**VILNIAUS UNIVERSITETAS**
**ŠIAULIŲ AKADEMIJA**

PROGRAMŲ SISTEMOS BAKALAURO STUDIJŲ PROGRAMA

E. paslaugų sistemos specializacija

**NEDAS SERPAUSKAS**

**Bakalauro darbas**

# MIKROSERVISŲ ARCHITEKTŪROS MODELIS ŽINIATINKLIO PROGRAMAI KURTI

Darbo vadovas (-ė): doc. dr. Sigita Turskienė

Šiauliai, 2025

**Studijuojančiojo, teikiančio baigiamąjį darbą, GARANTIJA**

***WARRANTY of Final Thesis***

| | |
|---|---|
| Vardas, pavardė<br>*Name, Surname* | **Nedas Serpauskas** |
| Padalinys<br>*Faculty* | **Šiaulių akademija**<br>***Šiauliai Academy*** |
| Studijų programa<br>*Study Programme* | **Programų sistemos**<br>***Software engineering*** |
| Darbo pavadinimas<br>*Thesis topic* | **Mikroservisų architektūros modelis žiniatinklio programai kurti**<br>***Microservices Architecture Model for Web Application Development*** |
| Darbo tipas<br>*Thesis type* | **Baigiamasis darbas**<br>***Final Thesis*** |

Garantuoju, kad mano baigiamasis darbas yra parengtas sąžiningai ir savarankiškai, kitų asmenų indėlio į parengtą darbą nėra. Jokių neteisėtų mokėjimų už šį darbą niekam nesu mokėjęs.
Šiame darbe tiesiogiai ar netiesiogiai panaudotos kitų šaltinių citatos yra pažymėtos literatūros nuorodose.

*I guarantee that my thesis is prepared in good faith and independently, there is no contribution to this work from other individuals. I have not made any illegal payments related to this work.*
*Quotes from other sources directly or indirectly used in this thesis, are indicated in literature references.*

**Aš, Nedas Serpauskas, pateikdamas (-a) šį darbą, patvirtinu** (*pažymėti*) ☒

**Embargo laikotarpis**
***Embargo Period***

Prašau nustatyti šiam baigiamajam darbui toliau nurodytos trukmės embargo laikotarpį:
*I am requesting an embargo of this thesis for the period indicated below:*

☐ _____ mėnesių / *months*
(embargo laikotarpis negali viršyti 60 mėn. / *an embargo period shall not exceed 60 months*).

☒ Embargo laikotarpis nereikalingas / *no embargo requested.*

Embargo laikotarpio nustatymo priežastis / *Reason for embargo period*:

# TURINYS

# SANTRAUKA

**Mikroservisų architektūros modelis žiniatinklio programai kurti**

Šiame baigiamajame darbe yra nagrinėjamos programinėje įrangoje naudojamų architektūrų problemos. Programinėje įrangoje naudojamų architektūrų problemoms nustatyti baigiamajame darbe yra nagrinėjama monolitinė architektūra bei į servisus orientuota architektūra, o joms spręsti buvo pasirinkta mikroservisų architektūra. Atsižvelgiant į aktualias mikroservisų architektūros problemas, sukurtas mikroservisų architektūros modelis. Pagal suprojektuotą modelį sukurta žiniatinklio programa, tenkinanti reikalavimus, formuluojamus pagal konkrečios įmonės poreikius.



# SUMMARY

**Microservices architecture model for web application development**

In this final thesis the issues of architectures used in software are examined. In order to define the problems of architectures used in software, monolithic architecture and service-oriented architecture are examined and to solve these problems, microservices architecture is chosen. Taking into consideration the current problems of microservices architecture, a microservices architecture model was developed. According to the designed model, a web application that meets the requirements formulated according to the needs of specific company was developed.

# ĮVADAS

Pagal šaltinį [11], dalis žiniatinklio programų kūrėjų susiduria su programinių produktų žlugimu ir tą įtakoja ne naudojamos priemonės realizacijai, o netinkamai apgalvotas žiniatinklio programų planavimo procesas ir ne pilnai atlikta kuriamo produkto analizė. Kai kuriose organizacijose, naudojančiose žiniatinklio programas verslo poreikiams įgyvendinti, nėra apgalvojama, kaip naudojama žiniatinklio programa įtakoja verslo procesų veikimą ir kaip reikia valdyti šių programų kūrimo procesą [11]. Kuriant žiniatinklio programas, taip pat yra svarbu atsižvelgti į jų tobulinimo ir plėtros poreikį ateityje, jų sudėtingumą [11]. Sėkmingam žiniatinklio programų kūrimui yra būtina planuoti kūrimo procesą, apgalvoti, kokius procesus žiniatinklio programa turi įgyvendinti, kaip ji turi būti struktūrizuojama, taip pat yra svarbu apgalvoti galimus iššūkius ateityje ir kaip jų būtų galima išvengti [11].

Šaltinyje [24] yra akcentuojama programinėje įrangoje naudojamos architektūros netinkamumo problema. Dėl rinkos konkurencingumo ir siekio įgyvendinti verslo procesus, atsiranda tobulinimo ir plėtros poreikis, o netinkamas architektūros naudojimas gali sukelti naujų iššūkių šių poreikių įgyvendinimui [24]. Architektūros netinkamumo problemai spręsti neretai yra renkamasi keisti programinės įrangos architektūrą, tačiau tam, jog pasirinkta architektūra būtų tinkama atitinkamiems poreikiams, ji turi turėti ryšį su įmonės vykdomais procesais ir jos aplinka [24].

Pagal šaltinį [23], programinėje įrangoje naudojamų architektūrų problemoms spręsti naudojama mikroservisų architektūra. Remiantis [15] šaltinio duomenimis, mikroservisų architektūrą pradėjo naudoti vienos iš žinomiausių pasaulinių įmonių, tokių kaip: Netflix, Spotify ir Uber. Šias įmones naudojamą architektūrą pakeisti į mikroservisų architektūrą paskatino plėtros poreikis dėl didėjančios teikiamų paslaugų paklausos [15]. Naudojamos architektūros pakeitimas į mikroservisų architektūrą šioms įmonėms suteikė privalumų, tokių kaip: programinių klaidų toleravimas, patikimumas, lankstumas bei kokybiškesnis paslaugų teikimas [15]. Šaltinyje [9] yra teigiama, jog vis dar yra neapibrėžtų šios architektūros problemų ir tobulinimo galimybių, o tai apsunkina šios architektūros panaudojimą praktikoje. Tikslingai panaudoti mikroservisų architektūrą vis dar gali būti sudėtinga, kadangi trūksta informacinių šaltinių, aprėpiančių visas galimas problemas ir kaip šios problemos gali būti sprendžiamos [23].

Atsižvelgiant į šaltiniuose pateikiamą informaciją galima teigti, jog viena iš priežasčių, kodėl yra susiduriama su programinės įrangos architektūros netinkamo pasirinkimo problema, yra programinių produktų kūrimo eigos ir tolimesnio verslo vystymo planavimo stoka. Naudojantis šaltinių, kuriuose yra akcentuojama ši problema, informacija, galima daryti prielaidą, jog programinės įrangos architektūra yra skirtinga priklausomai nuo to, koks programinis produktas yra

kuriamas ir kokius keliamus tikslus jis įgyvendina. Atsižvelgiant į tai, kuriamos programinės įrangos architektūros problemoms nustatyti yra reikalingas architektūros praktinis pritaikymas.

Nagrinėtuose literatūriniuose šaltiniuose apibūdinamoms programinės įrangos problemoms spręsti baigiamajame darbe yra pasirenkama mikroservisų architektūra, kuri yra panaudojama žiniatinklio programos komponentams ir sąveikai tarp jų suformuoti. Šį pasirinkimą paskatino didėjantis šios architektūros naudojimo populiarumas ir informacijos, kaip šią architektūrą naudoti siekiant spręsti aktualias programinės įrangos architektūros problemas, trūkumas.

**Baigiamojo darbo tikslas** – Sukurti žiniatinklio programą, kuri remiasi mikroservisų architektūros modeliu.

**Baigiamojo darbo uždaviniai**:

1. Naudojantis literatūriniuose šaltiniuose pateikta informacija, nustatyti, kokias programinės įrangos architektūros problemas sprendžia mikroservisų architektūra.
2. Sukurti mikroservisų architektūros modelį.
3. Sukurti žiniatinklio programos komponentus, tenkinančius mikroservisų architektūros modelį, ir juos apjungti į vientisą žiniatinklio programą.
4. Atlikti reikalingus sukurtos žiniatinklio programos testavimus pagal apibrėžtus testavimo atvejus ir įvertinti, kokias mikroservisų architektūros problemas pavyko išspręsti.

# 1. PROGRAMINĖS ĮRANGOS ARCHITEKTŪRA IR MIKROSERVISŲ ARCHITEKTŪROS MODELIS

## 1.1. Vizualinės perspektyvos žiniatinklio programų architektūrai įvertinti

Žiniatinklio programos kūrimo procese yra ypač svarbu tinkamai pasirinkti, kokia architektūra turi būti naudojama. Siekiant apibrėžti, kaip yra apibūdinama žiniatinklio programų architektūra, nagrinėjamas šaltinis [6]. Norint išanalizuoti žiniatinklio programų architektūrą ir įvertinti visus jos aspektus, vienos vizualinės perspektyvos neužtenka [6]. Šaltinyje [6] yra teigiama, jog žiniatinklio programos architektūrą galima suvokti ir suprasti atlikus analizę iš šių perspektyvų:

1. **Dizaino**. Šioje perspektyvoje yra analizuojami žiniatinklio programos reikalavimai ir yra apibrėžiamos problemos.
2. **Procesų**. Procesų perspektyvoje galima išanalizuoti programos veikimo procesus ir jų charakteristikas, taip pat įžvelgti kuriamos sistemos našumą.
3. **Įgyvendinimo**. Šioje perspektyvoje yra analizuojami žiniatinklio programos tarpusavyje sąveikaujantys komponentai.
4. **Panaudos atvejų**. Panaudos atvejų perspektyvos pagalba galima įžvelgti žiniatinklio programos naudotojų sąveiką su pačia programa.
5. **Diegimo**. Šioje perspektyvoje yra vizualizuojami fiziniai mazgai, kuriuose yra diegiami komponentai, ir jų tarpusavio sąveika.

Žiniatinklio programos architektūros esminiams aspektams ir kylančioms problemoms įvertinti yra reikalingos įvairaus tipo požiūrio sritys, orientuotus į skirtingus aspektus. Atsižvelgiant į tai, mikroservisų modeliui kurti yra naudojamos šiame skyrelyje apibrėžtos vizualinės perspektyvos.

## 1.2. Programinės įrangos architektūrų tipai

Atsižvelgiant į tai, jog programinės įrangos architektūros gali būti skirtingos priklausomai nuo jos panaudojimo, vienos architektūros analizės, siekiant apibrėžti jose egzistuojančias problemas, neužtenka. Šaltinyje [24] yra išskiriami trys naudojamų programinės įrangos architektūrų tipai: monolitinė architektūra, į servisus orientuota architektūra ir mikroservisų architektūra. Šiame skyrelyje, naudojantis literatūrinių šaltinių informacija, yra detalizuojamos anksčiau minėtų programinės įrangos architektūrų charakteristikos, šių architektūrų naudojimo privalumai ir kylančios problemos.

**Monolitinė (*angl. Monolithic*) architektūra.** Programos, kuriose yra taikoma monolitinė architektūra, yra vientisos, tai yra: visos egzistuojančios programos funkcionalumo elementai yra talpinami toje pačioje programoje [12].

Šaltinyje [3] yra išskiriami šios architektūros privalumai: monolitinėje programoje yra atliekama komunikacija tarp vidinių jos procesų, todėl informacijos apsikeitimas vyksta greičiau, taip pat tokią programą yra lengviau stebėti, taisyti, kuomet ji nėra pakankamai sudėtinga.

Tobulinamos monolitinės programos sudėtingėja, o dėl to taip pat tampa sudėtinga diegti naujoves arba tam tikrų veikiančių servisų pakeitimus, nes jų diegimas gali sąlygoti programos dalies neveiksnumą [3]. Programose, kuriose yra naudojama ši architektūra, funkciniai elementai yra vienas nuo kito priklausomi, todėl monolitinę programą laikui bėgant tampa vis sunkiau valdyti didėjant jos apimčiai [20].

**Į servisus orientuota architektūra (*Angl. Service-Oriented Architecture*).** Šaltinyje [2] į servisus orientuota architektūra (*Angl. trumpinys – SOA*) yra apibrėžiama taip: „Į servisus orientuota architektūra (SOA) – tai architektūros stilius, kuris nustato gaires, kaip servisai yra apibūdinami, surandami ir naudojami. Šios architektūros tikslas yra nustatyti programos kūrimo reikalavimus paskirstytos informacijos sistemoms, kurios yra laisvai susietos ir potencialiai nevienalytės“. Anot šaltinio [30], naudojant šią architektūrą, programa yra suskaidoma į atskirus komponentus, galinčius komunikuoti tarpusavyje. Šaltinyje [10] į servisus orientuota architektūra yra palyginama su mikroservisų architektūra ir yra išryškinamos į servisus orientuotos architektūros charakteristikos: servisai gali būti skirtingos apimties ir jiems nėra būdingas duomenų valdymo atskyrimas.

Vienas iš šios architektūros privalumų yra tai, jog servisai gali laisvai migruoti ir į juos gali būti kreipiamasi pagal tam tikrus kriterijus neatsižvelgiant tai, kur šis servisas yra talpinamas, tačiau šio privalumo įgyvendinimui yra reikalingas registras, kuriame yra registruojami servisai [30]. Šaltinyje [30] taip pat yra išskiriami ir kiti šios architektūros privalumai: daugialypio servisų panaudojimo ir pritaikymo galimybė, taip pat programos, kurioje yra taikoma į servisus orientuota architektūra, plėtros galimybė.

Šaltinyje [4] yra akcentuojama viena iš pagrindinių šios architektūros problemų – serviso keitimo problema. Naudojant šią architektūrą atsiranda centralizuoto valdymo ir sąsajos, kuria naudojantis galima pasiekti servisus, poreikis ir šiam poreikiui patenkinti yra realizuojami komponentai, atsakingi už servisų valdymą ir jų pasiekiamumą [4]. Keičiant tam tikrą servisą, kuris įtakoja šiuos komponentus, tuo pačiu gali prireikti perdiegti ir kitus servisus [4]. Dėl šios problemos serviso keitimas gali įtakoti visos programos veikimą [4].

**Mikroservisų (*angl. Microservices*) architektūra.** Šaltinyje [20] mikroservisų architektūra (*Angl. trumpinys – MSA*) yra apibrėžiama taip: „Mikroservisų architektūros (MSA) stilius – tai būdas sukurti vieną programą kaip mažų servisų rinkinį, kuriame kiekvienas servisas veikia atskirame procese ir tarpusavyje komunikuoja naudojant nesudėtingus mechanizmus (pavyzdžiui: HTTP API).“. Programą, kurioje yra taikoma mikroservisų architektūra, yra lankstesnė,

kadangi visi programos komponentai yra dalinami į atskirus, mažos apimties ir tarpusavyje laisvai susietus komponentus, kurie gali būti atskirai diegiami [20]. Trikdžių atveju, komponento neveiksnumas neįtakoja visos programos griūties [20].

Vienas iš mikroservisų architektūros privalumų yra galimybė stebėti kiekvieno mikroserviso apkrovą ir naudojamų resursų kiekį ir atsižvelgiant į tai bei esant poreikiui, kurti jų identiškas kopijas (*angl. replicate*) ir paskirstyti tiek vidinių, tiek išorinių užklausų apkrovą į kelis mikroservisus arba šias kopijas sunaikinti, kuomet jos yra nereikalingos [8]. Kitas privalumas, minimas šaltinyje [8], yra mikroservisų elastiškumas, dėl kurio yra tinkamiau išnaudojami resursai. Priklausomai nuo mikroservisų panaudojamų resursų, jie gali būti lengvai perkeliami į kitas sistemas arba plečiami atsižvelgiant į tai, ar yra užfiksuojamas resursų trūkumas, ar turimų resursų yra per daug ir jie nebus išnaudojami [8].

Mikroservisų architektūros trūkumai yra identifikuojami šaltinyje [7] aprašytame tyrime. Šio tyrimo metu buvo identifikuotas didžiausias trūkumas – bendros mikroservisų sistemos veikimo valdymo sudėtingumas, kadangi atskirų mikroservisų funkcionalumo apjungimui į vieną veikiančią sistemą didėja stebėjimo bei mikroservisų valdymo poreikis [7]. Taip pat šaltinyje [7] yra išskiriami ir kiti svarbūs šios architektūros problemos: duomenų vientisumo problema, kuomet atskiri mikroservisai gali valdyti tik tuos duomenis, kurie yra reikalingi jų funkcionalumui bei komunikacijos tarp mikroservisų problema, kuomet dėl komunikacijos dažnumo gali suprastėti programos, kurioje naudojama mikroservisų architektūra, kokybė, taip pat saugumo užtikrinimo ir programinės įrangos testavimo sudėtingumas [7].

Apžvelgiant iš literatūrinių šaltinių surinktą informaciją apie tris naudojamas programinės įrangos architektūras galima pasakyti, jog visos nagrinėtos architektūros turi joms būdingus privalumus, tačiau tuo pačiu egzistuoja ir kiekvienai architektūrai būdingos problemos bei iššūkiai. Galima daryti prielaidą, jog programinės įrangos architektūrų tipų įvairovę įtakoja aktualios problemos, su kuriomis susiduria programinės įrangos kūrėjai, o atitinkamos architektūros pasirinkimas padeda išspręsti jau egzistuojančias problemas. Nors šiame skyrelyje minėtos problemos yra išsprendžiamos pasirinkus atitinkamą architektūrą, atsiranda naujų, nepriklausomai nuo architektūros programinėje įrangoje praktinio pritaikymo, bendrai architektūrai būdingų iššūkių ir problemų, kurias programinės įrangos kūrėjai turi įvertinti ir pritaikyti papildomas priemones šioms problemoms spręsti.

### 1.3. Mikroservisų architektūros modelis. Mikroservisų architektūros problemų apibrėžimas

Mikroservisų architektūros problemoms įžvelgti ir jų sprendimo būdams rasti, yra reikalingas mikroservisų architektūros modelis, aprėpiantis šias problemas. Šaltinyje [21] yra išskiriamos šios sąvokos, apibrėžiančios bendro mikroservisų architektūros modelį:

1. Atskirai įdiegti vienetai.
2. Servisų komponentai, turintys skirtingą detalumą.
3. Išskaidyta architektūra.

Dėl programinės įrangos, kurioje yra naudojama mikroservisų architektūra, įvairovės, sukurti modelį, atvaizduojantį visas architektūros problemas yra sudėtinga. Nors bendrasis mikroservisų architektūros modelis gali atvaizduoti bendras šios architektūros problemas, praktiniame pritaikyme šio modelio neužtenka. Siekiant nustatyti, kokie modeliai yra naudojami praktikoje, nagrinėjamas šaltinis [16]. Šaltinyje [16] yra aprašomas atliktas tyrimas, kurio metu buvo analizuojami moksliniai ir pramoniniai šaltiniai ir pagal surinktą informaciją yra išskiriami architektūros modeliai, sprendžiantys skirtingas mikroservisų architektūros problemas.

Dėl tyrime užfiksuotų architektūros modelių, sprendžiančių skirtingas problemas, gausos, mikroservisų modeliui kurti buvo pasirinkti šie architektūros modeliai:

1. API užklausų vartų modelis. Pagal šaltinį [16], šiame modelyje yra apibrėžiama komunikacijos tarp išorinių klientų ir programos servisų problema.
2. Servisų registro modelis.
3. Servisų atpažinimo modelis.
4. Stebėjimo (*Angl. Monitoring*) modelis.
5. Rezultatų saugyklos modelis.
6. Duomenų bazė yra servisas (*Angl. Database is the service*) modelis. Pagal šaltinį [16], šiame modelyje yra vaizduojama duomenų bazės panaudojimo problema mikroservisų architektūroje.
7. Konteinerio modelis. Šiame modelyje yra apibrėžiama mikroservisų diegimo problema.

Nors šaltinyje [16] aprašytame tyrime yra išskiriami mikroservisų modeliai praktiniam pritaikymui, orientuoti į diegimo, komunikacijos, palaikymo ir valdymo problemas, tačiau jame nėra apibrėžiamas joks modelis, orientuotas į saugumo užtikrinimo naudojant mikroservisų architektūros problemas. Remiantis šaltinyje [22] pateikiamomis saugumo problemomis ir jų sprendimo būdais, galima suformuluoti šiuos modelius, kurie yra naudojami šiame baigiamajame darbe:

1. Autentifikacijos ir autorizacijos modelis. Šio modelio pagalba yra sprendžiama išorinės komunikacijos ir vidinės komunikacijos saugumo problemos.

2. Saugumo filtrų modelis. Šio modelio pagalba yra sprendžiama programoje veikiančių servisų pasiekiamumo problema.

Atsižvelgiant į šiame skyrelyje nagrinėtuose šaltiniuose pateiktą informaciją galima pasakyti, jog mikroservisų architektūros problemoms įžvelgti yra reikalingi specializuoti modeliai, aprėpiantys tiek bendrai mikroservisų architektūrai, tiek praktiniame pritaikyme būdingas problemas. Iš to seka, jog mikroservisų modelis skirtas konkrečiai programinei įrangai kurti, yra sudarytas iš kelių modelių, priklausomai nuo realizacijos pobūdžio. Šiame baigiamajame darbe mikroservisų modelis yra sudaromas iš bendro mikroservisų modelio ir devynių specializuotų modelių, skirtų žiniatinklio programos, kurioje yra naudojama mikroservisų architektūra, problemoms įžvelgti.

# 2. ŽINIATINKLIO PROGRAMOS, PAREMTOS MIKROSERVISŲ ARCHITEKTŪROS MODELIU, PROJEKTAVIMAS

## 2.1. Žiniatinklio programos kūrimo etapai

1. Nustatyti žiniatinklio programos reikalavimus.
2. Apibrėžti mikroservisų modelį kuriamai žiniatinklio programai, remiantis nustatytais žiniatinklio programos reikalavimais.
3. Pasirinkti tinkamas priemones ir įrankius žiniatinklio programai kurti ir jos keliamiems reikalavimams įgyvendinti.
4. Pasitelkiant pasirinktas realizacijos priemones ir įrankius, sukurti programinius komponentus, įgyvendinančius keliamus reikalavimus reikalavimų specifikacijoje.
5. Sujungti atskirus programinius komponentus į vientisą žiniatinklio programą ir atlikti šios programos testavimus remiantis sukurtu testavimo planu.
6. Apibendrinti visą kūrimo procesą, įžvelgti galimus trūkumus ir nustatyti tobulinimo galimybes.

## 2.2. Mikroservisų modelio realizacija. Žiniatinklio programos reikalavimų specifikacija

Tam, jog būtų galima spręsti mikroservisų architektūros problemas, yra kuriama žiniatinklio programa, kuriai yra keliami reikalavimai pagal konkrečios įmonės poreikius (*Žr. priedą Nr.1*).

Mikroservisų architektūros problemoms atvaizduoti grafiškai yra naudojamas mikroservisų modelis, apibrėžiantis mikroservisų architektūros problemas, būdingas tiek mikroservisų architektūrai nepriklausomai nuo jos realizacijos, tiek problemas, kurios kyla realizuojant būtent šią žiniatinklio programą. Mikroservisų modelis yra kuriamas apjungiant 1.3 skyrelyje apibrėžtus modelius, o jis yra atvaizduojamas pasitelkiant tris skirtingas perspektyvas:

1. Panaudos atvejų (*Žr. priedą Nr.1 1 pav. ir 2 pav.*). Ši perspektyva yra atvaizduojama naudojant panaudos atvejų diagramas. Panaudos atvejų perspektyvoje yra vaizduojamas stebėjimo panaudos atvejai, priklausantys stebėjimo modeliui, autentifikacijos ir autorizacijos panaudos atvejai, priklausantys autentifikacijos ir autorizacijos modeliui, taip pat yra apibrėžiami specializuoti panaudos atvejai, kuriuos gali atlikti programos naudotojai.

2. Diegimo (*Žr. priedą Nr.1 6 pav.*). Ši perspektyva yra atvaizduojama naudojant diegimo diagramą. Diegimo perspektyvoje yra aprėpiamas bendras mikroservisų architektūros modelis, konteinerio modelis, taip pat yra vaizduojamas kituose modeliuose esančių komponentų diegimas.

3. Įgyvendinimo (*Žr. priedą Nr.1 3,4,5 pav.*). Ši perspektyva yra atvaizduojama naudojant sekų diagramas. Įgyvendinimo perspektyvoje yra vaizduojamas API užklausų vartų modelis, servisų registro modelis, servisų atpažinimo modelis, duomenų bazė yra mikroservisas modelis, taip pat yra detalizuojami stebėjimo bei autentifikacijos ir autorizacijos modeliai.

Kuriamoje žiniatinklio programoje taip pat yra pasitelkiami saugumo filtrų ir grandinės pertraukties modeliai. Pagal šių dviejų modelių principus yra pritaikomi papildomi mechanizmai žiniatinklio programoje siekiant išspręsti šiuose modeliuose apibrėžiamas problemas, tačiau šie modeliai vizualinėse perspektyvose nėra atvaizduojami.

Žiniatinklio programoje ĮFAM yra sprendžiamos šios mikroservisų architektūros modelio problemos:

1. Servisų išskaidymo ir jų veikimo sričių apibrėžimo problema.
2. Išorinės ir vidinės komunikacijos problema.
3. Autentifikacijos ir autorizacijos problema.
4. Servisų veikimo stebėjimo problema.
5. Duomenų bazės panaudojimo problema.
6. Servisų registro ir servisų atpažinimo problema.
7. Servisų pasiekiamumo saugumo problema.
8. Klaidų toleravimo problema.
9. Klaidų pasekmių mažinimo problema.
10. Servisų talpinimo į konteinerius problema.

### 2.3. Žiniatinklio programos ir jos architektūros modelio kūrimo priemonių pasirinkimas

**Mikroservisų modelio kūrimas.** Mikroservisų modeliui kurti buvo naudojama vieningoji modeliavimo kalba (*Angl. Unified Modelling Language*). Atsižvelgiant į reikalavimų specifikacijoje (*Žr. priedą Nr.1*) keliamus tiražavimo reikalavimus, vieningajai modeliavimo kalbai panaudoti buvo pasirinkti du modeliavimo įrankiai: draw.io ir Lucidchart. Kadangi modeliavimui naudojant vieningą modeliavimo kalbą skirtų įrankių įvairovė yra didelė, būtent šių įrankių pasirinkimą paskatino šiuose įrankiuose esanti vieningosios modeliavimo kalbos diagramų ir jose turinčių atsispindėti komponentų įvairovė, platus įrankių naudojimo žinynas.

**Karkasas ir jame naudojama programavimo kalba.** Žiniatinklio programai, kuri remiasi mikroservisų modeliu, kurti buvo pasirinkta Java programavimo kalba ir šios kalbos karkasas Spring.

Naudojant Spring karkasą, programinės įrangos kūrėjams yra suteikiama galimybė į kuriamus projektus integruoti papildomas šio karkaso siūlomas integracijas, pavyzdžiui: Spring Boot,

Spring Security, Spring Cloud [32]. Naudojant šį karkasą ir kartu integruojant Spring palaikomus karkasus, yra palengvinamas mikroservisų ir jų architektūros kūrimo procesas, kuomet kuriamų programų funkcionalumas yra praplėčiamas naujomis bibliotekomis, šablonais ir serveriais [32]. Vienas iš šio karkaso privalumų – informatyvus ir plačios apimties pagalbos žinynas, Spring karkaso pamokos, gidai, bei mokymosi kursai [13].

**Karkaso integracijos.** Pagal keliamus reikalavimus reikalavimų specifikacijoje (*Žr. priedą Nr.1*) ir pagal karkaso teikiamas integracijos galimybes, žemiau yra pateikiamos panaudotos integracijos.

**Spring Boot**. Šios integracijos privalumas yra tai, jog ši integracija pagal standartinę konfigūraciją turi savyje integruotą tinklo serverį (*angl. web server*) Apache Tomcat, todėl panaudojus šią integraciją, kuriamas programas galima paleisti ir vykdyti be papildomos konfigūracijos [25].

**Spring Cloud**. Spring Cloud integracija praplečia Spring karkaso funkcionalumą ir suteikia debesijoje naudojamų ir priemonių palaikymą, tokių kaip: servisų atpažinimas, apkrovos paskirstymas, klaidų toleravimo priemonės [26].

**Spring Security**. Į šį karkasą įeina Spring karkaso siūlomi autentifikacijos ir autorizacijos mechanizmai, papildomos saugumą prieš atakas užtikrinančios priemonės, kurias atitinkamai galima panaudoti, keisti ir pritaikyti užtikrinant kuriamų programinių produktų saugumą [27].

**Spring WebFlux**. Šis Spring karkasas yra integruojamas į bazinį Spring karkasą ir suteikia galimybę programinės įrangos kūrėjams kurti reaktyvias programas (*angl. reactive applications*) bei yra suteikiama galimybė siųsti API užklausas asinchroniškai [18].

**Duomenų bazė ir jos valdymas.** Keliamiems reikalavimams duomenims, sistemų integravimo apribojimui Nr. 2 ir saugumo reikalavimui Nr.1 (*Žr. priedą Nr.1*) įgyvendinti, buvo pasirinkta reliacinė MySQL duomenų bazė.

Duomenų bazės valdymui buvo pasirinktas atvirojo kodo duomenų bazių valdymo įrankis DBeaver, palaikantis dažniausiai naudojamas duomenų bazes, tokias kaip: MySQL, PostgreSQL, SQLite [1]. Šis įrankis buvo pasirinktas dėl grafinės sąsajos ir jos valdymo patogumo, taip pat dėl to, jog šis įrankis yra tobulinamas ir būtent jo naudojimas gali pasitarnauti keliamų reikalavimų įgyvendinime.

**Išorinės integracijos.** Siekiant įgyvendinti keliamus reikalavimus, apibrėžtus reikalavimų specifikacijoje, tuo pačiu atsižvelgiant į apibrėžtus apribojimus, žemiau yra pateikiamos pasirinktos išorinės integracijos.

**Prometheus**. Prometheus – tai atviro kodo įrankių rinkinys, skirtas programinės įrangos parametrų stebėjimui, jų surinkimui bei pranešimų apie juos generavimui ir pateikimui [19]. Į šį įrankių rinkinį įeina Prometheus serveris, gebantis atpažinti veikiančius servisus ir surinkti programinės įrangos pateikiamus parametrus, pranešimų valdiklis, kuris yra atsakingas už pranešimų sugeneravimą ir jų perdavimą [19].

**Grafana**. Grafana – tai atviro kodo programinė įranga, suteikianti galimybę stebėti ir vizualizuoti parametrus, nepriklausomai nuo to, kur jie yra talpinami [5]. Grafana geba siųsti užklausas į duomenų šaltinius, kurie talpina informaciją, susijusią su parametrais ir taip išgauti reikalingus duomenis, kurie, pagal Grafana naudotojų poreikį, gali būti vizualizuojami grafikuose, kuriuose pasikeitimai gali būti atvaizduojami realiu laiku [5].

**Docker**. Docker – tai atvira platforma, orientuota į programinės įrangos kūrimą bei jos paleidimą ir ji leidžia programinę įrangą diegti į izoliuotos aplinkos konteinerius, kurie yra iš anksto sukonfigūruoti taip, jog palaikytų programinės įrangos veikimą [31]. Kuriamą programinę įrangą galima talpinti į Docker konteinerius, kuriuos galima patogiai testuoti, diegti produkcinėje aplinkoje, juos plėsti priklausomai nuo programinės įrangos, kurioje jie yra talpinami, sunaudojamų resursų kiekio [31].

**Keycloak**. Keycloak – tai atviro kodo tapatybės ir prieigos valdymo platforma, orientuota į autentifikacijos, autorizacijos, naudotojų valdymą programinėje įrangoje. Ši platforma palaiko OpenID Connect autentifikacijos protokolą, OAuth 2.0 autorizacijos standartą, SAML ir SSO autentifikacijos standartus, kartu užtikrinant tapatybės nustatymo ir jos valdymo saugumą [17].

Ši platforma žiniatinklio programos kūrimui buvo pasirinkta dėl autentifikacijos, autorizacijos ir naudotojų valdymo galimybių lankstumo, tuo pačiu užtikrinant saugumą naudojant tam specializuotus standartus.

**Resilience4j**. Resilience4j – tai papildomai integruojama biblioteka, kurią sudaro klaidų toleravimo funkcijos, tokios kaip: grandinės pertrauktis, kartojimo mechanizmas ar dažnumo limitavimo priemonė, o šią biblioteką galima naudoti funkcinėse sąsajose ar metoduose [14].

**Netflix Eureka**. Netflix Eureka – tai atviro kodo Netflix kompanijos sukurtas servisų registras, dažnai naudojamas programinėje įrangoje, kuri yra paremta mikroservisų architektūra [28]. Servisų registras atlieka servisų atpažinimą bei servisų veikimo ir būklės nustatymo funkcijas [28].

**Thymeleaf**. Thymeleaf – tai HTML šablonų valdymo sprendimas, leidžiantis į šiuos šablonus integruoti papildomą funkcionalumą [29]. Šį šablonų valdymo valdiklį galima integruoti kartu su Spring karkasu siekiant kuriamus šablonus papildyti naujomis ypatybėmis [29].

Žiniatinklio programai kurti buvo pasirinkta Java programavimo kalba ir jos karkasas Spring. Šis karkasas buvo pasirinktas atsižvelgiant į jo siūlomą priemonių, integracijų įvairovę, taip

pat į kuriamos žiniatinklio programos veiksnumui užtikrinti naudojamų išorinių integracijų palaikymą. Spring karkaso pasirinkimą įtakojo ir šio karkaso žinynas, kuriame aprašomos mikroservisų architektūros panaudojimo praktinės problemos ir pateikiami jų sprendimo būdai, aktualūs kuriant žiniatinklio programą ĮFAM.

Aukščiau aprašomos Spring karkaso siūlomos integracijos buvo pasirinktos dėl mažo jų konfigūracijos poreikio siekiant sutaupyti laiką ir tuo pačiu įgyvendinti keliamus reikalavimus, aprašytus reikalavimų specifikacijoje bei išspręsti apibrėžtas mikroservisų architektūros problemas. Kadangi šių integracijų atitikmenų kurti nereikia, jos gali būti tiesiogiai integruojamos į kuriamus komponentus.

Išorinės integracijos buvo pasirinktos atsižvelgiant į tai, ar šios integracijos gali būti integruojamos į naudojamą Spring karkasą. Konkrečių išorinių integracijų pasirinkimą lėmė jų funkcionalumo tinkamumas siekiant įgyvendinti žiniatinklio programos kūrimui suprojektuotus kriterijus.

# 3. ŽINIATINKLIO PROGRAMOS KŪRIMO PROCESAS

## 3.1. Mikroservisų ir jų funkcionalumo ribų apibrėžimas

Prieš pradedant kurti žiniatinklio programą, paremtą mikroservisų modeliu, buvo ypač svarbu apsibrėžti šioje programoje naudojamus mikroservisus ir jų funkcionalumo ribas. Naudojantis parengtoje reikalavimų specifikacijoje (*Žr. priedą Nr.1*) keliamais reikalavimais ir juos atvaizduojančiomis vizualinėmis perspektyvomis, buvo išskirti šie mikroservisai:

**Pagrindinis programos mikroservisas**. Pagrindinis programos mikroservisas naudojant HTTP siunčia užklausas į duomenų bazės mikroserviso galutinį tašką (angl. endpoint) siekiant gauti įdomių faktų apie mikroservisus sąrašą. Priklausomai nuo to, koks užklausos atsakymas yra gaunamas, yra suformuojamas faktų sąrašo atsakymas:

- Jeigu yra gaunamas įdomių faktų sąrašas iš duomenų bazės mikroserviso, jis yra apdorojamas ir paruošiamas pateikimui grafinėje sąsajoje.
- Jeigu įdomių faktų sąrašas iš duomenų bazės nėra gaunamas, yra tikrinama laikinoji duomenų saugykla. Jeigu yra saugomų duomenų laikinoje duomenų saugykloje, jie yra panaudojami kaip gautas sąrašas iš duomenų bazės mikroserviso, tačiau kartu yra sugeneruojamas pranešimas, jog yra naudojami duomenys iš laikinosios duomenų saugyklos ir siunčiant užklausą duomenų bazės mikroservisui, atsakymas nebuvo gautas. Jeigu jokių duomenų laikinoje duomenų saugykloje nėra saugoma, įdomių faktų sąrašas nėra formuojamas, tačiau yra sugeneruojamas pranešimas, jog nėra jokių duomenų, kuriuos galima pateikti.

Pagal tai, koks grafinės sąsajos langas yra rodomas ĮFAM žiniatinklio programos naudotojui, po atskleidimo valdiklio paspaudimo, jame yra atvaizduojamas įdomių faktų apie mikroservisus sąrašas ir pranešimas:

- Naudotojams yra pateikiamas įdomių faktų apie mikroservisus sąrašas. Jeigu duomenys yra gaunami tiesiogiai iš duomenų bazės mikroserviso, joks pranešimas nėra rodomas. Jei duomenys yra naudojami iš laikinosios duomenų saugyklos, arba jei nebuvo gautas atsakymas iš duomenų bazės mikroserviso ir laikinoji duomenų saugykla yra tuščia pateikiamas pranešimas apie duomenų pateikimo būseną ir kokio pobūdžio jie yra.

**Duomenų bazės mikroservisas.** Šiam mikroservisui gavus HTTP užklausą į jo galinį tašką, pirmiausia yra surenkami užklausai reikalingi parametrai (duomenų bazės prisijungimo duomenys, duomenų bazės URL), gaunami iš duomenų bazės mikroserviso konfigūracijos failo. Tada, atitinkamai pagal gautą užklausą galutiniame taške, yra suformuojama ir siunčiama užklausa į

duomenų bazę duomenims gauti. Gauti duomenys yra surenkami į sąrašą ir yra grąžinami kaip atsakymas į užklausą mikroservisui, kuris kreipėsi į duomenų bazės mikroserviso galinį tašką.

### 3.2. Pagalbinių servisų realizacija yra jų apibrėžta veikimo sritis

Remiantis reikalavimų specifikacija (*Žr. priedą Nr. 1*), buvo sukurti šie pagalbiniai servisai, kurie nėra priskiriami mikroservisams, tačiau yra reikalingi mikroservisų palaikymui:

**API užklausų vartai.** API užklausų vartai dalyvauja kaip sąsaja tarp žiniatinklio programos kliento (žiniatinklio programos naudotojo arba administratoriaus) ir žiniatinklio programos.
Žiniatinklio programos naudotojas, norėdamas pasiekti žiniatinklio programą, turi pateikti HTTP užklausą į API užklausų vartus. API užklausų vartai nukreipia srautą į pagrindinį programos mikroservisą ir klientui atlikus autentifikaciją ir autorizaciją, yra rodoma grafinė sąsaja, kuri yra pateikiama pagrindinio programos mikroserviso.
API užklausų vartuose yra pritaikytas mechanizmas, kuris paskirsto apkrovą tarp pagrindinio programos mikroserviso veikiančių kopijų (*angl. replicas*).
API užklausų vartuose yra pateikiama prisijungimo grafinė sąsaja, kurioje programos naudotojas turi įvesti prisijungimo duomenis. Tuomet prisijungimo duomenys yra surenkami ir yra siunčiama užklausa į išorinį tapatybės teikėją Keycloak, kuris atlieka kliento autentifikaciją ir autorizaciją ir grąžina prieigos žetoną.

**Servisų registras (Eureka serveris).** Servisų registras žiniatinklio programoje atlieka servisų atpažinimo ir servisų būklės stebėjimo funkcijas.

**Servisų atpažinimas**. Servisų registras siunčia „širdies dūžius“ (*angl. „heartbeats“*) į duomenų bazės mikroservisą, pagrindinį programos mikroservisą ir API užklausų vartų pagalbinį servisą (Eureka klientams). Servisų registras turi išankstinę konfigūraciją, kurioje yra nurodoma, koks turi būti „širdies dūžių“ siuntimo dažnis, kiek laiko turi servisas būti laikomas aktyviu. Prieš servisų atpažinimą, kiekvienas Eureka klientas yra papildomai sukonfigūruojamas, jog jis būtų atpažįstamas kaip Eureka klientas, taip pat, yra nustatomas Eureka kliento vardas, pagal kurį yra atpažįstamas servisas. Servisų registro atpažinti servisai yra pasiekiami tame pačiame tinkle, todėl kreipiantis naudojant HTTP užklausą, galima kreiptis nurodant serviso vardą vietoj jo IP adreso.

**Servisų būklės stebėjimas.** Eureka serveris, sulaukęs užklausos į jo galinį tašką (*angl. endpoint*), pateikia grafinę sąsają, kurioje yra matomi šiuo metu užfiksuoti kaip veikiantys servisai, taip pat yra rodomas Eureka serverio veikimo laikas.

### 3.3. Duomenų bazės struktūra ir jos veikimas žiniatinklio programoje

ĮFAM žiniatinklio programoje yra naudojama MySQL reliacinė duomenų bazė, į kurią gali kreiptis tik jį valdantis mikroservisas, šiuo atveju – duomenų bazės mikroservisas.

**Duomenų bazės sandara**. Duomenų bazę sudaro viena lentelė, kurioje yra apibrėžti šie atributai (stulpeliai):

- **id** (Duomenų tipas: bigint). Šio atributo paskirtis yra indeksuoti įrašus lentelėje.
- **facts** (Duomenų tipas: varchar(255)). Šis atributas yra naudojamas laukams, kuriuose yra saugoma įdomių faktų apie mikroservisus informacija, apibrėžti.

**Duomenų bazės veikimas.** Duomenų bazėje yra saugoma informacija apie įdomius faktus apie mikroservisus, gavus SQL užklausą į duomenų bazę, atitinkamai nuo užklausos pobūdžio (šiuo atveju, visiems faktams surinkti), yra atliekamos operacijos užklausai patenkinti. Tuomet yra grąžinamas atsakymas užklausos siuntėjui.

### 3.4. Išorinė autentifikacijos, autorizacijos ir programos naudotojų valdymo integracija

Žiniatinklio programoje jos naudotojų valdymui, jų autentifikacijai ir autorizacijai atlikti yra naudojamas išorinis tapatybės teikėjas (*Angl. Identity provider*) Keycloak.

Tapatybės teikėjo platformos veikimo metu yra suteikiama administratoriaus grafinė sąsaja, į kurią prisijungus pateikiant Keycloak administratoriaus prisijungimo duomenis, yra atveriamas konfigūracijų langas. Tapatybės teikėjo administratoriaus grafinėje sąsajoje yra atliekama tapatybės teikėjo papildoma konfigūracija naudotojų valdymui, autentifikacijai ir autorizacijai.

Pirmiausia, yra sukurta karalystė (*angl. realm*) – tai konfigūracijos (naudotojų, rolių, nukreipimo, autentifikacijos, autorizacijos) valdymo sfera. Karalystėje yra sukuriami du klientai: žiniatinklio programos naudotojų klientas (web-app) ir žiniatinklio programos vidinės komunikacijos klientas (app-mgmt). Klientai – tai esybės, kurios gali prašyti autentifikuoti naudotoją arba pasinaudoti kliento autentifikacijos informacija komunikacijai tarp servisų, apsaugotų Keycloak.

Žiniatinklio programos naudotojų klientas yra atsakingas už išorinių žiniatinklio naudotojų autentifikaciją ir autorizaciją, šio kliento konfigūracijoje yra nustatomi OpenID Connect autentifikacijos ir OAuth 2.0 autorizacijos nustatymai, taip pat nustatomos nuorodos, kurios įtakoja šį klientą: galimos nukreipimo nuorodos, nukreipimo nuoroda atsijungiant, šio kliento veikimo sferos (šiuo atveju, API užklausų vartų serviso) adresas. Papildomai yra nustatoma šio kliento autentifikavimo priemonė: Kliento ID ir kliento paslaptis - šie du parametrai yra reikalingi kliento autentifikacijai atlikti. Taip pat yra sukuriamos dvi atskiros rolės: user (rolė yra orientuota į naudotoją) ir admin (rolė yra orientuota į administratorių)

Vidinės komunikacijos klientas yra atsakingas už vidinę komunikaciją tarp mikroservisų. Vidinės komunikacijos kliento konfigūracijoje nustatoma, jog šis klientas bus konfidencialus ir jis bus pasiekiamas tik su Bearer žetonu, taip pat yra nustatomi Kliento ID ir kliento paslapties parametrai. Šiam klientui taip pat yra sukuriamas nauja kliento apibrėžtis (*angl. client scope*)

Naudotojų konfigūracijoje yra sukuriami du atskiri naudotojai: user, admin. Šiems naudotojams yra sukuriami prisijungimo vardai ir papildomai yra nustatomi kiti prisijungimo duomenys: elektroninis paštas, sugeneruojamas. Papildomai šiems klientams yra priskiriamos rolės: user naudotojui – rolė „user“, admin naudotojui – rolė „admin“. Šios rolės bus reikalingos tam, jog juos būtų galima ateityje atlikti naudotojų valdymą pagal roles.

Lygiagrečiai su tapatybės teikėjo platforma yra paleidžiama šiam teikėjui skirta MySQL duomenų bazė, kurioje yra talpinamos visos konfigūracijos pagal aktyvias karalystes. Keycloak dokumentacija šios duomenų bazės struktūros nenurodo, tačiau ji yra pasiekiama prisijungus prie šios duomenų bazės pateikiant prisijungimo duomenis, kuriuos naudoja Keycloak.

### 3.5. Komunikacija tarp žiniatinklio programos naudotojų ir API užklausų vartų

API užklausų vartuose yra nustatoma papildoma OpenID galinių taškų (*angl. endpoints*) konfigūracija, kuri yra pateikiama karalystės nustatymuose, esančiuose Keycloak administratoriaus grafinėje sąsajoje. Papildomai yra nustatomas naudojamo kliento ID ir kliento paslaptis, kurie yra reikalingi naudotojų, bandančių pasiekti API užklausų vartus, autentifikacijai ir autorizacijai. Žiniatinklio programos naudotojams bandant pasiekti API užklausų vartus, jie yra nukreipiami į Keycloak prisijungimo puslapį. Naudotojui pateikus prisijungimo duomenis, jis yra autentifikuojamas (nustatoma, kas tai per naudotojas) ir autorizuojamas (nustatomos šio kliento rolės ir apibrėžtys), tada šis tapatybės teikėjas grąžina prieigos žetoną (*angl. access token*) atgal į API užklausų vartus ir naudotojas yra patvirtinamas.

### 3.6. Komunikacija tarp mikroservisų

Šioje žiniatinklio programoje naudojami mikroservisai: pagrindinis programos mikroservisas ir duomenų bazės mikroservisas yra papildomai sukonfigūruojami: pagrindiniame programos mikroservise yra naudojamas vidinės komunikacijos klientas ir tolimesniems veiksmams atlikti (kreiptis į duomenų bazės mikroservisą) yra naudojama šio kliento konfigūracija. Pagrindiniam programos mikroservisui prieš siunčiant užklausą duomenų bazės mikroservisui, pirmiausiai yra siunčiama užklausa į Keycloak žetonų galinį tašką kartu pateikiant vidinės komunikacijos kliento ID ir kliento paslaptį ir iš jo gauna atsakymą su šio kliento prieigos žetonu. Tuomet yra siunčiama užklausa naudojant gautą žetoną kaip Bearer žetoną į duomenų bazės mikroservisą. Duomenų bazės

mikroserviso saugumo filtras gavus užklausą tikrina, ar šiame žetone yra nurodytas vidinės komunikacijos apibrėžtis ir jeigu taip, užklausa yra patenkinama.

### 3.7. Žiniatinklio programos komponentų ir išorinių integracijų talpinimas į konteinerius

Žiniatinklio programos išorinės integracijos ir jų duomenų bazės yra talpinamos į Docker konteinerius naudojant Docker Compose. Konteineriuose yra nurodomos papildomos konteinerių konfigūracijos, nurodoma, kokie yra naudojami atvaizdai ir kur yra saugoma informacija lokalioje sistemoje, jeigu būtų nutraukiama konteinerių veikla.

### 3.8. Žiniatinklio programos pagalbinių servisų ir joje esančių mikroservisų stebėjimas

Naudojant Spring Boot Actuator, pagrindiniam programos mikroservisui, duomenų bazės mikroservisui, API užklausų vartams, Eureka serveriui (servisų registrui) yra automatiškai sukuriamas naujas galinis taškas, kuriame yra pateikiamos visi veikimo parametrai (gyvybės, veikimo laiko ir kiti.), kurie padeda nusakyti veikiančio serviso charakteristikas.

**Prometheus integracija.** Prometheus serveris yra papildomai sukonfigūruojamas: specialiame konfigūracijos faile yra nustatomi darbai, kuriuos šis serveris turi atlikti, šiuo atveju, servisų parametrų gavimas (*angl. scrape*). Kiekvienam darbui yra nurodomas darbo objektas (*angl. target*), kelias, kuriame yra pateikiami parametrai (*angl. metrics path*), Papildomai yra nustatoma, koks yra parametrų išgavimo intervalas. Šiuo atveju, Prometheus serveris yra naudojamas Grafana integracijai palaikyti.

**Grafana integracija.** Grafana serveris yra papildomai sukonfigūruojamas: konfigūracijos faile, sukurtame po jo konteinerio paleidimo, yra nustatomas duomenų šaltinis, į kurį Grafana geba siųsti užklausas ir išgauti parametrus, kuriuos galima vizualiai atvaizduoti, šiuo atveju – Prometheus išgautų parametrų vizualizacijai. Grafana siūlo grafinę sąsają, kurioje galima kurti skydelius (*angl. dashboards*) ir juose atvaizduoti pasirinktus grafikus, juos konfigūruoti ir pritaikyti pagal poreikius. ĮFAM žiniatinklio programos parametrų informacijai atvaizduoti yra sukuriamas skydelis, kuriame administratorius gali pasirinkti tam tikro serviso filtravimą, pagal kurį yra atvaizduojamas jo veikimo laikas, operatyviosios atminties, procesoriaus, disko resursų sunaudojimas, taip pat suteikiama galimybė kurti papildomus grafikus ar naujus skydelius.

### 3.9. Klaidų toleravimas ir jų pasekmių mažinimas

**Apkrovos paskirstymo filtras.** API užklausų vartuose yra naudojamas apkrovos paskirstymo filtras. Šis filtras yra panaudojamas žiniatinklio programos naudotojo nukreipimo į pagrindinį programos mikroservisą konfigūracijoje. Šis filtras palaiko servisų registro servisų atpažinimą ir yra veikiantis tik tada, kuomet yra kelios pagrindinio programos mikroserviso identiškos kopijos (*angl. replicas*). Kartu pritaikant servisų atpažinimą, yra nurodoma, kokiu vardu veikiančių servisų apkrovą reikia

paskirstyti, šiuo atveju – nurodomas pagrindinio programos mikroserviso vardas. Negana to, jei viena iš kopijų tampa nepasiekiama, automatiškai vyksta nukreipimas į kitą veikiančią kopiją.

**Grandinės pertrauktis.** Grandinės pertrauktis yra naudojama pasitelkiant Resilience4j biblioteką ir šis mechanizmas yra naudojamas API užklausų vartuose ir pagrindiniame programos. Grandinės pertrauktis API užklausų vartuose yra naudojama kartu su srauto nukreipimu, mikroservisuose – užklausų siuntimo metu.

Grandinės pertrauktis veikia skaičiavimo principu, kuomet yra skaičiuojama, kiek yra nepavykusių išsiųsti paskutinių užklausų ir pagal tai yra keičiama grandinės būsena. Naudojamos grandinės būsenos yra šios: atidaryta, pusiau atidaryta ir uždaryta. Uždarytos grandinės metu grandinės pertrauktis nesiima papildomų veiksmų, tačiau pagal nustatytą konfigūraciją tikrina paskutines siųstas užklausas. Jeigu nepavykusių išsiųsti užklausų nustatytas limitas yra viršijamas, grandinė yra atidaroma ir užklausos nebėra siunčiamos. Praėjus nustatytam laiko limitui nuo grandinės atidarymo, grandinė yra pusiau atidaroma – užklausų siuntimas tampa nebeblokuojamas. Tikrinimo metu yra siunčiamos užklausos pačios grandinės pertraukties ir jei užfiksuojamas nepavykusių užklausų kiekis didesnis arba lygus nustatytai ribai, grandinė vėl yra atidaroma. Jei nepavykusių užklausų kiekis yra mažesnis – grandinė yra uždaroma.

Grandinės pertraukties konfigūracijoje papildomai yra nustatomi skaičiavimo principo parametrai: tikrinamų užklausų kiekis, nepavykusių užklausų limitas, atidarytos grandinės laikas, užklausų siuntimo kiekis pusiau atidarytos grandinės metu, minimalus užklausų kiekis nesėkmių rodikliui skaičiuoti, taip pat papildomai yra nurodoma, jog grandinės pertrauktis galėtų automatiškai pakeisti būsena iš atidarytos į pusiau atidarytą.

**Laikinoji duomenų saugykla.** Ši saugykla yra naudojama pagrindiniame programos mikroservise. Laikinoje duomenų saugykloje yra talpinami gaunami duomenys užklausų siuntimo metu. Kuomet yra gaunamas atsakymas iš duomenų bazės mikroserviso, pagrindiniame programos mikroservise laikinoji duomenų saugykla yra papildoma po sėkmingos užklausos į duomenų bazės mikroservisą gautais įdomiais faktais apie mikroservisus. Kartu panaudojant grandinės pertrauktį, laikinoji duomenų saugykla gali būti naudojama nesiunčiant išorinių užklausų norint gauti atsakymą ir panaudojant laikinoje duomenų saugykloje saugomą informaciją gautai užklausai patenkinti. Kadangi ši duomenų saugykla yra sukuriama tik mikroserviso paleidimo metu, ji yra laikinojo pobūdžio, naudojant ją, sumažinamos galimų iškylančių klaidų neigiamos pasekmės.

### 3.10. Žiniatinklio programos testavimas

Žiniatinklio programos testavimams apibrėžti buvo sukurtas testavimo planas, kuris yra pateikiamas priede Nr. 2 (*Žr. priedą Nr.2*). Atlikti testavimai žiniatinklio programoje:

**1)Testavimo atvejis**

- Testavimo priemonės: JUnit ir Mockito
- Testavimo aplinka: Spring karkasas, konkrečiai, duomenų bazės mikroservisas.
- Testavimo charakteristikos: Testavimo metu buvo testuojamas metodas, atsakingas už duomenų surinkimą. Duomenų bazės įrašams simuliuoti buvo pasirinkti du įrašai.

Gauti rezultatai:

1. Gautų duomenų sąrašas nėra tuščias, yra matomi du įrašai
2. Gautų duomenų sąraše yra du elementai
3. Tikrinant abu simuliuojamus įrašus, yra aptinkami abu įrašai pagal jų parametrus.

**5) Testavimo atvejis**

- Testavimo priemonės: JUnit ir Mockito
- Testavimo aplinka: Spring karkasas, konkrečiai, pagrindinis programos mikroservisas.
- Testavimo charakteristikos: Testavimo metu buvo testuojamas metodas, atsakingas už grafinės sąsajos pateikimą. Duomenų bazės įrašams simuliuoti buvo pasirinkti vienas įrašas, pranešimui simuliuoti buvo pasirinktas vienas pranešimas.

Gauti rezultatai:

1. Į naudotojo grafinę sąsają yra pridedami du atributai: faktų atributas ir pranešimų atributas
2. Grąžinama naudotojo grafinė sąsaja
3. Testavimo metu yra iškviečiami duomenų surinkimo iš duomenų bazės ir vidinės komunikacijos prieigos žetono gavimo metodai, kurių vykdymas yra simuliuojamas

**6) Testavimo atvejis**

- Testavimo priemonės: JUnit ir Mockito
- Testavimo aplinka: Spring karkasas, konkrečiai, pagrindinis programos mikroservisas.
- Testavimo charakteristikos: Testavimo metu buvo testuojamas metodas, atsakingas už grafinės sąsajos pateikimą. Duomenų bazės įrašams simuliuoti buvo pasirinkti vienas įrašas, pranešimui simuliuoti buvo pasirinktas vienas pranešimas.

Gauti rezultatai:

1. Į administratoriaus grafinę sąsają yra pridedami du atributai: faktų atributas ir pranešimų atributas
2. Grąžinama administratoriaus grafinė sąsaja
3. Testavimo metu yra iškviečiami duomenų surinkimo iš duomenų bazės ir vidinės komunikacijos prieigos žetono gavimo metodai, kurių vykdymas yra simuliuojamas

**12) Testavimo atvejis**

- Testavimo priemonės: JUnit ir Mockito
- Testavimo aplinka: Spring karkasas, konkrečiai, API užklausų vartai.

- Testavimo charakteristikos: Testavimas atliekamas metode, grąžinančiam grafinės sąsajos langą

Gauti rezultatai:

1. Pateikiama grafinė sąsaja, kurio yra pateikiamas pranešimas apie pagrindinio programos mikroserviso neveikimą

**13) Testavimo atvejis**

- Testavimo priemonės: JUnit ir Mockito
- Testavimo aplinka: Spring karkasas, konkrečiai, API užklausų vartai.
- Testavimo charakteristikos: Testavimas atliekamas metode, grąžinančiam grafinės sąsajos langą

Gauti rezultatai:

1. Pateikiama grafinė sąsaja, kurio yra pateikiamas pranešimas apie servisų registro neveikimą

**14) Testavimo atvejis**

- Testavimo priemonės: JUnit ir Mockito
- Testavimo aplinka: Spring karkasas, konkrečiai, API užklausų vartai.
- Testavimo charakteristikos: Testavimas atliekamas metode, grąžinančiam grafinės sąsajos langą

Gauti rezultatai:

1. Pateikiama grafinė sąsaja, kurio yra pateikiamas pranešimas apie Grafana neveikimą

**15) Testavimo atvejis**

- Testavimo priemonės: JUnit ir Mockito
- Testavimo aplinka: Spring karkasas, konkrečiai, API užklausų vartai.
- Testavimo charakteristikos: Testavimas atliekamas metode, grąžinančiam grafinės sąsajos langą

Gauti rezultatai:

1. Pateikiama grafinė sąsaja, kurio yra pateikiamas pranešimas apie Prometheus neveikimą

**16) Testavimo atvejis**

- Testavimo priemonės: Postman
- Testavimo aplinka: Postman įrankis Testavimo charakteristikos: testuojama pagrindinio programos mikroserviso ir duomenų bazės mikroserviso komunikacija.
- Testavimo charakteristikos: Testavimas atliekamas gaunant vidinės komunikacijos prieigos raktą (imituojant rakto gavimo metodą) ir siunčiant užklausą į duomenų bazės mikroserviso galinį tašką (imituojant duomenų surinkimo metodą)

Gauti rezultatai:

1. Pagrindinis programos mikroservisas gauna vidinės komunikacijos (app-mgmt kliento) prieigos raktą iš tapatybės teikėjo
2. Pagrindinis programos mikroservisas siunčia užklausas su Bearer žetonu į duomenų bazės mikroservisą
3. Duomenų bazės mikroservisas priima užklausą tik su vidinės komunikacijos prieigos žetonu.

**24) Testavimo atvejis**

- Testavimo priemonės: Postman
- Testavimo aplinka: Postman įrankis
- Testavimo charakteristikos: Testavimas atliekamas siunčiant užklausas į API užklausų vartų galinius taškus (nukreiptus pagrindinio programos mikroserviso taškus).

Gauti rezultatai:

1. API užklausų vartų galiniai taškai yra pasiekiami naudojant API užklausas, tačiau yra pateikiamas prisijungimo langas.

**25) Testavimo atvejis**

- Testavimo priemonės: Vizualus testavimas
- Testavimo aplinka: internetinė naršyklė
- Testavimo charakteristikos: atliekamas vizualinis testavimas siunčiant užklausas į API užklausų vartų galinius taškus (nukreiptus pagrindinio programos mikroserviso taškus).

Gauti rezultatai:

1. Jeigu yra aktyvi naudotojo sesija, žiniatinklio programos naudotojui yra pateikiamas grafinės sąsajos langas, į kurį kreipėsi
2. Jeigu nėra aktyvios sesijos, yra pateikiamas prisijungimo langas

**26) Testavimo atvejis**

- Testavimo priemonės: Postman
- Testavimo aplinka: Postman įrankis
- Testavimo charakteristikos: Testavimas atliekamas siunčiant užklausas į pagrindinio programos mikroserviso taškus.

Gauti rezultatai:

1. Jeigu užklausos siuntimo metu yra pridedamas aktyvus prieigos žetonas, pagrindinis programos mikroservisas yra pasiekiamas, gaunamas HTTP 200 kodas.
2. Jeigu užklausos siuntimo metu yra nepateikiamas prieigos žetonas, yra grąžinamas HTTP 401 kodas.
3. 3.Jeigu užklausos siuntimo metu yra pateikiamas klaidingas prieigos žetonas, grąžinamas HTTP 401 kodas

**27) Testavimo atvejis**

- Testavimo priemonės: Vizualus testavimas
- Testavimo aplinka: internetinė naršyklė
- Testavimo charakteristikos: Atliekamas vizualinis testavimas siunčiant užklausas į pagrindinio programos mikroserviso taškus

Gauti rezultatai:

1. Naudojant grafinę sąsają, pagrindinis programos mikroservisas yra tiesiogiai nepasiekiamas, yra grąžinamas HTTP 401 kodas.

**28) Testavimo atvejis**

- Testavimo priemonės: Postman
- Testavimo aplinka: Postman įrankis
- Testavimo charakteristikos: Testavimas atliekamas siunčiant užklausas į duomenų bazės mikroserviso taškus

Gauti rezultatai:

1. Jeigu užklausos siuntimo metu yra pridedamas vidinės komunikacijos kliento (app-mgmt) prieigos žetonas, duomenų bazės mikroservisas yra pasiekiamas, grąžinamas HTTP 200 kodas.
2. Jeigu užklausos siuntimo metu yra nepateikiamas prieigos žetonas, yra grąžinamas HTTP 401 kodas.
3. Jeigu užklausos siuntimo metu yra pateikiamas klaidingas prieigos žetonas, grąžinamas HTTP 401 kodas.
4. Jeigu užklausos siuntimo metu yra pateikiamas žiniatinklio programos naudotojo (web-app) prieigos žetonas, grąžinamas HTTP 401 kodas.

**29) Testavimo atvejis**

- Testavimo priemonės: Vizualus testavimas
- Testavimo aplinka: internetinė naršyklė
- Testavimo charakteristikos: Atliekamas vizualinis testavimas siunčiant užklausas į duomenų bazės mikroserviso taškus

Gauti rezultatai:

1. Naudojant grafinę sąsają, duomenų bazės mikroservisas yra nepasiekiamas, yra grąžinamas HTTP 401 kodas.

### 3.11. Žiniatinklio programos ĮFAM kūrimo iššūkiai

Žiniatinklio programos kūrimo ir realizacijos planavimo metu teko susidurti su išorinių sistemų integracijos problemomis, dėl kurių nepavyko įgyvendinti visų keliamų funkcinių ir

nefunkcinių reikalavimų. ĮFAM buvo neįgyvendinta atsijungimo (sąsajos nutraukimo sistema), kuri yra apibrėžiama priede Nr.1 1 ir 2 pav., taip pat 4 ir 7 funkciniuose reikalavimuose (*Žr. priedą Nr.1*).

Dar vienas kriterijus, kuris nebuvo įgyvendintas kūrimo procese – rolių valdymas pagal naudotojo tipą. Šis funkcionalumas yra apibrėžiamas 1 funkciniame reikalavime (*Žr. priedą Nr.1*) Viena iš priežasčių, kuri lėmė šių kriterijų neįgyvendinimą yra išorinio paslaugų teikėjo programinės įrangos ir karkaso, kuris buvo naudojamas žiniatinklio programai kurti, šio funkcionalumo palaikymo sprendimų trūkumas.

ĮFAM žiniatinklio programa yra dalinai realizuota, buvo įgyvendinti visi funkciniai reikalavimai, išskyrus 1 ir 7 reikalavimus (*Žr. priedą Nr.1*). Tam, jog būtų galima įsitikinti, ar ši žiniatinklio programa įgyvendina visus keliamus reikalavimus ir ją būtų galima diegti į įmonėje naudojamas sistemas, yra būtina ją ištestuoti pagal visus keliamus testavimo atvejus, esančius testavimo plane (*Žr. priedą Nr.2*). Žiniatinklio programos kūrimo procese teko susidurti su testavimo iššūkiais, kuomet dėl mikroservisų architektūrai būdingo išskaidymo buvo ganėtinai sudėtinga apibrėžti visus testavimo atvejus.

# Išvados

1. Atlikus literatūrinių šaltinių apžvalgą buvo identifikuotos programinės įrangos problemos, kurios yra sprendžiamos naudojant mikroservisų architektūrą.
2. Mikroservisų architektūros problemoms nustatyti buvo sukurtas mikroservisų architektūros modelis, sudarytas iš dešimties į skirtingas problemas specializuotų modelių, o modelis yra atvaizduojamas skirtingose vizualinėse perspektyvose.
3. Pagal keliamus konkrečios žiniatinklio programos reikalavimus ir vizualines perspektyvas, buvo sukurti žiniatinklio programos komponentai: mikroservisai, pagalbiniai servisai ir buvo pritaikytos išorinės integracijos. Visi komponentai yra apjungti į vientisą žiniatinklio programą.
4. Iš žiniatinklio programos testavimo rezultatų seka, jog buvo išspręstos visos mikroservisų architektūros modelyje apibrėžiamos problemos, išskyrus autentifikacijos ir autorizacijos bei servisų talpinimo į konteinerius problemas. Siekiant įsitikinti, jog problemų sprendimo būdai neturi iš anksto nenumatytų trūkumų, yra būtina atlikti tolimesnius testavimus.

# Rekomendacijos

1. Tam, jog būtų įgyvendinami visi reikalavimai reikalavimų specifikacijoje ir būtų išlaikomas mikroservisų architektūros principas, mikroservisus ir pagalbinius servisus reikia patalpinti į konteinerius, jog juos būtų galima apjungti į klasterius ir diegti, tačiau prieš diegimą reikia atlikti papildomus testavimus ir rasti sprendimus pritaikyti reikalingus mechanizmus, tokius kaip: programos naudotojų skirstymas į atskiras roles, sesijos nutraukimas (atsijungimas).
2. Sukurtos žiniatinklio programos pilnam funkcionalumui ir tinkamumui įvertinti, reikia integruoti jau įmonės sferoje naudojamas duomenų bazes tiek duomenims talpinti, tiek informacijai apie naudotojus saugoti, taip pat integruoti ir papildomas įmonės sistemas.
3. Sukurtai žiniatinklio programai konfigūruoti reikia sukurti centruotą konfigūracijų serverį, kuriame būtų galima valdyti visų pagalbinių servisų ir mikroservisų konfigūracijas vienoje erdvėje. Šiuo metu konfigūracija kiekviename komponente yra atliekama atskirai, todėl šio konfigūracijos serverio pritaikymas sumažintų konfigūravimo laiką, taip pat padėtų išvengti tolimesnių konfigūracijos tarp atskirų komponentų nesuderinamumo.

# Literatūra


1. DBEAVER.IO. *About*. Prieiga internete: https://dbeaver.io/about/ [žiūrėta 2025 01 06]

2. BENATALLAH, Boualem; MOTAHARI NEZHAD, Hamid R. Service oriented architecture: Overview and directions. *Advances in Software Engineering: Lipari Summer School 2007, Lipari Island, Italy, July 8-21, 2007, Revised Tutorial Lectures*, 2008, 116-130.

3. BLINOWSKI, Grzegorz; OJDOWSKA, Anna; PRZYBYŁEK, Adam. Monolithic vs. microservice architecture: A performance and scalability evaluation. *IEEE Access*, 2022, 10: 20357-20374.

4. CERNY, Tomas; DONAHOO, Michael J.; PECHANEC, Jiri. Disambiguation and comparison of soa, microservices and self-contained systems. In: *Proceedings of the International Conference on research in adaptive and convergent systems*. 2017. p. 228-235.

5. GRAFANA.COM *Dashboard everything. Observe everything* Prieiga internete: https://grafana.com/grafana/ [žiūrėta 2025 01 06]

6. DAVID, Nicoleta; CARSTEA, Claudia. Web Applications Architecture. *Review of the" Henry Coanda" Air Force Academy, The Scientific Informative Review No. 1/2008*, 2008, 71-75.

7. DOS SANTOS KRUG, Daniel; CHANIN, Rafael; SALES, Afonso. Exploring the Pros and Cons of Monolithic Applications versus Microservices.

8. DRAGONI, Nicola, et al. Microservices: How to make your application scale. In: *Perspectives of System Informatics: 11th International Andrei P. Ershov Informatics Conference, PSI 2017, Moscow, Russia, June 27-29, 2017, Revised Selected Papers 11*. Springer International Publishing, 2018. p. 95-104.

9. GHOFRANI, Javad; LÜBKE, Daniel. Challenges of Microservices Architecture: A Survey on the State of the Practice. *ZEUS*, 2018, 2018: 1-8.

10. GIAO, Joao, et al. A framework for service-oriented architecture (SOA)-based IoT application development. *Processes*, 2022, 10.9: 1782.

11. GINIGE, Athula; MURUGESAN, San. The essence of web engineering-managing the diversity and complexity of web application development. *IEEE multimedia*, 2001, 8.2: 22-25.

12. GOS, Konrad; ZABIEROWSKI, Wojciech. The comparison of microservice and monolithic architecture. In: *2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*. IEEE, 2020. p. 150-153.

13. SPRING.IO. *Guides*. Prieiga internete: https://spring.io/guides [žiūrėta 2025 01 06]

14. RESILIENCE4J.README.IO *Introduction* Prieiga internete: https://resilience4j.readme.io/docs/getting-started [žiūrėta 2025 01 06]

15. KOROTENKO, Anton. Microservices Architecture: practical implementations, benefits, and nuances. 2024.

16. MÁRQUEZ, Gastón; ASTUDILLO, Hernán. Actual use of architectural patterns in microservices-based open source projects. In: *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. Ieee, 2018. p. 31-40.

17. KEYCLOAK.ORG. *Open Source Identity and Access Management* Prieiga internete: https://www.keycloak.org/ [žiūrėta 2025 01 06]

18. DOCS.SPRING.IO. *Overview*. Prieiga internete: https://docs.spring.io/spring-framework/reference/web/webflux/new-framework.html [žiūrėta 2025 01 06]

19. PROMETHEUS.IO. *Overview*. Prieiga internete: https://prometheus.io/docs/introduction/overview/ [žiūrėta 2025 01 06]

20. PONCE, Francisco; MÁRQUEZ, Gastón; ASTUDILLO, Hernán. Migrating from monolithic architecture to microservices: A Rapid Review. In: *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*. IEEE, 2019. p. 1-7.

21. RICHARDS, Mark. Microservices Architecture Pattern. In: *Software architecture patterns*. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, Incorporated, 2015.

22. RUDRABHATLA, Chaitanya K. Security design patterns in distributed microservice architecture. *arXiv preprint arXiv:2008.03395*, 2020.

23. SÖYLEMEZ, Mehmet; TEKINERDOGAN, Bedir; KOLUKISA TARHAN, Ayça. Challenges and solution directions of microservice architectures: A systematic literature review. *Applied sciences*, 2022, 12.11: 5507.

24. SOLBERG, Eivind. *The transition from monolithic architecture to microservice architecture: A case study of a large Scandinavian financial institution*. 2022. Master's Thesis.

25. SPRING.IO. *Spring Boot*. Prieiga internete: https://spring.io/projects/spring-boot [žiūrėta 2025 01 06]

26. SPRING.IO. *Spring Cloud* Prieiga internete: https://spring.io/projects/spring-cloud [žiūrėta 2025 01 06]

27. SPRING.IO. *Spring Security*. Prieiga internete: https://spring.io/projects/spring-security [žiūrėta 2025 01 06]

28. CLOUD.SPRING.IO *Spring Cloud Netflix*.Prieiga internete: https://cloud.spring.io/spring-cloud-netflix/reference/html/#service-discovery-eureka-clients [žiūrėta 2025 01 06]

29. THYMELEAF.ORG *Thymeleaf.* Prieiga internete: https://www.thymeleaf.org/index.html [žiūrėta 2025 01 06]

30. VALIPOUR, Mohammad Hadi, et al. A brief survey of software architecture concepts and service oriented architecture. In: *2009 2nd IEEE International Conference on Computer Science and Information Technology*. IEEE, 2009. p. 34-38.

31. DOCKS.DOCKER.COM. *What is Docker?* Prieiga internete: https://docs.docker.com/get-started/docker-overview/#what-can-i-use-docker-for [žiūrėta 2025 01 06]

32. SPRING.IO. *Why Spring?* Prieiga internete: https://spring.io/why-spring [žiūrėta 2025 01 06]

# Priedai

***Priedas Nr. 1***

## Reikalavimų specifikacija

## Įvadas

### Realizuojamos žiniatinklio programos pavadinimas

Kuriamos žiniatinklio programos pavadinimas: „Įdomūs faktai apie mikroservisus“. Šio pavadinimo trumpinys naudojamas reikalavimų specifikacijoje – ĮFAM

### Žiniatinklio programos reikalingumas

Žiniatinklio programa yra naudinga įmonei „Cherry Servers“, kuri teikia debesijos paslaugas ir užsiima dedikuotų serverių priegloba. Šios įmonės plėtra paslaugų teikimo sferoje ir vis didėjanti kitų įmonių konkurencija skatina imtis inovacijų, kurių dėka siekiama užtikrinti nenutrūkstamą paslaugų teikimą, tuo pačiu užtikrinant ir šių paslaugų kokybę. Norint visa tai užtikrinti, yra būtina tobulinti įmonės vidinę infrastruktūrą atsižvelgiant į rinkoje naudojamas sistemas, jų privalumus, trūkumus, pritaikymo ir panaudojimo galimybes, tuo pačiu užtikrinant klientų, įmonės ir jos darbuotojų duomenų saugumą.

### ĮFAM žiniatinklio programos paskirtis

Vienas iš būdų, kurį siekia tobulinti įmonė „Cherry Servers“ – naudojamų vidinių programų ir jų sistemų architektūra ir ši žiniatinklio programa, paremta mikroservisų architektūra, padės įvertinti infrastruktūros tobulinimo galimybes.

### Sprendžiamos problemos

- **Šiuo metu įmonėje taikomos architektūros vidinėje infrastruktūroje problema.** Šios problemos priežastis yra nuo įmonės sukūrimo besitęsianti įmonės plėtra tiek paslaugų sektoriuje, tiek įmonės viduje. Bėgant laikui, buvo atliekamos analizės plėtros įvertinimui ir rengiami plėtros planai, kurių pagalba buvo tobulinamos jau esamos ir tuo pačiu integruojamos naujos sistemos taikant monolitinės architektūros principus. Nors ši problema iš pradžių nebuvo pastebima dėl tuo metu teikiamų paslaugų pobūdžio ir klientų kiekio, tačiau palaipsniui ši problema tampa vis didesnė. Monolitinės sistemos, kurias naudoja įmonė, dažnu atveju yra viena nuo kitos priklausomos, turi mažą klaidų toleravimą ir jas plėsti yra sudėtinga, todėl įmonė nori

pradėti taikyti mikroservisų architektūrą, kuri galėtų šias problemas išspręsti.

- **Mikroservisų architektūros sampratos ir jos palaikymo problema**. Nors įmonėje yra norima taikyti mikroservisų architektūrą, šiuo metu ji nėra taikoma, todėl nėra žinoma, kaip ją galima būtų pritaikyti vidinėje infrastruktūroje. Siekiant išlaikyti paslaugų teikimą, monolitines sistemas būtų reikalinga perdaryti ir skaldyti į smulkesnius servisus – mikroservisus. Žiniatinklio programa ĮFAM mikroservisų architektūros pagrindu padės įvertinti šios architektūros principus, taip pat jos pritaikymą infrastruktūroje, kuomet vidines įmonės sistemas bus galima panaudoti bei integruoti į šią žiniatinklio programą veikimo ir taip bus galima įvertinti realių procesų veikimą.

## ĮFAM naudotojai

**Administratorius.** Administratorius yra įmonės viduje paskirtas asmuo, kuris yra atsakingas už šios programos administravimą bei stebėjimą.

- **Reikalinga kvalifikacija** – aukštasis išsilavinimas IT srityje.
- **Reikalinga patirtis įmonėje** – darbo su vidinėmis įmonės sistemomis patirtis pagal turimą kompetenciją.
- **Susipažinimas su programinės įrangos veikimu** - reikalingas
- **Programinės įrangos kūrimo, integravimo ir tobulinimo priemonių taikymo patirtis** - reikalinga

**Naudotojai.** Naudotojai yra įmonės darbuotojai, kurie pagal savo kvalifikaciją įmonėje, naudos šią programą su ribota prieiga.

- **Reikalinga kvalifikacija** – aukštasis išsilavinimas IT srityje.
- **Reikalinga patirtis įmonėje** – darbo su vidinėmis įmonės sistemomis patirtis pagal turimą kompetenciją
- **Susipažinimas su programine įrangos veikimu** – reikalingas
- **Programinės įrangos kūrimo, integravimo ir tobulinimo priemonių taikymo patirtis** - nereikalinga

## Žiniatinklio programos apribojimai

**Struktūros apribojimai**

- ĮFAM turi būti žiniatinklio programa ir turi remtis žiniatinklio programos principais.
- ĮFAM turi būti paremta mikroservisų architektūra ir griežtai laikytis jos principų.

**Diegimo apribojimai**

- Žiniatinklio programa turi būti diegiama „Cherry Servers“ nuomojamuose serveriuose.

- Serveriai, kuriuose diegiama žiniatinklio programa, turi būti įdiegta viena iš įmonės siūlomų ir virtualiuose serveriuose palaikomų Linux distribucijų.
- ĮFAM komponentai turi būti talpinami konteineriuose, jog juos būtų galima diegti į konteinerių klasterius.

**Sistemų integravimo apribojimai**

- ĮFAM turi palaikyti vidinėje infrastruktūroje naudojamų sistemų integraciją.
- ĮFAM turi palaikyti šių išorinių sistemų integraciją: MySQL, Docker, Keycloak, Grafana, Zabbix.
- ĮFAM turi turėti modifikavimo galimybę, jog, esant poreikiui, programą būtų galima integruoti į kitas vidinės infrastruktūros sistemas.

**Naudojimo apribojimai**

- Programa turi naudotis tik įmonės darbuotojai, programoje panaudojami duomenys ir pateikiama informacija neturi būti prieinama asmenims, kurie nėra apibrėžti šios žiniatinklio programos naudotojai.

## Žiniatinklio programos funkciniai reikalavimai

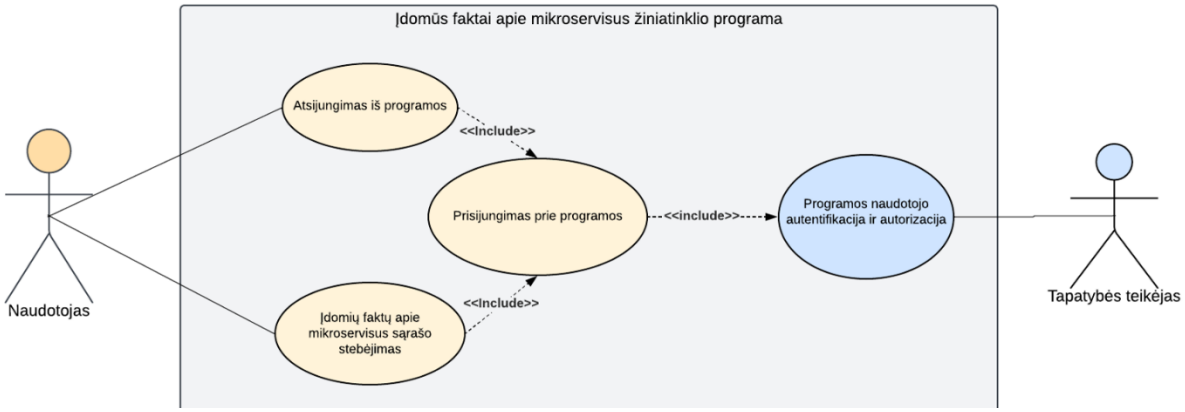
### ĮFAM panaudojimas įmonėje

ĮFAM žiniatinklio programa turi būti kuriama remiantis mikroservisų modeliu. Šios žiniatinklio programos dėka būtų galima analizuoti mikroservisų architektūrą, jos esmines charakteristikas. Negana to, yra būtina atsižvelgti ir į įmonėje vykstančius procesus – kuriamą žiniatinklio programą turi turėti integracijos galimybę, kuomet tiek pačią programą, tiek į ją būtų galima integruoti jau naudojamas sistemas, tokias kaip:

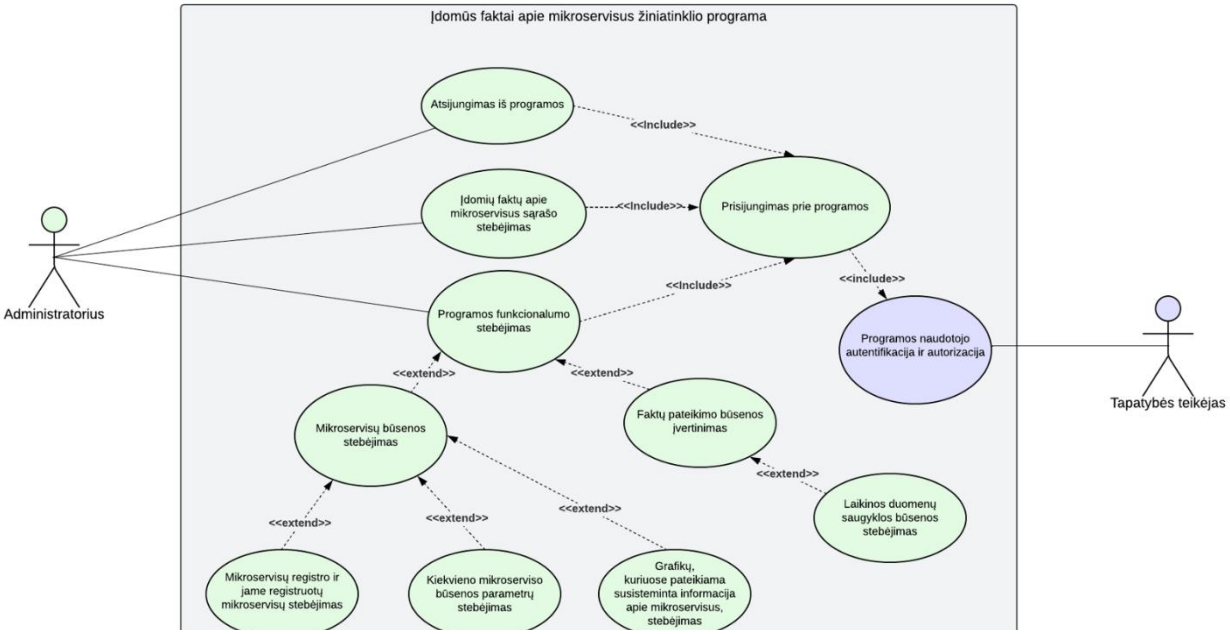
- Reliacinės duomenų bazės (MySQL).
- Stebėjimo ir klaidų fiksavimo priemonės (Grafana, Zabbix).
- Kiti įmonės darbinėje srityje naudojami įrankiai, servisai, komponentai.

### ĮFAM žiniatinklio programos panaudojimo atvejai pagal programos naudotoją

Žemiau yra pateikiami žiniatinklio programos panaudos atvejai pagal naudotojus: Naudotojas (*Žr. 2 pav.*), Administratorius (*Žr. 2 pav.*)




*1 pav. Naudotojo panaudos atvejų diagrama*



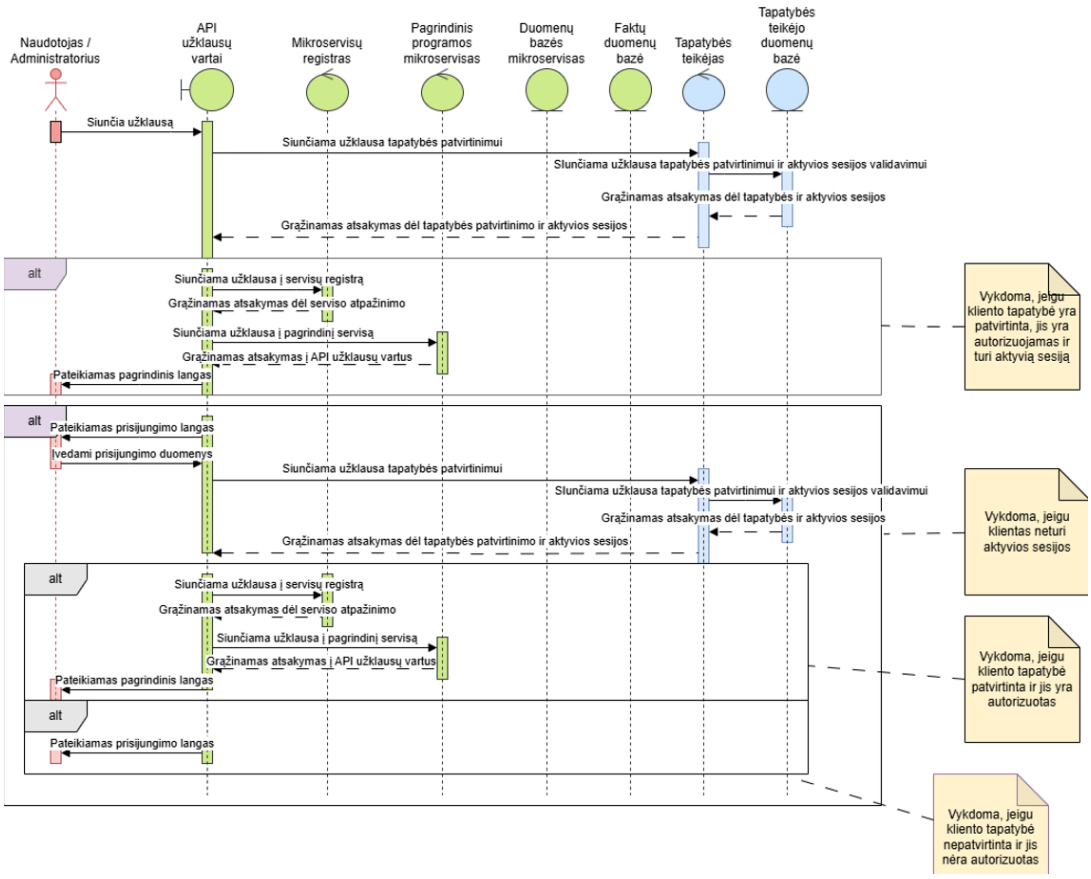

*2 pav. Administratoriaus panaudos atvejų diagrama*

## Reikalavimai duomenims

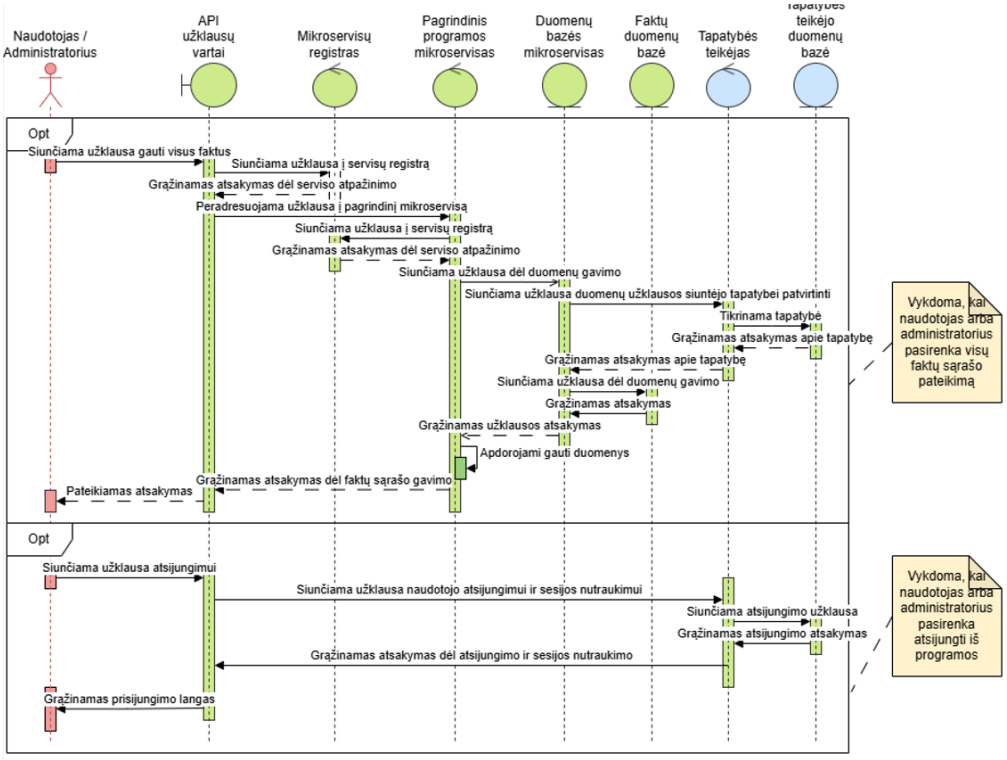
- Žiniatinklio programoje turi būti panaudojama duomenų bazė, kurią tiesiogiai gali pasiekti tik vienas mikroservisas
- Kuriama duomenų bazė turi būti reliacinio pobūdžio. Šis reikalavimas yra apibrėžiamas dėl galimos duomenų kiekio ir pobūdžio kaitos.

## Panaudos atvejų vykdymas laike

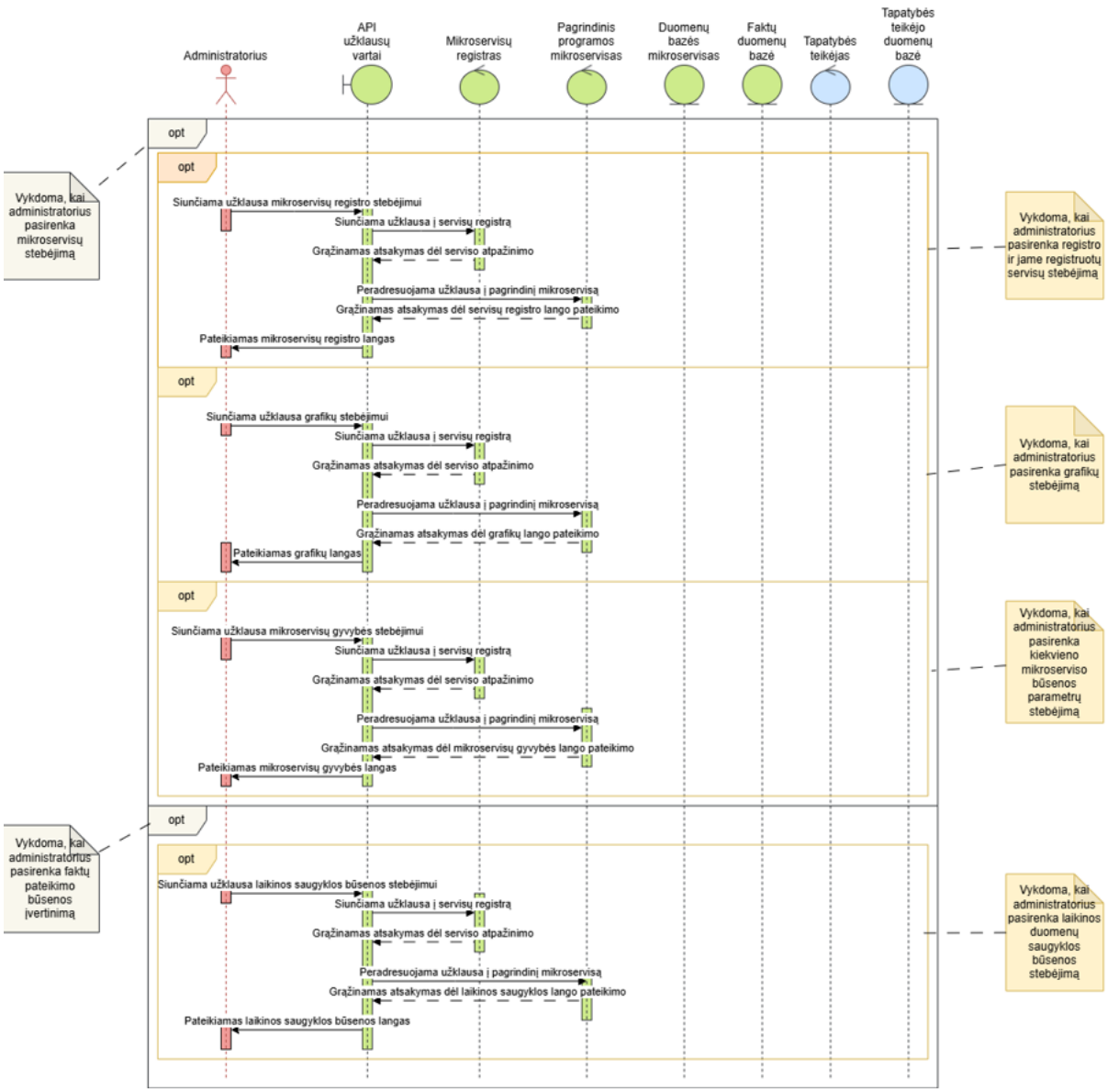
Žemiau yra pateikiamos panaudos atvejų vykdymą laike atvaizduojančios sekų diagramos (*Žr.3,4,5 pav.*)




*3 pav. Naudotojo bei administratoriaus autentifikacijos ir autorizacijos sekų diagrama*



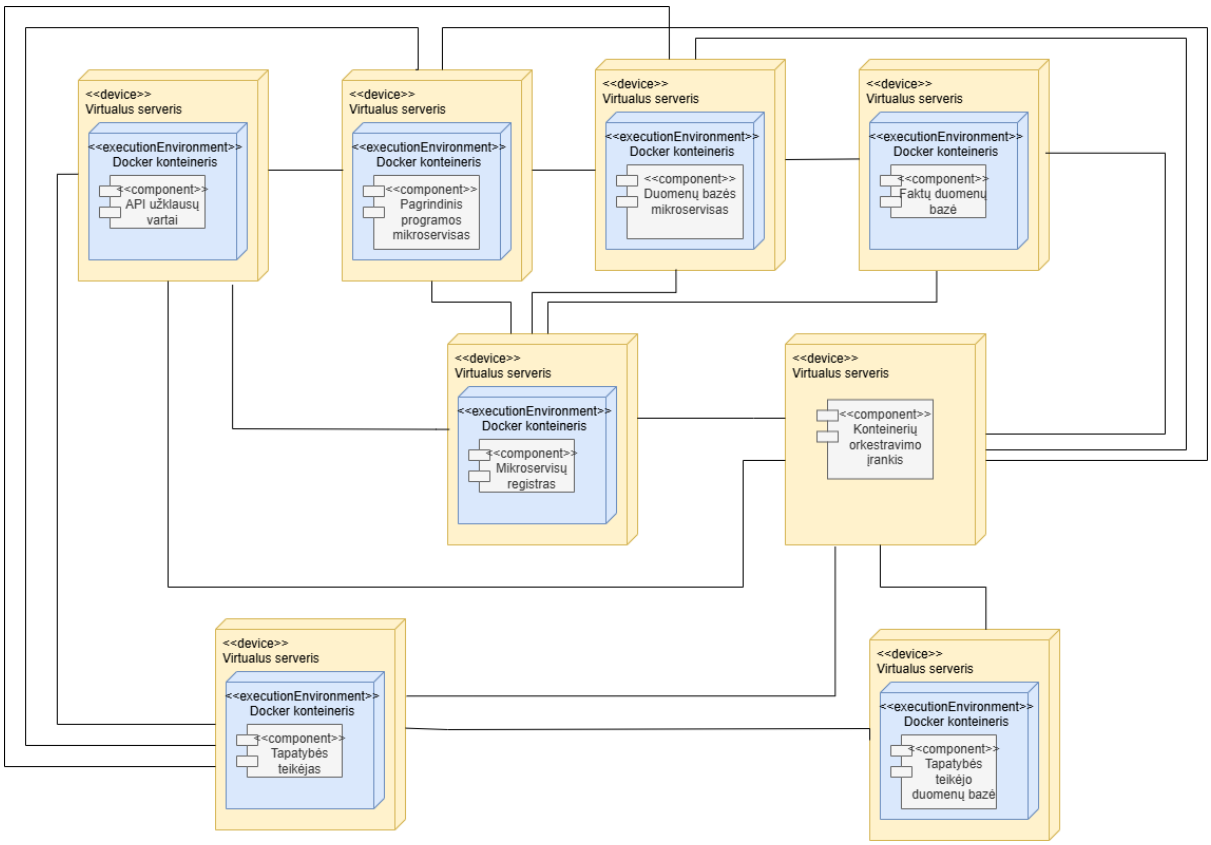

*4 pav. Naudotojo bei administratoriaus sąrašo pateikimo ir atsijungimo sekų diagrama*




*5 pav. Administratoriaus stebėjimo panaudos atvejų sekų diagrama*

## Žiniatinklio programos diegimas serveriuose

Žemiau yra pateikiama diegimo diagrama, kuria galima nusakyti, kaip žiniatinklio programos komponentai turi būti diegiami ir kaip turi vykti komunikacija tarp jų (*Žr. 6 pav.*)




*6 pav. ĮFAM diegimo diagrama*

## Funkciniai reikalavimai

Pagal ĮFAM panaudos atvejus, jų vykdymą laike yra apibrėžiami šie funkciniai reikalavimai:

| **Reikalavimas Nr.1** |
|---|
| **Aprašymas:** Priklausomai nuo žiniatinklio naudotojo grupės, turi būti pateikiamos skirtingos grafinės sąsajos panaudos atvejams atlikti |

| **Reikalavimas Nr.2** |
|---|
| **Aprašymas:** Turi būti įgyvendinta galimybė integruoti išorines sistemas į žiniatinklio programą |

| **Reikalavimas Nr.3** |
|---|
| **Aprašymas:** Žiniatinklio programoje turi būti įgyvendinti duomenų mainai tarp duomenis valdančio mikroserviso ir duomenų bazės |

| **Reikalavimas Nr.4** |
|---|
| **Aprašymas:** Žiniatinklio programą jos naudotojas turi pasiekti tik per vieną kelią |

| **Reikalavimas Nr.5** |
|---|
| **Aprašymas:** Žiniatinklio programoje turi būti įgyvendintas apkrovos balansavimo sistema |

| **Reikalavimas Nr.6** |
|---|
| **Aprašymas:** Turi būti įgyvendinta žiniatinklio programos naudotojo autentifikacija ir autorizacija |

| **Reikalavimas Nr.7** |
| --- |
| **Aprašymas:** Pagal programos naudotojo pasirinkimą ir aktyvios sesijos būseną, žiniatinklio programoje turi būti įgyvendintas sesijos pradžios ir jos nutraukimo mechanizmas |

| **Reikalavimas Nr.8** |
| --- |
| **Aprašymas:** Pagal žiniatinklio programos naudotojo pasirinkimą, grafinėje sąsajoje turi būti atvaizduojamas visų įdomių faktų apie mikroservisus sąrašas |

| **Reikalavimas Nr.9** |
| --- |
| **Aprašymas:** Žiniatinklio programoje kiekvienam mikroservisui ir pagalbiniam servisui turi būti pritaikytas jo gyvybės, veikimo charakteristikų surinkimas |

| **Reikalavimas Nr.10** |
| --- |
| **Aprašymas:** Žiniatinklio programoje turi būti pritaikytas kiekvieno mikroserviso ir pagalbinio serviso gyvybės, veikimo charakteristikų atvaizdavimas |

| **Reikalavimas Nr.11** |
| --- |
| **Aprašymas:** Turi būti įgyvendinta komunikacija tarp mikroservisų ir juos palaikančių servisų |

| **Reikalavimas Nr.12** |
| --- |
| **Aprašymas:** Turi būti sukurta mikroservisų registravimo ir jų būsenos nustatymo sistema |

| **Reikalavimas Nr.13** |
| --- |
| **Aprašymas:** Turi būti įgyvendintas mikroservisų registravimo ir jų būsenos nustatymo sistemos stebėjimas |

| **Reikalavimas Nr.14** |
| --- |
| **Aprašymas:** ĮFAM turi būti įgyvendintas gyvybės, veikimo, būsenos charakteristikų grafinis atvaizdavimas |

| **Reikalavimas Nr.15** |
| --- |
| **Aprašymas:** Žiniatinklio programoje turi būti sukurtas klaidų toleravimo mechanizmas |

| **Reikalavimas Nr.16** |
| --- |
| **Aprašymas:** ĮFAM turi būti pritaikoma laikinoji duomenų saugykla (*Angl. temporary data cache*) |

| **Reikalavimas Nr.17** |
| --- |
| **Aprašymas:** Turi būti įgyvendintas laikinosios duomenų saugyklos būsenos stebėjimas. |

# Žiniatinklio programos nefunkciniai reikalavimai

## Vizualiniai reikalavimai

Šioje dalyje yra pateikiami nefunkciniai reikalavimai, kurie apibrėžia vizualizacijos kriterijus.

### Grafinės sąsajos išvaizdos reikalavimai

- Grafinėje sąsajoje esantys komponentai turi būti aiškiai išskirti iš grafinės sąsajos aplinkos.

### Stiliaus reikalavimai

- ĮFAM turi būti pasirinktas vizualinis stilius, atitinkantis įmonės tematiką, kadangi žiniatinklio programa bus naudojama įmonės darbuotojų. Pasirinktos spalvos: raudona, balta, pilka.

## Panaudojimo reikalavimai

### Paprastumo reikalavimai

- ĮFAM turi būti taip realizuojama, jog būtų aiškiai suprantama, kaip naudotis programos funkcijomis tiek paprastam naudotojui, tiek administratoriui.

### Kalbos reikalavimai

- ĮFAM turi būti pateikiama informacija anglų kalba tam, jog būtų užtikrinamas suprantamumas įmonės darbuotojams, kurie naudojasi šia žiniatinklio programa. Šį reikalavimą įtakoja įmonės darbuotojų įvairovė ir skirtinga gimtoji kalba.

### Suprantamumo reikalavimai

- Žiniatinklio programoje naudotojams turi būti pateikiama tik ta informacija, kuri yra aktuali pagal kvalifikacijos lygį ir įmonės viduje egzistuojančią hierarchiją. Pagal programos naudotojo rolę, neturi būti pateikiama papildoma informacija, kuri gali apsunkti programos naudotojo darbą.

## Kokybės reikalavimai

### Greičio ir uždelsimo reikalavimai

- Komunikacijos tarp žiniatinklio programos naudotojo ir servisų atsako laikas – 2 sekundės. Jeigu atsakymo reikia laukti ilgiau, turi būti fiksuojama, jog serviso, iš kurio tikimasi gauti atsakymą, veikimas yra sutrikdytas.
- Komunikacijos tarp skirtingų servisų atsako laikas – 5 sekundės. Jeigu atsakymo reikia laukti ilgiau, turi būti fiksuojama, jog serviso, iš kurio tikimasi gauti atsakymą, veikimas yra sutrikdytas.
- Komunikacijos tarp servisų ir kritinių resursų atsako laikas – 5 sekundės. Jei atsakymo reikia laukti ilgiau, turi būti fiksuojama, jog komunikacija tarp servisų ir jiems priklausančių kritinių resursų yra sutrikdytas.

- Komunikacijos grandinės pertraukties veikimo pradžia – 2 sekundės po serviso ar kritinio resurso sutrikimo užfiksavimo.
- Servisų registre registruojamų servisų būsena turi būti atvaizduojama ne daugiau kaip su 2 sekundžių paklaida priklausomai nuo realiu laiku esamos servisų būsenos.

**Saugumo reikalavimai**

- ĮFAM turi būti užtikrinama programos naudotojų įvedamų duomenų saugumas. Naudotojų duomenys, kurie yra naudojami autentifikacijos metu, turi būti slaptinami ir šifruojami.
- ĮFAM turi būti užtikrinamas autorizacijos saugumas tiek tarp kliento ir jam prieinamų servisų, tiek tarp servisų, komunikuojančių tarpusavyje. Kiekvienai išorinei užklausai turi būti naudojama OAuth 2.0 autorizacija.
- Žiniatinklio programoje turi būti užtikrinta, jog resursai gali būti prieinami tik atlikus autentifikaciją ir autorizaciją.

**Tikslumo reikalavimai**

- ĮFAM turi būti pateikiama tiksli informacija, kuri yra gaunama užklausų siuntimo metu.

**Patikimumo ir klaidų toleravimo reikalavimai**

- ĮFAM turi būti pritaikytos klaidų toleravimo priemonės, kurios galėtų likviduoti klaidų padarinius, padėtų išvengti galimų klaidų arba sumažintų iškylančių klaidų tikimybę.
- Žiniatinklio programoje turi būti taikomos priemonės, kurios užtikrintų, jog programos veikimas būtų nenutrūkstamas atsiradus klaidoms. Integruojant išorines sistemas, turi būti užtikrinama, jog atsiradus klaidoms ar gedimams, jų padariniai būtų kuo mažesni.

**Juridiniai reikalavimai**

- Žiniatinklio programoje naudojama programinė įranga turi būti atviro kodo ir laisvai prieinama. Jei programinė įranga nėra atviro kodo, ji turi būti licencijuota.

***Priedas Nr. 2***

# Testavimo planas

## Įvadas

Testavimo plane yra aprašomi reikalingi atlikti testavimo atvejai, kurie remiasi kuriamos žiniatinklio programos reikalavimais.

## Testavimo objektai

Testavimai turi būti atliekami žiniatinklio programoje, kuri remiasi mikroservisų modeliu. Dėl mikroservisų architektūros charakteristikų, testavimai turi būti atliekami visiems jos komponentams ir komunikacijai tarp jų: naudojamų mikroservisų, pagalbinių servisų ir išorinių integracijų.

## Testavimo metodologija

Dėl žiniatinklio programos, sukurtos naudojant mikroservisų modelį sudėtingumo, testavimams pasirinkta „pilkos dėžės“ metodologija. Tai yra „baltos dėžės“ ir „juodos dėžės“ metodologijų derinys, kuomet yra testuojamas tiek vidinis žiniatinklio programos ĮFAM veikimas, tiek testuojamas išorinis veikimas, kuris yra matomas išorinio žiniatinklio programos naudotojo.

## Testavimo objekto darbinė sritis

**Testavimo objektas** – žiniatinklio programa, išskaidyta į atskirus komponentus, sąveikaujančius tarpusavyje. Jeigu komponentas yra mikroservisas, jis turi savo apibrėžtą veikimo sritį, yra vienas nuo kito nepriklausomi ir tarpusavyje bendrauja naudodami HTTP API užklausas. Norint patenkinti funkcionalumo poreikius, yra reikalinga sąveika tarp jų. Šiems mikroservisams sąveikauti ir žiniatinklio programos funkcionalumui įgyvendinti ir naudojami pagalbiniai servisai, kurie nėra priskiriami mikroservisams dėl savo mišraus pobūdžio. Kuriamos žiniatinklio programos funkciniai, nefunkciniai reikalavimai, apribojimai ir veikimo principas yra pateikiami reikalavimų specifikacijoje (*žr. priedą nr.1*)

## Testavimų tipai

Testavimams, aprėpiantiems skirtingus aspektus, atlikti yra naudojami šie testavimų tipai:

- **Vienetų testavimas.** Šio testavimo metu yra testuojami mikroserviso arba pagalbinio serviso klasėse esantys metodai. Jie yra uždari, testuojama tik konkretaus metodo veikimo sritis, išoriniai duomenys yra imituojami.

- **Komponentų testavimas.** Komponentų testavimo metu yra testuojami mikroserviso arba pagalbinio serviso komponentai. Komponentams priklauso keli metodai, kurie sąveikauja tarpusavyje. Šio testavimo metu išorinė komunikacija yra imituojama.
- **Integracijos testavimas.** Integracijos testavimo metu yra testuojami atskirų žiniatinklio programos komponentų integracija ir komunikacija tarp jų.
- **Nuo galo iki galo testavimas.** Šio testavimo metu yra tikrinamas bendras žiniatinklio programos veikimas remiantis „juodos dėžės“ metodologija
- **Kokybės testavimas.** Kokybės testavimo metu yra testuojami žiniatinklio programos nefunkciniai reikalavimai.

### Testavimo priemonės

Testavimams atlikti turi būti naudojamos šios priemonės:

- JUnit ir Mockito testavimo karkasai.
- Postman įrankis API užklausų simuliavimui
- Wiremock biblioteka API užklausų simuliavimui

Šios priemonės yra palaikomos žiniatinklio programos karkaso ir yra integruojamos skirtingiems testavimo tipams atlikti.

### Testavimo atvejai

**Vienetų testavimas**

| Testavimo atvejis | Tikėtinas rezultatas |
|---|---|
| 1) Faktų sąrašo gavimas duomenų bazės mikroservise iš duomenų bazės testavimas | 4. Gautų duomenų sąrašas nėra tuščias<br>5. Gautų duomenų sąraše yra tiek elementų, kiek jų yra apibrėžta<br>6. Tikrinant konkretų sąrašo elementą, yra randamas atitikmuo |
| 2) Sėkmingas faktų sąrašo gavimas iš duomenų bazės mikroserviso pagrindiniame programos mikroservise testavimas | 1. Gautų duomenų sąrašas nėra tuščias<br>2. Gaunamas atsakymas su HTTP 200 kodu<br>3. Gautų duomenų sąraše yra tiek elementų, kiek jų yra apibrėžta<br>4. Tikrinant konkretų sąrašo elementą, yra randamas atitikmuo<br>5. Grandinės pertraukties būsena yra uždaryta<br>6. Laikinoje duomenų saugykloje yra kaupiami paskutinės sėkmingos užklausos duomenys<br>7. Duomenys iš laikinosios duomenų saugyklos nėra naudojami<br>8. Pranešimas nėra kuriamas |
| 3) Duomenų panaudojimo iš laikinosios duomenų saugyklos, jei joje yra duomenų, | 1. Gautų duomenų sąrašas yra tuščias<br>2. Gaunamas atsakymas su HTTP 503 kodu |

| | |
|---|---|
| negavus faktų sąrašo iš duomenų bazės mikroserviso testavimas | 3. Laikinoje duomenų saugykloje yra duomenų<br>4. Gautų duomenų nėra<br>5. Tikrinant konkretų sąrašo elementą laikinoje duomenų saugykloje, yra randamas atitikmuo<br>6. Grandinės petraukties būsena yra atidaryta<br>7. Laikinoje duomenų saugykloje nauji duomenys yra nekaupiami<br>8. Yra naudojami duomenys iš laikinosios duomenų saugyklos<br>9. Sukuriamas pranešimas, jog duomenų bazės mikroservisas yra nepasiekiamas ir duomenys yra naudojami iš laikinosios duomenų saugyklos. |
| 4) Faktų sąrašo pranešimų pateikimo, jei negautas atsakymas iš duomenų bazės mikroserviso ir laikinoji duomenų saugykla tuščia, testavimas | 1. Gautų duomenų sąrašas yra tuščias<br>2. Gaunamas atsakymas su HTTP 503 kodu<br>3. Laikinoje duomenų saugykloje nėra duomenų<br>4. Gautų duomenų iš duomenų bazės mikroserviso nėra<br>5. Grandinės petraukties būsena yra atidaryta<br>6. Laikinoje duomenų saugykloje nauji duomenys yra nekaupiami<br>7. Duomenys iš laikinosios duomenų saugyklos nėra naudojami<br>8. Sukuriamas pranešimas, jog duomenų bazės mikroservisas yra nepasiekiamas ir duomenys iš laikinosios duomenų saugyklos nėra naudojami. |
| 5) Faktų sąrašo ir pranešimo pateikimo naudotojo grafinėje sąsajoje testavimas | 1. Į grafinę sąsają yra pridedami tinkami atributai<br>2. Grąžinama tinkama grafinė sąsaja<br>3. Testavimo metu yra iškviečiami reikalingi metodai funkcionalumui atvaizduoti |
| 6) Faktų sąrašo ir pranešimo pateikimo administratoriaus grafinėje sąsajoje testavimas | 1. Į grafinę sąsają yra pridedami tinkami atributai<br>2. Grąžinama tinkama grafinė sąsaja<br>3. Testavimo metu yra iškviečiami reikalingi metodai funkcionalumui atvaizduoti |
| 7) Prieigos prie servisų registro grafinės sąsajos pateikimo pagrindinio programos mikroserviso grafinėje sąsajoje testavimas | Paspaudus ant specialaus mygtuko, yra atveriamas servisų registro grafinės sąsajos langas |

| | |
|---|---|
| 8) Prieigos prie kiekvieno mikroservisų parametrų grafinės sąsajos pateikimo pagrindinio programos mikroserviso grafinėje sąsajoje testavimas | Paspaudus ant specialaus mygtuko, yra atveriamas kiekvieno mikroservisų parametrų (Prometheus) grafinės sąsajos langas |
| 9) Prieigos prie grafikų pateikimo pagrindinio programos mikroserviso grafinėje sąsajoje testavimas | Paspaudus ant specialaus mygtuko, yra atveriamas Grafana grafinės sąsajos langas |
| 10) Laikinosios duomenų saugyklos būsenos atvaizdavimo testavimas | 1. Jeigu duomenys yra tiesiogiai naudojami iš duomenų bazės mikroservisų grąžinto atsakymo, joks pranešimas nėra rodomas<br>2. Jeigu duomenys nėra gaunami iš duomenų bazės mikroserviso, tačiau duomenų laikinoje duomenų bazėje yra, pateikiamas pranešimas, jog duomenų bazės mikroservisas nepasiekiamas ir duomenys yra naudojami iš laikinosios duomenų saugyklos<br>3. Jeigu duomenys nėra gaunami iš duomenų bazės mikroserviso ir laikinoje duomenų saugykloje nėra, pateikiamas pranešimas, jog duomenų bazės mikroservisas yra nepasiekiamas ir laikinoje duomenų saugykloje duomenų nėra |
| 10) Duomenų panaudojimo iš laikinosios duomenų saugyklos pagrindiniame programos mikroservise testavimas | 1. Jeigu pagrindinio programos mikroserviso grandinės pertraukties būsena yra atidaryta arba pusiau atidaryta, naudojami duomenys iš laikinosios duomenų saugyklos<br>2. Jeigu pagrindinio programos mikroserviso grandinės pertraukties būsena yra uždaryta, duomenys iš laikinosios duomenų saugyklos nėra naudojami |
| 11) Grandinės pertraukties pagrindiniame programos mikroservise testavimas | 1. Grandinė yra atidaroma, jei yra gaunamas HTTP 503 kodas iš duomenų bazės mikroserviso<br>2. Grandinė yra visada uždaryta, jei nėra gaunamas HTTP 503 kodas |
| 12) Grandinės pertraukties pagrindiniam programos mikroservisui API užklausų vartuose testavimas | Atidarius grandinę, yra pateikiamas grafinės sąsajos langas, kuriame yra pateikiamas pranešimas, jog šis mikroservisas yra nepasiekiamas |
| 13) Grandinės pertraukties servisų registrui API užklausų vartuose testavimas | Atidarius grandinę pateikiamas grafinės sąsajos langas, kuriame yra pateikiamas pranešimas, jog servisų registras yra nepasiekiamas |
| 14) Grandinės pertraukties Grafana sąsajai API užklausų vartuose testavimas | Atidarius grandinę, pateikiamas grafinės sąsajos langas, kuriame yra pateikiamas pranešimas, jog Grafana yra nepasiekiama |

| | |
|---|---|
| 15) Grandinės pertraukties Prometheus sąsajai API užklausų vartuose testavimas | Atidarius grandinę, yra pateikiamas grafinės sąsajos langas, kuriame yra pateikiamas pranešimas, jog Prometheus yra nepasiekiamas |

**Integracijos testavimas**

| **Testavimo atvejis** | **Tikėtinas rezultatas** |
|---|---|
| 16) Komunikacijos tarp pagrindinio programos mikroserviso ir duomenų bazės mikroserviso testavimas | 1. Pagrindinis programos mikroservisas gauna vidinės komunikacijos prieigos raktą iš tapatybės teikėjo<br>2. Pagrindinis programos mikroservisas siunčia užklausas su Bearer žetonu į duomenų bazės mikroservisą<br>3. Duomenų bazės mikroservisas priima užklausą tik su vidinės komunikacijos žetonu. |
| 17) Komunikacijos tarp API užklausų vartų ir pagrindinio programos mikroserviso testavimas | API užklausų vartai nukreipia naudotojų srautą į pagrindinį programos mikroservisą kartu perduodant prieigos žetoną. |
| 18) Servisų registro komunikacijos su atpažįstamais servisais testavimas | Servisų registras siunčia „širdies dūžius“ į registruojamus servisus ir servisai yra registruojami servisų registre |
| 19) API užklausų vartų komunikacijos su išoriniu tapatybės teikėju testavimas | Žiniatinklio programos naudotojo duomenys yra surenkami ir siunčiami išoriniam tapatybės teikėjui autentifikacijai ir autorizacijai atlikti ir išorinis tapatybės teikėjas grąžina atsakymą su šio naudotojo prieigos žetonu. |

**Komponentų testavimas**

| **Testavimo atvejis** | **Tikėtinas rezultatas** |
|---|---|
| 20) Pagrindinio programos mikroserviso naudotojo grafinės sąsajos pateikimas su reikalingais duomenimis ir elementais | 1. Grafinės sąsajos pateikimo metodas, vidinės komunikacijos prieigos rakto gavimo metodas, užklausos siuntimo į duomenų bazės mikroservisą metodas veikia kartu.<br>2. Naudotojo grafinėje sąsajos pateikimo metu yra pateikiamas įdomių faktų apie mikroservisus sąrašas, yra pateikiami reikalingi pranešimai ir yra sukuriamas mygtukas atsijungimui<br>3. Vidinės komunikacijos prieigos rakto gavimo metodo vykdymo metu yra gaunamas prieigos žetonas<br>4. Užklausos į duomenų bazės mikroservisą metodo vykdymo metu yra grąžinamas faktų sąrašas ir pranešimas |
| 21) Pagrindinio programos mikroserviso administratoriaus grafinės sąsajos | 1. Grafinės sąsajos pateikimo metodas, vidinės komunikacijos prieigos rakto gavimo metodas, užklausos siuntimo |

| | |
|---|---|
| pateikimas su reikalingais duomenimis ir elementais | į duomenų bazės mikroservisą metodas veikia kartu.<br>2. Naudotojo grafinėje sąsajoje yra pateikiamas įdomių faktų apie mikroservisus sąrašas, yra pateikiami reikalingi pranešimai, yra sukuriami mygtukai stebėjimo galiniams taškams pasiekti ir yra sukuriamas atsijungimo mygtukas<br>3. Vidinės komunikacijos prieigos rakto gavimo metodo vykdymo metu yra gaunamas prieigos žetonas Užklausos į duomenų bazės mikroservisą metodo vykdymo metu yra grąžinamas faktų sąrašas ir pranešimas |

**Nuo galo iki galo (*angl. end-to-end*) testavimas**

| **Testavimo atvejis** | **Tikėtinas rezultatas** |
|---|---|
| 22) Žiniatinklio programos bendro veikimo iš programos naudotojo perspektyvos testavimas | Žiniatinklio programa iš naudotojo perspektyvos atlieka šį funkcionalumą: pateikiama prisijungimo grafinę sąsają, prisijungus yra pateikiama grafinė sąsaja, kurioje galima pasirinkti visų faktų sąrašo ir laikinosios duomenų saugyklos būsenos stebėjimo pateikimą, servisų registro stebėjimą, veikiančių servisų parametrų stebėjimą, grafikų stebėjimą, taip pat yra mygtukas, leidžiantis nutraukti sesiją (atsijungti). |

**Kokybės testavimas**

| **Testavimo atvejis** | **Tikėtinas rezultatas** |
|---|---|
| 23) Apkrovos tarp pagrindinio programos mikroserviso veikiančių kopijų testavimas | Apkrova yra sėkmingai paskirstoma tarp pagrindinio programos mikroserviso veikiančių kopijų ir srautas yra nukreipiamas į veikiančias kopijas, jei sutrinka veikimas vienos iš kopijų. |
| 24) API užklausų vartų pasiekiamumo naudojant API užklausas testavimas | API užklausų vartai yra pasiekiami naudojant API užklausas, tačiau yra pateikiamas prisijungimo langas. |
| 25) API užklausų vartų pasiekiamumo naudojant grafinę sąsają testavimas | 1. Jeigu yra aktyvi naudotojo sesija, žiniatinklio programos naudotojui yra pateikiamas grafinės sąsajos langas, į kurį kreipėsi<br>2. Jeigu nėra aktyvios sesijos, yra pateikiamas prisijungimo langas |
| 26) Pagrindinio programos mikroserviso pasiekiamumo naudojant API užklausas testavimas | 1. Jeigu užklausos siuntimo metu yra pridedamas aktyvus prieigos žetonas, pagrindinis programos mikroservisas yra pasiekiamas, gaunamas HTTP 200 kodas. |

| | |
|---|---|
| | 2. Jeigu užklausos siuntimo metu yra nepateikiamas prieigos žetonas, yra grąžinamas HTTP 401 kodas.<br>3. Jeigu užklausos siuntimo metu yra pateikiamas klaidingas prieigos žetonas, grąžinamas HTTP 401 kodas |
| 27) Pagrindinio programos mikroserviso pasiekiamumo naudojant grafinę sąsają testavimas | Naudojant grafinę sąsają, pagrindinis programos mikroservisas yra tiesiogiai nepasiekiamas, yra grąžinamas HTTP 401 kodas. |
| 28) Duomenų bazės mikroserviso pasiekiamumo naudojant API užklausas testavimas | 1. Jeigu užklausos siuntimo metu yra pridedamas vidinės komunikacijos kliento (app-mgmt) prieigos žetonas, duomenų bazės mikroservisas yra pasiekiamas, grąžinamas HTTP 200 kodas.<br>2. Jeigu užklausos siuntimo metu yra nepateikiamas prieigos žetonas, yra grąžinamas HTTP 401 kodas.<br>3. Jeigu užklausos siuntimo metu yra pateikiamas klaidingas prieigos žetonas, grąžinamas HTTP 401 kodas,<br>4. Jeigu užklausos siuntimo metu yra pateikiamas žiniatinklio programos naudotojo (web-app) prieigos žetonas |
| 29) Duomenų bazės mikroserviso pasiekiamumo naudojant grafinę sąsają testavimas | Naudojant grafinę sąsają, pagrindinis programos mikroservisas yra nepasiekiamas, yra grąžinamas HTTP 401 kodas. |
| 30) Pagrindinio programos mikroserviso naudotojo grafinėje sąsajoje atvaizduojamų komponentų testavimas | Grafinėje sąsajoje yra atvaizduojami visi tai grafinei sąsajai priskirti komponentai ir jie yra vizualizuojami pagal nefunkcinius reikalavimus. |
| 31) Pagrindinio programos mikroserviso administratoriaus grafinėje sąsajoje atvaizduojamų komponentų testavimas | Grafinėje sąsajoje yra atvaizduojami visi tai grafinei sąsajai priskirti komponentai ir jie yra vizualizuojami pagal nefunkcinius reikalavimus. |

| | |
|---|---|
| 32) Grandinės petraukties veikimo API užklausų vartuose testavimas | Pasiekus nustatytą nepavykusių siunčiamų užklausų ribą, grandinė tampa atidaryta. Toliau yra tikrinamas nepavykusių užklausų kiekis ir pagal jį atitinkamai yra keičiama grandinės pertraukties būsena. |
| 33) Grandinės petraukties veikimo pagrindiniame programos mikroservise testavimas | Pasiekus nustatytą nepavykusių siunčiamų užklausų ribą, grandinė tampa atidaryta. Toliau yra tikrinamas nepavykusių užklausų kiekis ir pagal jį atitinkamai yra keičiama grandinės pertraukties būsena. |