yra ketvirtadalis, nes programų sistemos efektyvumui įvertinti yra svarbu atsižvelgti ir į fizinių resursų naudojimą. Šiuo atveju procesoriui skiriama 5% daugiau nei operatyviajai atminčiai, nes pastarąją, bendruoju atveju, yra lengviau padidinti. Sekantį ketvirtadalį sudaro bendri užklausų kiekiai. Tačiau vidinėms užklausos skiriami tik 5%. To priežastis – išorinės užklausos atspindi užklausų kiekį, kurį siunčia klientas iš išorinio tinklo ir tai yra vienas iš esminių kriterijų, kuris apibūdina metodo efektyvumą. Tačiau vidinių užklausų kiekis vidiniame tinkle šio prototipu atveju turi mažiau reikšmės, tačiau vis tiek nuspręsta atsižvelgti į šį parametrą, nes kuo daugiau programų sistema generuoja vidinio srauto, tuo mažiau ji yra efektyvesnė. Tačiau svarbiausias kriterijus prototipo efektyvumui nustatyti yra vidutinė užklausų įvykdymo trukmė. Šis kriterijus bene tiksliausiai gali apibūdinti režimo variacijos efektyvumą, nes tai yra vienas iš esminių aspektų iš kliento pusės. Įprastai tai yra esminis faktorius, kuris išorinėms šalims apibūdina žiniatinklio sistemos efektyvumą. Taigi, galutiniam įverčiui aparatinė įranga ir apdorotų užklausų kiekis turi po 25% įtaką, o užklausų įvykdymo trukmė – 50%.

Tokie kokybiniai rodikliai kaip procesoriaus apkrova, operatyviosios atminties naudojimas, užklausų apdorojimo kiekiai ir trukmės yra žymimos ne tik skirtingais vienetais (procentais, gigabaitais, vienetais, milisekundėmis) bet ir skirtingose skalėse. Todėl nuspręsta šias reikšmes normalizuoti į reikšmes, kurių rėžiai yra nuo 0 iki 1. Kadangi šios reikšmės yra sumuojamos į vieną bendrą įvertį, kartu reikia gauti atvirkštinę normalizuotą reikšmę, jei kriterijaus dydis yra atvirkštinai proporcinga režimo variacijos efektyvumui. Pavyzdžiui, kuo daugiau procesoriaus resursų apdorota, tuo režimo efektyvumas yra mažesnis, todėl normalizuotą reikšmę reikia atimti iš 1. Vienintelė išimtis – išorinių užklausų kiekis, nes kuo jis didesnis, tuo daugiau per tą patį laiką su vienodu apkrovos intensyvumu režimas sugebėjo apdoroti užklausų, kas demonstruoja jo efektyvumą. Galiausiai šis skirtumas yra padauginamas iš svertinio koeficiento. Todėl pirmoji normalizavimo formulė kiekvienam kriterijui gali būti išreikšta kaip x_{norm}:

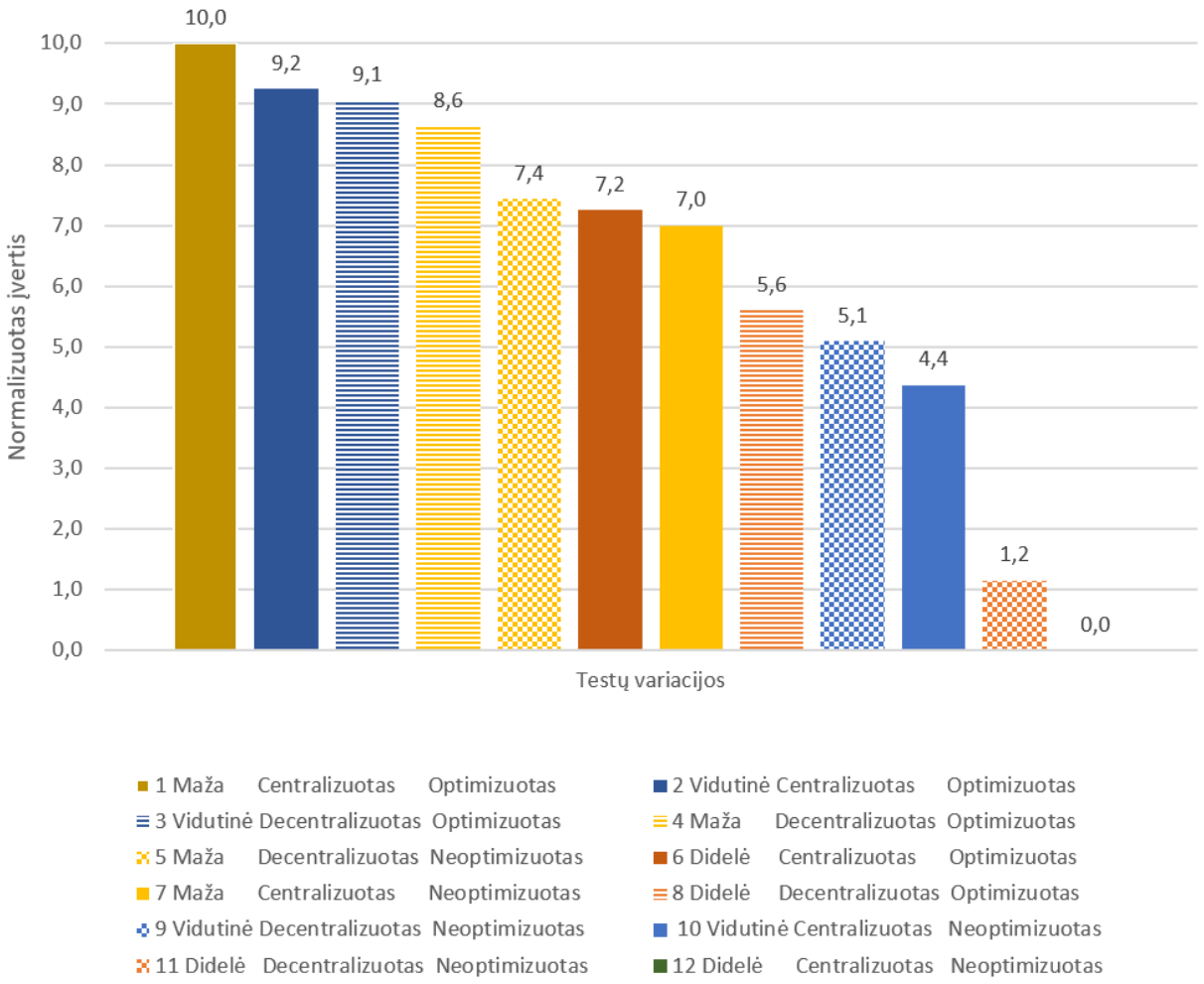
$$x_{norm} = \left(1 - \frac{x_i - \min(\mathrm{x})}{\max(x) - \min(x)}\right) * k$$

Tuomet normalizuotas reikšmes su svertiniais koeficientais reikia susumuoti ir gauti bendrą testo variacijos įvertį su formule:

$$s = \sum^{5} x_{norm}$$

Tačiau gautos aibės rėžiai išsidėstę nuo 0,15 iki 0,8, todėl nuspręsta šią reikšmę dar kartą normalizuoti ir pakeisti skalę iš nuo 0 iki 1 į nuo 0 iki 10. Taip gaunama antroji normalizavimo formulė v_{norm}, kuri normalizuoja reikšmių sumas su normalizuotais svertiniais koeficientais:

$$v_{norm} \frac{s_i - \min(s)}{\max(s) - \min(s)} * 10$$



pav. 4.15 Visų testavimo variacijų įverčių palyginimas

Apskaičiavus galutinius įverčius su aukščiau nurodytomis formulėmis gaunamas rezultatas, kuris vaizduojamas **pav. 4.15**. Pirmoje ir antroje vietoje yra centralizuotas optimizuotas metodas su maža ir didele apkrova, todėl galima teigti, jog vis dėlto tiek centralizacija, tiek optimizacija pasiteisino ir tai yra efektyviausias režimas prieigos valdymo metodui. Svarbu atkreipti dėmesį, kad diagramoje tarpusavy lyginamos visos variacijos kartu, net su skirtingomis apkrovomis. Nepaisant to, šeštoje vietoje yra centrinis optimizuotas režimas, kuris pralenkia kitus režimus su maža, vidutine ir didele apkrovomis. Prasčiausiai pasirodęs režimas yra centralizuotas neoptimizuotas – jis naudoja daug atminties ir rodo apkrovą procesoriui, užklausos buvo vykdomos ilgiausiai bei turėjo daugiausiai vidinių užklausų. Šis metodas po pirminio normalizavimo surinko mažiausią 0,15 įvertį, kuris buvo

normalizuotas į 0. Tai yra atskaitomasis taškas, kuris nurodo prasčiausią pasirodymą beveik visais kriterijais. Taip pat atkreiptinas dėmesys, jog optimizuotas režimas vidutinėje apkrovoje aplenkė neoptimizuotus mažoje apkrovoje. Skirtumas tarp vidutinio decentralizuoto neoptimizuoto ir decentralizuotų neoptimizuotų režimų yra net 4 punktų atotrūkis. Tai reiškia, jog optimizavimas turi nemažą įtaką galutiniam įverčiui. Iš šio grafiko galima drąsiai teigti, kad labiausiai efektyvus režimas yra centralizuotas ir optimizuotas, o mažiausiai – centralizuotas neoptimizuotas.

Tačiau pavienių atvejų analizavimas neatsileidžia bendros tendencijos. Todėl **lentelė 4.1** pateikiami išvestiniai rodikliai, kurie palygina įverčių vidurkius įvairiais skerspjūviais. Pirmasis jų – pagal apkrovą. Skirtumas tarp didelės ir vidutinės apkrovos yra beveik dvigubas (3,5 punkto), tačiau tarp mažos ir vidutinės – vos 1,3. Tai reiškia, jog prieigos valdymo režimo efektyvumas apkrovos atžvilgiu nėra tolygiai pasiskirstęs, ir yra matomas didesnis atotrūkis didelėje apkrovoje. Todėl darytina išvada, jog prieigos valdymo metodas vidutiniškai yra efektyviausias su maža ir vidutine, bet ne didele apkrovomis. Kalbant apie optimizaciją – matomas 100% skirtumas tarp optimizuoto ir neoptimizuoto režimų. Tai reiškia, jog optimizuotas režimais su visomis apkrovomis yra dvigubai efektyvesnis nei neoptimizuotas. Šis požymis išryškina svarbą naudojamų technologijų efektyvume. Sekantis kriterijus yra įverčių vidurkis pagal centralizuotą ir decentralizuotą režimą visose apkrovos. Tačiau skirtumas labai nežymus – centralizuotas metodas vos 2% pranašesnis už decentralizuotą. Tačiau šis vidurkis įtraukia ir neoptimizuotų režimų rezultatus, kurie gali daryti įtaką režimo palyginimui. Todėl išvestas papildomas rodiklis, kuris įtraukia tik optimizuotų režimų įverčius. Šis rodiklis rodo kiek didesnį, 7% bendrą centralizuoto režimo pranašumą.

lentelė 4.1 Greitaveikos testų išvestinių įverčių palyginimas

Įverčių vidurkis pagal apkrovą		
Maža	Vidutinė	Didelė
8,27	6,94	3,50
Įverčių vidurkis pagal optimizaciją		
Optimizuotas	Neoptimizuotas	Skirtumas, %
7,11	3,58	*99*
Įverčių vidurkis pagal režimą		
Decentralizuotas	Centralizuotas	Skirtumas, %
6,16	6,31	*2*
Įverčių vidurkis pagal optimizuotą režimą		
Decentralizuotas	Centralizuotas	Skirtumas, %
7,77	8,28	*7*

Prieigos valdymo metodo mikropaslaugų architektūroje tyrimo išvados

1. Tyrimo duomenims surinkti realizuota žymų analizės logika, kuri padėjo išanalizuoti duomenis įvairiais užklausų kiekių pasiskirstymo ir apdorojimo laikų skerspjūviais. Su tikslu palyginti skirtingų darbo režimų tarpusavio efektyvumą nuspręsta apskaičiuoti išvestinius rodiklius, kurie nusako bendrinį kiekvieno testo įvertį.
2. Išvestiniai įverčiai nurodo, jog prototipas visais darbo režimais veikė su maža ir vidutine apkrovomis efektyviau, nei su didele. Didelės apkrovos efektyvumas yra dvigubai mažesnis, nei vidutinės. Todėl darytina išvada, jog prototipas optimaliausiai apdoroja vidutinį išorinių užklausų srautą.
3. Tarpinių eksperimentų metu pastebėta, jog atvirkštinis tarpinis serveris *Microsoft Service Fabric* platformoje smarkiai lėtina užklausų apdorojimo laikus. Sukūrus ir pritaikius komunikacijos klasterio viduje optimizaciją nustatyta, jog optimizuotas režimas yra dvigubai efektyvesnis bei apdoroja užklausas trečdaliu greičiau, nei neoptimizuotas.
4. Išanalizavus ir apibendrinus visus kriterijus, įskaitant aparatinės įrangos resursų naudojimą, apdorotų užklausų kiekius ir jų trukmes, siūlomas centralizuotas prieigos valdymo režimas yra 2% efektyvesnis už decentralizuotą. Tačiau vertinant tik optimizuotus režimus, centralizuoto režimo efektyvumo skirtumas padidėja iki 7%.
5. Siūlomas centralizuotas režimas rodo geriausius greitaveikos rezultatus tuomet, kai mikropaslauga verslo posistemės ribose siunčia papildomas vidines užklausas. Nustatyta, jog tokiu atveju centralizuotas režimas gali apdoroti užklausas trečdaliu greičiau, nei decentralizuotas. Priešingu atveju, kai mikropaslauga nesiunčia autorizuotų užklausų klasterio viduje, decentralizuotas režimas rodo 1% pranašumą užklausų apdorojimo trukmėje.

Prieigos valdymo metodo mikropaslaugų architektūroje išvados

1. Literatūros šaltinių analizės metu pastebėta, jog esminės problemos, su kuriomis susiduria prieigos valdymo strategijos mikropaslaugų architektūroje yra metodo sudėtingumas, jo greitaveika ir perimetro saugos užtikrinimas. Taip pat tarp skirtingų strategijų įžvelgti tokie panašumai, kaip API vartų infrastruktūrinio komponento naudojimas bei prieigos valdymo informacijos perdavimui naudojami JWT žetonai.
2. Suprojektuotas metodas mikropaslaugų architektūroje sujungia vidinius ir išorinius žetonus prieigos valdymui. Tokia atskirtis suteikia didesnį saugumą ir lengvesnį pritaikymą tiek iš kliento, tiek iš programų sistemos vystytojo pusės. Pritaikytas užklausų valdytojo infrastruktūrinis elementas, kuris, užklausą papildo vidiniu prieigos žetonu, gautu iš autorizavimo mikropaslaugos. Užklausų valdytojas tampa centriniu įeigos į verslo posistemę tašku, o tai leidžia papildomai valdyti užklausas, priklausomai bendros sistemos būklės.
3. Prototipo realizacijai pasirinkta *Microsoft Service Fabric* platforma, kuri suteikia tokias mikropaslaugų savybes kaip horizontalus plečiamumas, izoliacija ir komunikacija. Sukūrus verslo posistemę, tarpinių eksperimentų metu pastebėta, jog šios platformos tarpinis atvirkštinis serveris daro neigiamą įtaką prototipo greitaveikai. Pritaikius optimizaciją nustatyta, jog prieigos valdymo prototipas yra dvigubai efektyvesnis, jeigu nėra naudojami klasteryje esantys atvirkštiniai tarpiniai serveriai.
4. Atlikus prieigos valdymo prototipo greitaveikos analizę nustatyta, jog centralizuotas režimas yra 2% efektyvesnis nei decentralizuotas, o vertinant tik optimizuotus režimus – 7%. Tačiau didžiausias centralizuoto prieigos valdymo režimo greitaveikos privalumas pastebėtas vienu atveju – kai verslo posistemės mikropaslauga siunčia papildomas autorizuotas užklausas klasterio viduje. Tokiu atveju nustatyta, kad centralizuoto režimo užklausų apdorojimo trukmė gali būti net iki 30% mažesnė, nei decentralizuotame režime. Todėl darytina išvada, kad centralizuotas prieigos valdymo mechanizmas yra ypač efektyvus, jei verslo posistemėje yra siunčiamos vidinės autorizuotos užklausos.

Literatūros šaltiniai

[1] W. Hasselbring ir G. Steinacker, „Microservice Architectures for Scalability, Agility and Reliability in E-Commerce,“ *IEEE International Conference on Software Architecture Workshops (ICSAW),* pp. 243-246, 2017.

[2] O. Al-Debagy ir P. Martinek, „A Comparative Review of Microservices and Monolithic Architectures,“ *IEEE 18th International Symposium on Computational Intelligence and,* pp. pp. 149-154, 5 2018.

[3] C. Esposito, A. Castiglione ir K.-K. R. Choo, „Challenges in Delivering Software in the Cloud as Microservices,“ *IEEE Cloud Computing,* t. 3, nr. 5, pp. 10-14, 2016.

[4] T. Yarygina ir A. H. Bagge, „Overcoming Security Challenges in Microservice Architectures,“ *IEEE Symposium on Service-Oriented System Engineering (SOSE,* pp. 11-20, 2018.

[5] R. M. Munaf, J. Ahmed, F. Khakwani ir T. Rana, „Microservices Architecture: Challenges and Proposed Conceptual Design,“ *International Conference on Communication Technologies,* pp. 82-87, 2019.

[6] G. Márquez ir H. Astudillo, „Identifying availability tactics to support security architectural design of microservice-based systems,“ *13th European Conference on Software Architecture,* t. 2, p. 123–129, 2019.

[7] M. Stocker, O. Zimmermann, U. Zdun, D. Lübke ir C. Pautasso, „Interface Quality Patterns,“ *23rd European Conference on Pattern Languages of Programs,* pp. 1-16, 2018.

[8] C. K. Rudrabhatla, „Security Design Patterns in Distributed Microservice Architecture,“ *IJCSIS,* t. 18, nr. 7, pp. 72-75, 8 2020.

[9] A. Pereira-Vale, G. Marquez, H. Astudillo ir E. B. Fernandez, „Security Mechanisms Used in Microservices-Based Systems: A Systematic Mapping,“ *XLV Latin American Computing Conference (CLEI),* pp. 1-10, 2019.

[10] Z. Triartono, R. M. Negara ir Sussi, „Implementation of Role-Based Access Control on OAuth 2.0 as Authentication and Authorization System,“ *Computer Science and Informatics (EECSI),* pp. 259-263, 2019.

[11] E. T. Lodderstedt ir S. M., „OAuth 2.0 Token Revocation,“ Internet Engineering Task Force, 08 2013. [Tinkle]. Available: https://tools.ietf.org/html/rfc7009. [Kreiptasi 18 01 2021].

[12] Internet Engineering Task Force (IETF), „CFRG Elliptic Curve Diffie-Hellman (ECDH) and Signatures,“ Sausis 2017. [Tinkle]. Available: https://datatracker.ietf.org/doc/html/rfc8037. [Kreiptasi 12 Gegužės 2021].

[13] M. B. J. Jones ir N. Sakimura, „JSON Web Token (JWT),“ Internet Engineering Task Force (IETF), gegužė 2015. [Tinkle]. Available: https://datatracker.ietf.org/doc/html/rfc7519. [Kreiptasi 6 birželis 2021].

[14] „Reverse proxy in Azure Service Fabric,“ Microsoft, 06 12 2021. [Tinkle]. Available: https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-reverseproxy. [Kreiptasi 11 05 2022].

1 priedas. Tyrimo metu surinkti duomenys.

Šiame priede pateikiami apibendrinti duomenys, kurie buvo surinkti dvylikos greitaveikos testų vykdymo metu. Šios lentelės yra duomenų šaltinis ketvirtojo skyriaus diagramoms. Prie baigiamojo projekto elektroninės versijos pateikiamas failas pavadinimu „*1 Priedas – Tyrimo metu surinkti duomenys.xlsx*“, kuriame pateikiami visi duomenys. Pirmajame lape pavadinimu „Pagrindinis“ yra pateikiamos išvestinės lentelės pagal kurias nubraižyti grafikai. Jų pirminių duomenų šaltinis yra kiti faile esantys lapai, kuriuose yra išsamūs kiekvieno testo rezultatai.

Užklausų pasiskirstymas mikropaslaugose												
Mikropaslauga	**Maža Dec. Opt.**	**Maža Centr. Opt.**	**Maža Dec. Neopt.**	**Maža Centr. Neopt.**	**Vid. Dec. Opt.**	**Vid. Centr. Opt.**	**Vid. Dec. Neopt.**	**Vid. Centr. Neopt.**	**Did. Dec. Opt.**	**Did. Centr. Opt.**	**Did. Dec. Neopt.**	**Did. Centr. Neopt.**
Užklausų valdytojas	0	43728	0	31200	0	111936	0	67104	0	153048	0	95256
Autentifikavimas	3230	3644	2334	2600	9044	9328	5462	5592	11856	12754	7516	7938
Autorizavimas	64600	40084	46680	28600	180880	102608	109240	61512	237120	140294	150320	87318
Krepšelis	14535	16398	10503	11700	40698	41976	24579	25164	53352	57393	33822	35721
Užsakymai	11305	12754	8169	9100	31654	32648	19117	19572	41496	44639	26306	27783
Prekės	25840	29152	18672	20800	72352	74624	43696	44736	94848	102032	60128	63504
Siuntos	12920	14576	9336	10400	36176	37312	21848	22368	47424	51016	30064	31752
Vid.	18918,6	22905,1	14692,8	18200,0	52972,0	58633,1	31991,7	39144,0	66690,0	80168,0	44022,3	55566,0

Procesoriaus naudojimas												
	Maža Dec. Opt.	**Maža Centr. Opt.**	**Maža Dec. Neopt.**	**Maža Centr. Neopt.**	**Vid. Dec. Opt.**	**Vid. Centr. Opt.**	**Vid. Dec. Neopt.**	**Vid. Centr. Neopt.**	**Did. Dec. Opt.**	**Did. Centr. Opt.**	**Did. Dec. Neopt.**	**Did. Centr. Neopt.**
1	44	33	25	27	67	67	56	54	96	93	87	84
2	50	25	20	28	70	63	54	54	96	96	87	86
3	49	25	16	26	64	67	49	52	92	95	86	83
4	50	27	19	27	63	62	51	54	93	93	89	83
5	49	31	22	24	64	62	53	53	97	91	92	84
6	49	26	23	25	64	63	53	55	94	95	86	81
7	51	26	23	25	66	65	53	54	95	92	87	83
8	49	27	23	25	64	61	55	54	94	92	84	84
9	49	27	23	26	67	64	51	54	94	84	87	84
10	49	26	22	25	64	61	54	55	94	84	85	84
11	49	29	18	26	63	48	50	54	93	87	72	87
12	49	26	25	25	65	55	55	54	96	88	73	85
13	49	25	22	31	66	59	53	53	94	89	70	86
14	49	26	23	25	65	65	50	52	97	89	76	85
15	51	28	23	25	63	57	43	53	94	86	72	81
16	49	25	39	25	65	64	53	53	94	87	70	78
17	59	36	25	24	65	62	52	55	95	85	74	85
18	50	28	23	25	64	63	54	53	95	83	74	85
19	50	26	24	24	67	61	56	52	93	86	76	86
20	44	25	23	25	66	50	55	55	94	80	78	84
Vid	49,4	27,3	23,1	25,6	65,1	60,9	52,4	53,7	94,4	88,8	80,3	83,9

Ram naudojimas												
Tyrimo laikas	**Maža Dec. Opt.**	**Maža Centr. Opt.**	**Maža Dec. Neopt.**	**Maža Centr. Neopt.**	**Vid. Dec. Opt.**	**Vid. Centr. Opt.**	**Vid. Dec. Neopt.**	**Vid. Centr. Neopt.**	**Did. Dec. Opt.**	**Did. Centr. Opt.**	**Did. Dec. Neopt.**	**Did. Centr. Neopt.**
1	13,9	13,3	14,2	13,8	12,5	13,1	13,8	13,7	12,7	12,7	14,3	13,7
2	12,6	12,5	14,0	13,8	12,2	12,4	14,2	14,2	12,4	12,5	14,8	14,3
3	12,2	12,4	14,0	13,9	12,1	12,5	14,5	14,4	12,6	12,5	15,0	14,5
4	12,2	12,4	14,1	14,3	12,2	12,6	14,5	14,6	12,5	12,7	15,1	14,6
5	12,1	11,5	14,4	14,3	12,3	12,7	14,5	14,0	12,5	12,6	15,0	14,6
6	12,1	11,5	14,5	14,3	12,3	12,5	14,4	13,5	12,4	12,5	13,9	14,8
7	11,3	11,5	14,4	14,4	12,2	12,6	14,6	13,7	12,4	12,6	14,2	14,7
8	11,1	11,5	14,4	14,4	12,1	12,6	14,5	13,7	11,7	12,1	14,2	14,8
9	11,1	11,6	14,5	14,4	11,9	12,6	14,5	13,8	11,4	11,7	14,3	14,8
10	11,1	11,5	14,6	14,4	11,2	12,6	13,5	13,8	11,4	11,8	14,4	14,8
11	11,1	11,5	14,6	13,4	11,2	12,4	13,6	13,8	11,5	11,7	14,4	13,8
12	11,1	11,5	13,6	13,4	11,2	12,4	13,7	13,7	11,4	11,7	14,3	13,7
13	11,1	11,6	13,3	13,5	11,2	11,7	13,6	13,7	11,4	11,7	14,2	13,9
14	11,1	11,6	13,4	13,6	11,2	11,5	13,6	13,7	11,4	11,7	14,1	14,0
15	11,1	11,6	13,5	13,7	11,2	11,5	13,8	13,7	11,4	11,7	14,2	14,1
16	11,1	11,6	13,5	13,8	11,2	11,5	13,7	13,8	11,4	11,7	14,1	14,1
17	11,1	11,6	13,6	13,8	11,2	11,4	13,7	13,8	11,4	11,7	14,2	14,0
18	11,2	11,6	13,7	13,8	11,3	11,4	13,7	13,7	11,4	11,6	14,3	14,1
19	11,1	11,7	13,7	13,8	11,3	11,5	13,6	13,7	11,3	11,5	14,0	14,1
20	11,1	11,6	13,7	13,7	11,1	11,5	13,5	13,6	11,5	11,6	13,8	14,1
Vid.	11,5	11,8	14,0	13,9	11,6	12,2	14,0	13,8	11,8	12,0	14,3	14,3

Bendri užklausų kiekiai												
	Maža Dec. Neopt.	**Maža Centr. Neopt.**	**Maža Dec. Opt.**	**Maža Centr. Opt.**	**Vid. Dec. Neopt.**	**Vid. Centr. Neopt.**	**Vid. Dec. Opt.**	**Vid. Centr. Opt.**	**Did. Dec. Neopt.**	**Did. Centr. Neopt.**	**Did. Dec. Opt.**	**Did. Centr. Opt.**
Išorinės užklausos	28008	31200	38760	43728	65544	67104	108528	111936	90192	95256	142272	153048
Vidinės užklausos	43179	79300	59755	111142	101047	170556	167314	284504	139046	242109	219336	388997
Viso	71187	110500	98515	154870	166591	237660	275842	396440	229238	337365	361608	542045

Užklausų pasiskirstymas mazguose												
	Maža Dec. Neopt.	**Maža Centr. Neopt.**	**Maža Dec. Opt.**	**Maža Centr. Opt.**	**Vid. Dec. Neopt.**	**Vid. Centr. Neopt.**	**Did. Dec. Neopt.**	**Did. Centr. Neopt.**	**Vid. Dec. Opt.**	**Vid. Centr. Opt.**	**Did. Dec. Opt.**	**Did. Centr. Opt.**
1 Mazgas	19138	22874	26486	32066	44798	49201	61628	69852	74165	82068	97222	112235
2 Mazgas	19150	22885	26486	32075	44798	49213	61638	69856	74158	82065	97215	112202
3 Mazgas	19132	22880	26484	32065	44778	49211	61630	69859	74159	82156	97220	112234
4 Mazgas	19140	22886	26484	32067	44790	49217	61635	69853	74163	82041	97221	112273
5 Mazgas	19134	22875	26490	32063	44778	49206	61625	69852	74159	82102	97218	112232

Išorinių užklausų įvykdymo laikai												
Funkcija	**Maža Dec. Opt.**	**Maža Centr. Opt.**	**Maža Dec. Neopt.**	**Maža Centr. Neopt.**	**Vid. Dec. Opt.**	**Vid. Centr. Opt.**	**Vid. Dec. Neopt.**	**Vid. Centr. Neopt.**	**Did. Dec. Opt.**	**Did. Centr. Opt.**	**Did. Dec. Neopt.**	**Did. Centr. Neopt.**
Prisijungti	100,4	94,2	4,0	54,0	76,5	84,8	4,9	67,9	102,3	96,2	8,4	50,9
Atnaujinti žetoną	1,3	2,0	1,5	5,9	1,9	2,6	2,4	10,3	7,1	9,0	5,4	19,1
Gauti prekę	13,4	14,2	15,5	20,7	16,1	16,1	19,7	28,3	23,5	22,6	27,2	39,1
Gauti prekes	12,0	12,4	13,5	20,1	13,4	13,0	18,0	29,5	20,1	18,5	25,1	42,5
Sukurti prekę	11,3	10,6	13,3	17,4	13,0	13,9	18,0	24,9	19,3	18,3	25,5	38,9
Atnaujinti prekę	24,1	19,9	29,5	27,2	27,2	24,0	41,3	36,2	41,8	32,8	57,1	52,5
Ištrinti prekę	24,4	20,1	30,0	27,1	27,7	24,5	41,3	36,1	41,9	33,3	57,0	50,5
Gauti krepšelį	25,6	22,1	35,1	33,0	29,4	26,1	47,7	46,1	44,5	35,3	66,1	65,8
Sukurti krepšelį	24,8	21,1	31,4	31,4	29,8	25,2	43,9	43,2	43,4	34,1	61,5	60,9
Pridėti prekę į krepšelį	11,3	10,5	13,7	16,0	13,7	13,3	18,0	22,7	20,1	19,8	25,5	34,5
Ištrinti krepšelio prekę	11,6	10,8	13,6	15,7	13,5	12,9	17,8	21,9	20,0	18,8	25,1	32,3
Ištrinti krepšelį	11,3	10,8	13,8	15,9	13,3	13,0	18,1	22,9	19,8	18,7	24,9	32,0
Gauti siuntą	12,5	11,5	14,3	18,3	13,5	13,2	18,1	25,4	20,1	19,5	25,3	37,1
Gauti siuntas	11,7	12,3	14,4	18,5	13,3	13,2	18,3	24,9	20,0	19,0	25,4	37,5
Sukurti siuntą	26,1	21,3	35,1	33,2	29,8	26,2	47,7	45,6	45,2	36,0	66,9	68,1
Atnaujinti siuntą	25,1	20,3	33,2	30,9	29,4	25,4	43,9	42,7	44,1	34,7	62,1	63,5
Ištrinti siuntą	25,8	22,4	35,4	30,9	30,1	26,5	47,1	43,9	44,8	35,8	66,8	60,8
Gauti užsakymą	11,8	11,4	14,4	18,2	14,2	13,5	18,1	25,2	20,8	19,7	25,3	39,6
Gauti užsakymus	15,1	14,3	17,4	24,6	17,3	17,3	21,6	34,8	23,7	24,1	28,7	61,2
Sukurti užsakymą	67,0	51,7	90,3	73,8	76,1	62,9	125,9	99,3	120,6	80,8	174,9	136,6
Ištrinti užsakymą	11,7	11,5	14,1	17,6	13,5	14,6	18,0	22,8	20,5	21,0	25,1	32,7
Pateikti užsakymo siuntą	38,8	30,3	52,7	44,7	45,2	37,9	71,4	61,6	69,1	50,3	99,7	87,4
Pateikti užsakymo mokėjimą	39,8	31,0	52,5	44,9	44,4	37,7	71,0	62,2	68,5	50,0	98,5	88,5

Pateikti užsakymą	53,4	42,3	73,0	61,4	60,3	50,8	100,3	82,0	93,5	66,5	140,0	116,3
Vid.Vykd. laikas	**25,4**	**22,0**	**27,6**	**29,2**	**27,6**	**25,4**	**37,2**	**40,0**	**41,4**	**34,0**	**52,0**	**56,2**

Apibendrinti tyrimo variacijų įverčiai												
Kriterijus	**4 Maža Dec. Opt.**	**1 Maža Centr. Opt.**	**5 Maža Dec. Neopt.**	**7 Maža Centr. Neopt.**	**3 Vid. Dec. Opt.**	**2 Vid. Centr. Opt.**	**9 Vid. Dec. Neopt.**	**10 Vid. Centr. Neopt.**	**8 Did. Dec. Opt.**	**6 Did. Centr. Opt.**	**11 Did. Dec. Neopt.**	**12 Did. Centr. Neopt.**
CPU	0,09	0,14	0,15	0,14	0,06	0,07	0,09	0,09	0,00	0,01	0,03	0,02
RAM	0,10	0,09	0,01	0,01	0,10	0,08	0,01	0,02	0,09	0,08	0,00	0,00
Išor. Užkl.	0,02	0,03	0,00	0,01	0,13	0,13	0,06	0,06	0,18	0,20	0,10	0,11
Vid. Užkl.	0,05	0,04	0,05	0,04	0,03	0,02	0,04	0,03	0,02	0,00	0,04	0,02
Užkl. Trukmė	0,45	0,50	0,42	0,39	0,42	0,45	0,28	0,24	0,22	0,33	0,06	0,00
Suma su svertiniu koeficientu	0,71	0,80	0,63	0,60	0,74	0,75	0,48	0,43	0,51	0,62	0,23	0,15
	Min:	**0,15**	**Max:**	**0,80**								
Normalizuotas įvertis	8,6	10,0	7,4	7,0	9,1	9,2	5,1	4,4	5,6	7,2	1,2	0,0