

VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS STUDIJŲ PROGRAMA

Pagrindiniai komunikavimo integracijų mikroservisų
architektūrose tipai ir jų analizė

Main communication integration types in microservices
architectures comparison and analysis

Kursinis darbas

Atliko: Lukas Milašauskas (parašas)

Darbo vadovas: Dr. Saulius Minkevičius (parašas)

Vilnius – 2020

TURINYS

ĮVADAS	3
Temos aktualumas	3
Problema.....	3
Darbo tikslas.....	3
Uždaviniai tikslui pasiekti	3
1. KAS YRA MIKROSERVISAI	4
1.1. Monolitinės sistemos ir mikroservisų atsiradimas.....	4
1.2. Mikroservisų pranašumai prieš monolitus.....	4
1.2.1. Technologijų nevienalytiškumas (angl. „ <i>Technology Heterogeneity</i> “)	4
1.2.2. Atsparumas (angl. „ <i>Resilience</i> “).	5
1.2.3. Plečiamumas (angl. „ <i>Scaling</i> “).	5
1.2.4. Lengvas diegimas (angl. „ <i>Ease of Deployment</i> “).	5
1.2.5. Organizacinis pasiskirstymas (angl. „ <i>Organizational Allignment</i> “)	5
1.2.6. Kompozicija (angl. „ <i>Composability</i> “)	5
1.2.7. Optimizuotas pakeičiamumas (angl. „ <i>Optimizing for Replaceability</i> “)	5
1.3. Monolitinių sistemų skaidymas į mikroservisus	6
2. MIKROSERVISŲ SISTEMOS VIDINIŲ INTEGRACIJŲ TIPAI	7
2.1. Tarnybų komunikavimas per duomenų bazę.....	8
3. SINCHRONINĖS (ANGL. „ <i>SYNCHRONOUS</i> “) INTEGRACIJOS	9
3.1. Sinchroninių integracijų principas	9
3.2. Sinchroninių integracijų technologijos.....	10
4. ASINCHRONINĖS (ANGL. „ <i>ASYNCHRONOUS</i> “) INTEGRACIJOS	12
4.1. Asinchroninių integracijų principas	12
4.2. Asinchroninių integracijų technologijos	15
5. SKIRTINGŲ INTEGRACIJŲ TIPŲ PRIVALUMAI IR TRŪKUMAI	16
5.1. Silpnas sujungimas ir stiprus sanglauda (angl. „ <i>Loose Coupling and High Cohesion</i> “) ...	16
5.2. Efektyvumas	16
5.3. Implementacijos kompleksiskumas.....	16
5.4. Veiksmų istorija	17
5.5. Įvykių sekos užtikrinimas	17
5.6. Tarnybų kompozicija	17
Tyrimas	17
6. REZULTATAI IR IŠVADOS	18
ŠALTINIAI	20

Įvadas

Temos aktualumas

Tobulėjant programinės įrangos (toliau PĮ) kūrimo įrankiams ir vis augant informacinių sistemų sudėtingumo poreikiams, kyla daug klausimų PĮ kūrėjams, kokį architektūros modelį ir kokiais technologijas pasirinkti, pradedant kurti naują informacinę sistemą. Vis daugėja skirtingų technologijų ir programavimo kalbų, kurios yra pranašesnės už kitas tik siaurose srityse, dėl to dažnu atveju neužtenka pasirinkti vieną technologiją ar programavimo kalbą norint sukurti kokybišką ir tvirtą PĮ. Kai kurios technologijos yra pranašesnės resursų taupyme, kitos pranašesnės daug skirtingų bibliotekų palaikymu ir lengvai naudojama programų (angl. „*application*“) programavimo sąsaja (angl. „*Application programming interface*“ arba „*API*“) ir t.t. Kuriant naują PĮ reikia gerai išsianalizuoti tuo metu esamas technologijas ir jų privalumus. Įmonės yra linkusios kurti PĮ tokiomis technologijomis, kurių specialistų yra daug ir kurie norėtų palaikyti ir kurti jomis. Renkantis skirtingas technologijas, atsiranda problema, kaip jas apjungti, kad veiktų vieningai. Tokiu atveju, galima naudotis saitynų tarnybų pagalba (angl. „*Web services*“), tačiau to neužtenka, nes norima skirtingus funkcionalumus įgyvendinti skirtingų technologijų pagalba. Tada dažnu atveju kuriamos mikroservisų (angl. „*Microservices*“) sistemos. Remiantis šiuo modeliu būtų kuriami atskiri moduliai (taip pavadinsime atskirus sistemos vienetus), kurių kiekvienas būtų atsakingas už tam tikrą funkcionalumą. Problema ta, kad moduliai yra skirtingi, tačiau jie turi sąveikauti tarpusavyje, o tai pasiekti gali būti sudėtinga.

Problema

Kokį integracijų tipą rinktis mikroservisų architektūrose, norint įgyvendinti komunikaciją tarp skirtingų tarnybų (angl. „*service*“)?

Darbo tikslas

Palyginti skirtingus komunikacijų tipus, apžvelgti jų pranašumus ir trūkumus. Pateikti situacijų, kur vieni tipai yra pranašesni už kitus, pavyzdžių ir technologijų, šiems tipams realizuoti.

Uždaviniai tikslui pasiekti

1. Remiantis literatūra apibūdinti, kas yra mikroservisai, kuo jie ypatingi, kaip jie projektuojami.
2. Išskirti pagrindinius integracijų ir komunikavimo tipus ir jų savybes.
3. Palyginti skirtingus komunikavimo būdus mikroservisų architektūrose.
4. Pateikti konkrečius komunikavimo integracijų ir technologijų pavyzdžius.
5. Pateikti rekomendacijas, kokiais atvejais, kokius tipus būtų geriau naudoti.

1. Kas yra mikroservisai

1.1. Monolitinės sistemos ir mikroservisų atsiradimas

Pagal autorių Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzarra publikuotą straipsnį „Microservices: yesterday, today, and tomorrow“ [DGL⁺17] jau 1960-iais buvo susiduriama su problemomis susijusiomis su didžiulio masto PĮ kūrimu ir kaip tai projektuoti. Buvo kuriama daug būdų kaip tai daryti, ir daug teorijų kaip turėtų atrodyti PĮ kodas, ir kaip projektuoti informacines sistemas. Daug vėliau, apie 2000 metus, susiformavo sąvokos „Service-Oriented Computing“ (toliau SOC) ir „Service-Oriented Architecture“ (toliau SOA), kurių idėja ir paradigmos buvo apie tai, kad programa, arba kitaip pavadinus tarnyba, turėtų būti atsakingas už konkretaus resurso ar verslo logikos informaciją. Šią tarnybą turi būti galima pasiekti su konkrečia technologija ir taip komunikuoti ir gauti informaciją. Taigi, remiantis jau anksčiau minėtu straipsniu „Microservices: yesterday, today, and tomorrow“ [Mis7] iš SOC ir SOA kiek vėliau, susiformavo mikroservisų idėja ir paradigmos. Pagal Martin Fowler ir James Lewis 2016-ais metais publikuotą straipsnį „Microservices“ [FL14] šis terminas „mikroservisai“ buvo pirmą kartą diskutuotas 2011 metais, Venecijoje vykusiose PĮ kūrimo architektų dirbtuvėse (angl. „workshop“). Po metų, ta pati grupė architektų nusprendė, kad labiausiai tinkantis pavadinimas šiam architektūriniam tipui yra mikroservisai (angl. „microservices“). Po šiuo terminu slypi daug idėjų ir paradigmų, tačiau pagrindinė mintis yra skaidymas didelės sistemos į mažas „granules“ ir mažas tarnybas. Taip pat norint apibūdinti senas sistemas, kurios buvo nedalomos ir paleidžiamos vienu vykdomuoju paketu, atsirado terminas „monolitas“ (angl. „monolith“). Šis terminas ir buvo naudojamas Unix bendruomenės jau ilgą laiką iki mikroservisų susikūrimo. Remiantis Eric Steven Raymon knyga „The Art of UNIX Programming“ [Ray03], kuri buvo išleista 2003 metais, terminas „monolitas“ buvo naudojamas apibūdinti sistemoms, kurios yra per didelės. Taigi, šie du terminai naudojami iki šiol apibūdinti informacinių sistemų architektūrinėms charakteristikoms.

1.2. Mikroservisų pranašumai prieš monolitus

Pagrindinius principus ir mikroservisų pranašumus detaliai išdėstė Sam Newman savo knygoje „Building Microservices: Designing Fine-Grained Systems“ [New15]. Šiame leidinyje autorius išgrynina keletą labai svarbių mikroservisų privalumų.

1.2.1. Technologijų nevienalytiškumas (angl. „*Technology Heterogeneity*“)

Kiekviena tarnyba turi atlikti skirtingas funkcijas ir turėti skirtingas atsakomybes. Norint pasiekti geriausius rezultatus galima rinktis skirtingas technologijas, kurios būtų geriausiai pritaikytos konkrečiam uždaviniui spręsti. Todėl mikroservisuose kiekvieną tarnybą galim projektuoti skirtingomis technologijomis, pavyzdžiui skirtingomis programavimo kalbomis.

1.2.2. Atsparumas (angl. „**Resilience**“)

Atsitikus problemai ir sugriuvus vienai sistemos komponentei, monolitinėje sistemoje tektų perkraudinėti arba taisyti visą sistemą. Mikroservisų architektūroje stambių pasikeitimų būtų išvengta ir žlugtų tik vienos tarnybos veikimas. Tokiu atveju kitos tarnybos apie tai nežinotų ir veiktų toliau, o norint ištaisyti problemą, užtektų sutaisyti ir perkrauti vieną tarnybą.

1.2.3. Plečiamumas (angl. „**Scaling**“)

Stambioje monolitinėje sistemoje visos plečiamumo ir efektyvumo problemos sprendžiamos kartu. Mikroservisų sistemoje kiekvieną tokio tipo problemą sprendžiame atskirose tarnybose. Tokiu atveju tarnyboms, kurios reikalauja mažiau resursų galima suteikti mažiau, o sunkesnioms ir mažiau efektyvioms tarnyboms išskirti daugiau. Tačiau verta paminėti, kad mikroservisų sistemos plečiamumas ne visada yra geras, tai aprašyta Omar Al-Debagy ir Peter Martinek straipsnyje „A Comparative Review of Microservices and Monolithic Architecture“ [AM18]

1.2.4. Lengvas diegimas (angl. „**Ease of Deployment**“)

Vystant PĮ dažnai susiduriama su diegimo problema. Su naujais funkcionalumais reikia iš naujo diegti naują PĮ versiją. Monolitinėje sistemoje tenka iš naujo sudiegti visą sistemą, tačiau mikroservisų sistemoje galima sudiegti tik tas tarnybas, kurios yra susijusios su pakeitimais.

1.2.5. Organizacinis pasiskirstymas (angl. „**Organizational Alignment**“)

Dažnai įmonėse prie informacinės sistemos dirba daug žmonių. Jie būna pasiskirstę komandomis ir turi skirtingas atsakomybes. Mikroservisų sistemose galima išvengti komunikavimo incidentų ir kiekvienai komandai dirbti su skirtingomis tarnybomis. Tokiu pavydžiu dirba stambi informacinių technologijų (toliau IT) įmonė „Netflix“.

1.2.6. Kompozicija (angl. „**Composability**“)

Įmonės dažnai susiduriama su problema, kai tas pats funkcionalumas reikalingas keliose informacinėse sistemose. Mikroservisų architektūroje, kadangi sistema susideda iš mažų autonomiškų tarnybų, jas galima atskirti ir perpanaudoti skirtingose sistemose, arba kitai sistemai suteikti prieigą tik prie konkrečių resursų, o ne visos sistemos.

1.2.7. Optimizuotas pakeičiamumas (angl. „**Optimizing for Replaceability**“)

Dirbant vidutinio dydžio arba didelėse imonėse dažnai susiduriama su problema, kai naudojama sena kodo bazė ir pasenusios technologijos. Dažnai tokią sistemą reikia atnaujinti siekiant efektyvumo arba palaikymo paprastumo. Tokiu atveju, norint atnaujinti bibliotekas arba technologijas, tenka iš naujo perprogramuoti dalį sistemos. Mikroservisų architektūros pagalba, tai tampa žymiai paprasčiau, kai užtenka perrašyti konkrečią tarnybą, neliečiant likusios informacinės sistemos.

1.3. Monolitinių sistemų skaidymas į mikroservisus

Paskutinį dešimtmetį tapo gan populiariu stambaus masto monolitus skaidyti į mikroservisus, tačiau tai nėra taip paprasta. Visų pirma skaidant monolitą į atskiras tarnybas labai svarbu identifikuoti, kokios mažesnės tarnybos bus. Tarnybų riboms apibrėžti panaudojau Micheal C. Feathers knygoje „Working Effectively with Legacy Code“ [Fea04] apibrėžtą terminą „siūlė“ (angl. „seam“). Siūlė šiuo atveju yra kodo dalis, kuri yra izoliuota ir autonomiška. Siūlės ir bus mūsų atskiri mikroservisai. Autorė Susan J. Fowler savo knygoje „Production-Ready Microservices“ [Fow17] aprašė patarimus ir žingsnius, kaip reikėtų skaidyti monolitą į mikroservisus. Ji pamini, kad tai reikėtų daryti etapais:

1. Monolitinę programą paleisti su tiek kopijų, kiek turėsime siūlių.
2. Išskirstyti kvietimus į kopijas, pagal tai kokias siūles kopijos reprezentuoja.
3. Išvalyti programų kopijas, paliekant tik siūlių funkcionalumą.

Verta paminėti, kad po kiekvieno išvardinto žingsnio būtų atliekami regresiniai testavimai, kurie patikrintų ar sistema veikia korektiškai.

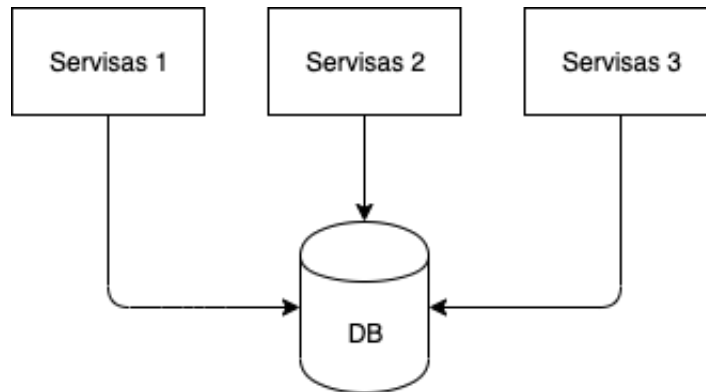
2. Mikroservisų sistemos vidinių integracijų tipai

Pagal Sam Newman knygą „Building Microservices: Designing Fine-Grained Systems“ [New15] vienas iš pagrindinių integracijų projektavimo apsektų yra stengtis išvengti nesuderinamų pakeitimų (angl. „*breaking changes*“). Norint išvengti tokių pakeitimų, reikia rinktis tokį technologinį sprendimą, kad pakeitus vieno mikroserviso grąžinamų duomenų struktūrą, kitos tarnybos be pakeitimų veiktų ir galėtų gauti duomenis. Sam Newman teigimu „tinkamos integracijos pasirinkimas yra pats svarbiausias technologinis su mikroservisais susijęs dalykas“. Todėl labai svarbu yra pasirinkti tinkamą integracijų tipą, nes nuo to priklauso kiek daug problemų sukels naujų funkcionalumų kūrimas. Autorius išskiria keletą skirtingų mikroservisų komunikavimo tipų, kuriuos ir aptarsime:

1. Tarnybų jungimas per duomenų bazę.
2. Sinchroninės užklauskos/atsakymo (angl. „*request/response*“) integracijos.
3. Asinchroninės įvykiais paremtos (angl. „*event-based*“) integracijos.

2.1. Tarnybų komunikavimas per duomenų bazę

Šis integracinis tipas yra paremtas vienos bendros duomenų bazės naudojimu per keletą skirtingų mikroservisų. Šis tipas ypatingas tuo, kad kiekvienas mikroservisas turi priėjimą prie tų pačių resursų, tik modifikuoja resursus už juos atsakingi mikroservisai. Mikroservisų sistemos komunikavimo per duomenų bazę schema (1 pav.):



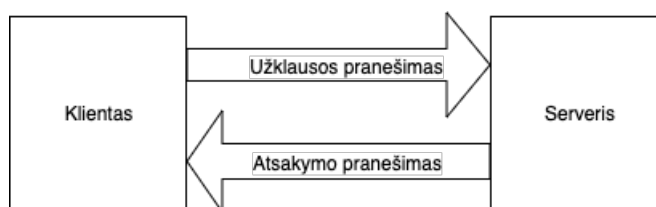
1 pav. Mikroservisų komunikavimas duomenų bazės pagalba.

Svarbu paminėti, jog toks komunikacijų tipas yra labai nesaugus ir duomenų migravimas kuriant stambesnius funkcionalumus yra neišvengiamas. Pačio Sam Newman, knygos [New15] autoriaus, nuomone, tai yra praščiausias komunikavimo būdas ir stipriai nerekomenduojamas. Dėl šios priežasties komunikavimo per bendrą duomenų bazę daugiau šiame darbe nenagrinėsime.

3. Sinchroninės (angl. „*synchronous*“) integracijos

3.1. Sinchroninių integracijų principas

Sinchroninės integracijos dar gali būti apibūdinamos kaip „dviejų žmonių komunikavimas realiu laiku“ [HKL09]. Su sinchroniniais komunikavimo pavyzdžiais dažnai susiduriame visi. Vienas iš tokių yra apsilankymas paprastoje internetinėje svetainėje. Interneto naršyklėje suvedus puslapio pavadinimą, HTTP protokolo pagalba, mums iš serverio yra užkraunamas svetainės turinys ir atvaizduojamas. Šis komunikavimo būdas yra paremtas užklauso/atsakymo (angl. „*request/response*“) principu. Šiame modelyje egzistuoja klientas (tai yra mūsų naršyklė) ir serveris (tai yra interneto svetainės savininkas arba įmonė teikianti svetainių talpinimo paslaugas). Klientas siunčia užklausą serveriui prašydamas resurso, šiuo atveju tai yra mūsų norimos pamatyti svetainės turinio, ir laukia kol svetainė atiduos atsakymą į užklausą. Serveris reaguodamas į kliento užklausą grąžina atsakymą, kuris turi būti arba svetainės turinys, arba grąžinama klaida, pavyzdžiui, kad toks resursas neegzistuoja arba klientas yra neautorizuotas šios svetainės lankytojas. Šis bendravimas yra perteiktas pateikta schema (2 pav.):



2 pav. Sinchroninio komunikavimo schema.

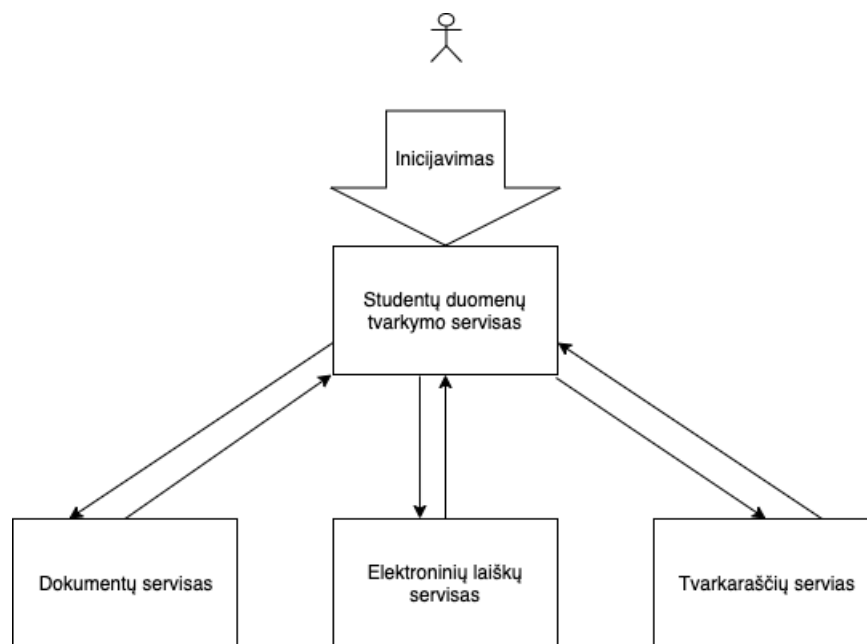
Taigi šis modelis puikiai gali veikti ir mikroservisų sistemoje. Viena tarnyba siunčia užklausą į kitą tarnybą norėdama gauti informaciją apie resursą arba inicijuoti, kokį nors veiksmą. Šis modelis ypatingas tuo, kad yra primityvus ir iškart užklausą išsiuntusi tarnyba žinos ar sėkmingai pavyko atlikti norimą veiksmą. Šis modelis yra labai geras, kai tik įvykus sėkmingai užklausiai leidžiame vartotojui vykdyti kitas operacijas ir kitaip bendrauti su mūsų sistema, tai yra vartotojo autentifikacija ir autorizacija.

Siekant geriau išaiškinti, kaip veikia sinchroninis komunikavimas mikroservisų sistemose, pateikiama modelinė situacija, kur komunikavimas vyktų būtent minėtu būdu. Įsivaizduokime, kad egzistuoja universiteto informacinė sistema, kuri yra suprojektuota pasiremiant mikroservisų architektūriniu modeliu. Šioje sistemoje yra atskiros tarnybos atsakingos už studentų duomenų tvarkymą, dokumentų generavimą, elektroninių laiškų siuntimą, tvarkaraščių sudarymą ir saugojimą. Be šių tarnybų sistema turėtų turėti ir daug kitos paskirties mikroservisų, bet dabar aptarsime tik šiuos. Taigi, sinchroninių komunikavimo modelių tarnyba atsakinga už studentų duomenų tvarkymą bus informuota apie naujo studento kūrimo inicijavimą. Ši tarnyba savo duomenų bazėje sukuria naują įrašą ir vykdo tokius veiksmus eilės tvarka:

1. Siunčiama užklausa į dokumentų generavimo tarnybą, kad būtų gauti sugeneruoti aktai apie naujo studento užregistravimą sistemoje, ir laukiama atsakymo. Dokumentų tarnyba sugeneravusi dokumentą grąžintų atsakymą apie sėkmingą dokumento sugeneravimą.

2. Gavus atsakymą iš dokumentų tarnybos, siunčiama užklausa į tvarkaraščių generavimo tarnybą ir laukiama atsakymo. Tvarkaraščių generavimo tarnyba formuoja studento tvarkaraštį ir grąžina atsakymą apie sėkmingą resurso sukūrimą.
3. Po sėkmingo tvarkaraščio sugeneravimo siunčiama užklausa į elektroninių laiškų tarnybą, siekiant informuoti naujo studento būsimus dėstytojus apie naują užsiėmimų dalyvį. Elektroninių laiškų tarnyba išsiuntusi visus laiškus grąžina atsakymą apie sėkmingai užbaigtą darbą.
4. Be sistemos sutrikimų sėkmingai įvykdžius visus procesus, tarnyba atsakinga už studentų duomenų tvarkymą užbaigia naujo studento kūrimo procesą.

Tokią veiksmų seką vaizdžiai parodo ši schema, kur veiksmai vyksta paeiliui, o ne lygiagrečiai (3 pav.):



3 pav. Sinchroninio komunikavimo mikroservisuose schema.

3.2. Sinchroninių integracijų technologijos

Būdų realizuoti sinchroninius komunikavimo modelius yra daug. Šiuo metu populiariausi yra du:

- RESTful saityno tarnybos.
- SOAP saityno tarnybos.

Turint omenyje, kad REST yra tik architektūrinis stilius, o ne protokolas, ir jų ne-
labai galima lyginti [Loi18]. Tačiau remiantis Joni Makkonen magistrinio darbo „REST ir
SOAP saityno tarnybų efektyvumo ir naudojamumo palyginimas“ [Mak17] galima teigti,
kad REST yra pranašesnė ir šiame darbe išsiplėsime tik su šiuo architektūriniu stiliumi.

REST saityno tarnybos užklausos struktūra yra paprasta. Ji susideda iš keletos atributų:

- HTTP metodo, pvz.: „*GET*“, „*POST*“, „*PUT*“, „*DELETE*“.
- Unikalaus resurso identifikatoriaus (arba adreso), pvz.: „*http://mif.vu.lt/*“.
- Antraščių (angl. „*Headers*“), pvz.: „*Content-Type: application/json*“.
- Užklausos turinio (čia gali būti bet koks standartinis formatas, pvz.: „*JSON*“).
- HTTP protokolo versija, pvz.: „*HTTP/1.1*“.

REST atsakymo struktūra skiriasi nuo užklausos tik tuo, kad vietoje HTTP metodo, gau-
namas HTTP statuso kodas. Tokio komunikavimo per REST saityno tarnybas pavyzdį galime
pamatyti šioje schemoje (4 pav.):

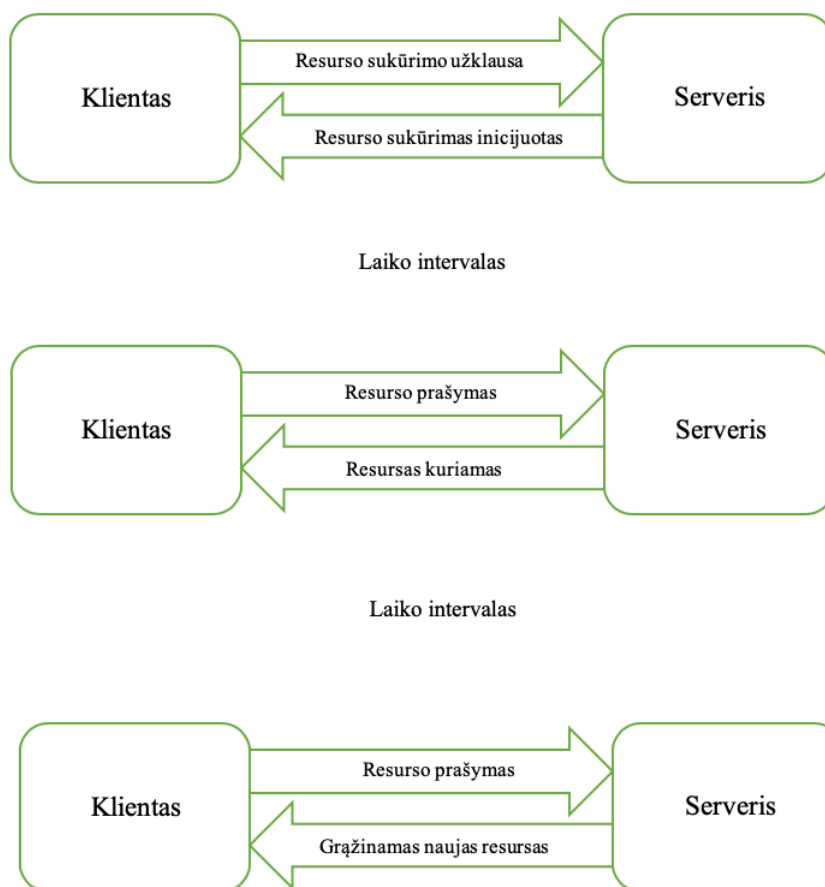


4 pav. REST saityno tarnybų schema.

4. Asinchroninės (angl. „*asynchronous*“) integracijos

4.1. Asinchroninių integracijų principas

Asinchroninės integracijos yra vis labiau populiarėjančios ir vis dažniau sutinkamos informacinėse sistemose. Tokio tipo komunikavimas yra labai tinkamas ilgiems procesams ir daug žingsnių turinčioms operacijoms [New15]. Pačio integracijos tipo principą galima būtų apibūdinti dviem žodžiais: „nelaukianti atsakymo“. Prisimenant jau aptartą serverio ir kliento modelį, sinchroninių integracijų metu klientas siunčia užklausą ir laukia atsakymo iš serverio. Asinchroninių integracijų metu, klientas siunčia užklausą ir priklausomai nuo naudojamos technologijos serveris gali grąžinti arba paprastą atsakymą, kad užregistravo užklausą, arba išvis nieko nepranešti. Tokiu būdu klientas dar negauna resurso, kurio jam reikia, ir neblokuojamas gali vykdyti kitus procesus arba siųsti kitas užklausas. Serveris tuo metu užregistravęs užklausą, bando gauti klientui reikiamą resursą arba vykdyti užklausoje nurodytą veiksmą. Klientas tik po kurio laiko vėl siunčia užklausą serveriui, tik šįkart jau neprašydamas resurso ir neliepdamas įvykdyti veiksmų, bet klausdamas ar galima pasiimti resursą, ar jau įvykdytas veiksmas ar jų seka. Serveris užklaustas kliento, jeigu yra įvykdęs savo užduotį, grąžina klientui atsakymą su užklaustu resursu. Kitu atveju klientui iš serverio gauna atsakymą, kad pradėtas procesas dar yra neužbaigtas. Šį veikimo principą vaizduoja pateikta schema (5 pav.)



5 pav. Asinchroninio komunikavimo schema.

Toks yra tradicinis asinchroninio komunikavimo principas. Kuriant mikroservisų architektūrą paremtą informacinę sistemą, neretai susiduriama su tokiu atveju, kai norint sukurti resursą arba įgyvendinti procesą neužtenka tik dviejų tarnybų įsikišimo. Tokiais atvejais, pasinaudojant asinchroninėmis integracijų technologijomis, galima perduoti žinutę ne vienai tarnybai, o kelioms iš karto. Tada kelios tarnybos gali klausytis šių žinučių ir į jas reaguoti vienu metu vykdant savo atsakomybes. Visoms tarnyboms baigus darbą, jie taip pat sukuria žinutes apie savo darbo pabaigą tik, kai visos žinutės yra išsiųstos, būna galutinai sukuriamas resursas arba procesas tampa įvykdytu. Įgyvendinti tokius procesus nebūna paprasta ir dažnu atveju reikalingos papildomos technologijos įgyvendinimui. Taip pat turi būti tarnyba, atsakinga už tikrinimą, ar visi savo procesus atliekančios tarnybos baigė darbą. Ši tarnyba turi saugoti proceso būseną (angl. „*state*“) realiu laiku. Ši būsena, procesus vykdančioms tarnyboms, turėtų būti nežinoma. Toks pavyzdys yra vadinamas „publikavimo/prenumeravimo“ (angl. „*publish/subscribe*“) mechanizmu. Šiame mechanizme yra tarnybos, kurios prenumeruojasi prie žinučių apie naujo resurso sukūrimo inicijavimą ir pačios paskelbia pranešimus apie savo procesų pabaigą. Tarnyba, kuri saugo proceso būseną, prenumeruojasi prie šių pranešimų ir atnauja būseną. Toks modelis yra aprašomas Espen Johnses magistriniame darbe „Reliable Asynchronous Communication in Distributed Systems“ [Joh18]. Taip pat šis mechanizmas yra labai plačiai naudojamas įvykiais paremtose (angl. „*event-based*“) architektūrose.

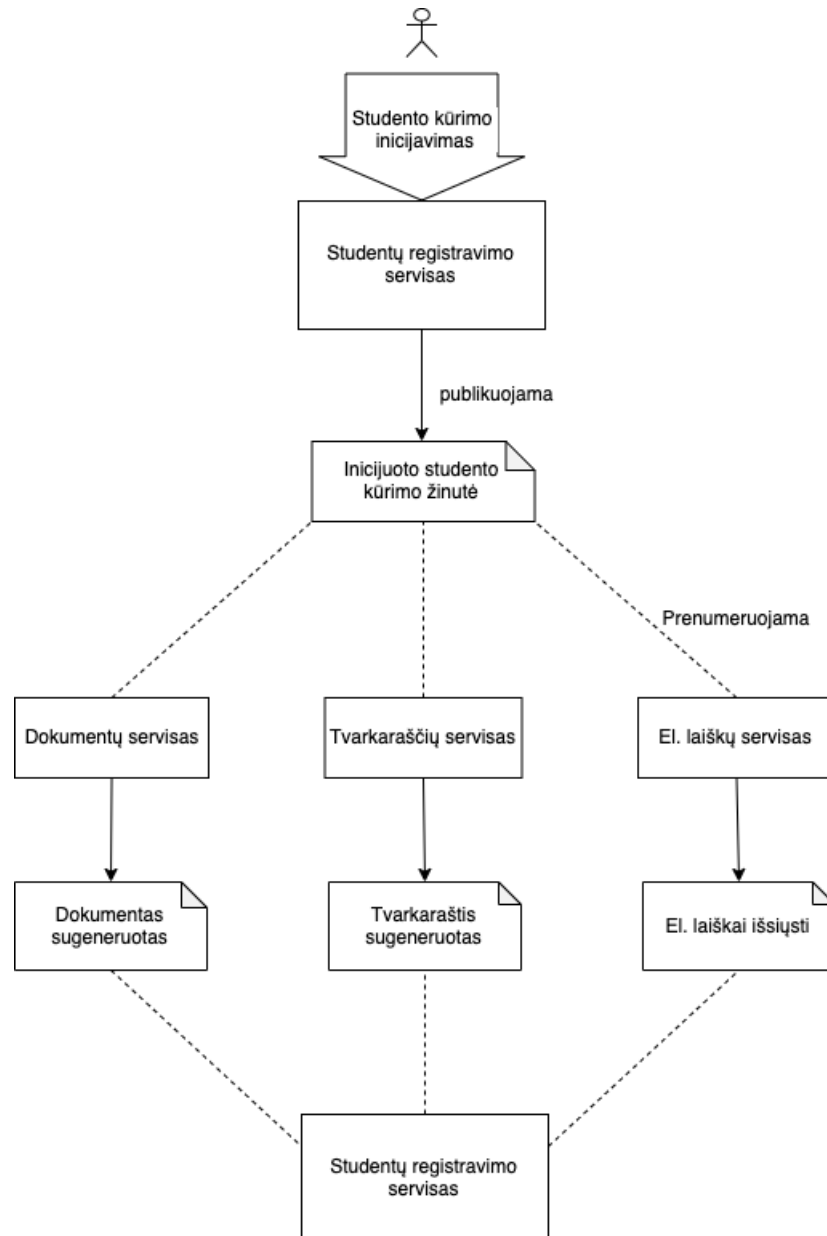
Verta paminėti, kad toks komunikavimo tipas yra neblokuojantis (angl. „*non-blocking*“). Tai reiškia, kad skirtingai nei sinchroninių komunikacijų metu, tarnyba išsiuntusi užklausa, gali toliau vykdyti kitus procesus. Vienas iš pavyzdžių būtų bekuriant naują resursą leisti vartotojui modifikuoti kitus nesusijusius resursus.

Siekiant geriau išaiškinti, kaip veikia asinchroninis komunikavimas mikroservisų sistemose, išanalizuosime modelinę situaciją, kur komunikavimas vyktų būtent minėtu būdu. Patį sistemos modelį pasirinksiame tokį patį, koks buvo naudojamas apibūdinti sinchroninį komunikavimą. Taigi, asinchroniniu komunikavimo modeliu tarnyba atsakinga už studentų duomenų tvarkymą bus informuota apie naujo studento kūrimo inicijavimą. Ši tarnyba savo duomenų bazėje sukuria naują įrašą ir vykdo tokius veiksmus eilės tvarka:

1. Studentų duomenų tarnyboje sukuriamą žinutę apie studento užregistravimo inicijavimą.
2. Dokumentų tarnyba prenumeruojasi tokią žinutę ir ją gavęs, sugeneruoja dokumentą ir sukuria žinutę apie sėkmingą sugeneravimą. Nesekmės atveju sukuria žinutę, kad nepavyko sukurti tokio dokumento.
3. Tvarkaraščių tarnyba taip pat prenumeruojasi prie inicijavimo žinutės ir tokią gavęs, sugeneruoja tvarkaraštį studentui. Sėkmingai įvykdęs darbą, kaip ir dokumentų tarnyba, sukuria žinutę apie sėkmingą sugeneravimą, o nesekmės atveju žinutę, kad nepavyko užbaigti proceso arba jis užbaigtas nekorektiškai.
4. Elektroninių laiškų tarnyba irgi prenumeruojasi inicijavimo žinutę ir ją gavęs vykdo savo atsakomybes – išsiųsti laiškus dėstytojams. Procesui pasibaigus, sėkmingai ar ne, išsiunčiama žinutė apie proceso pabaigos rezultatus.
5. Studentų tarnyba prenumeruojasi prie visų žinučių, kurias siunčia su studento kūrimu su-

sijusios tarnybos. Gaudama šiuos pranešimus, tarnyba pas save išsisaugo esamą kuriamo studento situaciją. Jeigu buvo gautas pranešimas apie nesėkmingą kažkurio iš tarpinių procesų darbą, tada tarnyba pasižymi, kad tokio studento sukurti nepavyko. Kitu atveju, jeigu viskas įvyko sėkmingai, tai tarnyba patvirtina sėkmingą studento užregistravimą ir baigia procesą.

Asinchorninio komunikavimo metu, visi procesai žinutes gauna nepertraukdami vienas kito. Tokią veiksmų seką vaizdžiai parodo ši schema (6 pav.):



6 pav. Asinchroninio komunikavimo mikroservisuose schema.

Toks modelis yra tik vienas iš būdų koku principu gali būti realizuojamas asinchroninis komunikavimas.

4.2. Asinchroninių integracijų technologijos

Yra daug skirtingų būdų ir technologinių, siekiant įgyvendinti asinchronines integracijas tarp mikroservisų. Kai kurios kalbos jau yra pritaikytos asinchroninėms integracijoms. Pavyzdžiui objektinė programavimo kalba JAVA nuo 8 versijos savo programų kūrimo sąsajoje jau turi „Completable Future“ modelį, kuriame pagalbinių metodų ir klasių pagalba galima įgyvendinti asinchroninius komunikavimus. Remiantis Vineet John ir Xia Liu publikacija „Apklausa apie paskirstytų pranešimų brokerius“ (angl. „*A Survey of Distributed Message Broker Queues*“) [JL17] pranešimų brokeriai (angl. „*Message Brokers*“) yra populiariausias komunikavimo pasirinkimas paskirstytose sistemose. Bendravimas per pranešimų brokerius iš savo principo yra asinchroninis. Dėl pranešimų brokerių populiarimo, kaip technologinį tipą šiame darbe aptarsime būtent šią technologiją.

Dar 2019 metų gale publikuotame leidinyje apie modernius pranešimų brokerius [Sha19] lyginami 3 labai populiarūs pranešimų brokeriai:

1. Apache Kafka.
2. Rabbit MQ
3. NATS.

Šios trys technologijos yra gan panašios savo principais, bet renkantis, kurią iš jų naudoti kuriant sistemą, reikia gerai atsižvelgti į jų privalumus ir trūkumus. Kaip ir minima PHilippe Dobbelaere ir Kyums Sheykh Esmaili publikuotame straipsnyje „Kafka prie RabbitMQ“ (angl. „*Kafka versus RabbitMQ*“) [DE17], nėra geresnio ar blogesnio iš šių dviejų pranešimų brokerių. Abu brokeriai buvo sukurti siekiant išspręsti skirtingas problemas ir įgyvendinti skirtingus tikslus. Neįšimtis yra ir su NATS. Nors visi brokeriai veikia panašiu mechanizmu, skirtingiems prioritetams reikia skirtingų brokerių.

Šie pranešimų brokeriai, veikia pagal „publikavimo/prenumeravimo“ mechanizmą, kuriame „publikuotojai“ siunčia pranešimus į brokerį, o „prenumeratoriai“ skaito pranešimus. Į brokerį atsiunčiami pranešimai būna saugomi eilėje, iš kurios „prenumeratoriai“ skaito prenumeruotas žinutes.

5. Skirtingų integracijų tipų privalumai ir trūkumai

5.1. Silpnas sujungimas ir stiprus sanglauda (angl. „*Loose Coupling and High Cohesion*“)

Sinchronines ir asinchronines integracijas mikroservisuose galima palyginti pagal tai ar tai atitinka pagrindinius mikroservisų kriterijus. Pagrindiniai principai, pagal kuriuos galima teigti, kad tarnyba yra tinkamai suprojektuota yra [New15]:

- Silpnas sujungimas (angl. „*Loose Coupling*“). Tai reiškia, kad kiekviena tarnyba turėtų kuo labiau būti nepriklausoma. Jeigu yra daromas pakeitimas tarnybai, tai turėtų tik jai vienai ir būti atliekama, nes vienas iš pagrindinių mikroservisų privalumų yra galimybė įdiegti vieną tarnybą, nediegiant kitų.
- Stiprus sąryšis (angl. „*High Cohesion*“). Tai reiškia, kad kiekviena tarnyba turėtų savo logiką laikyti pas save ir atlikti visus su savo atsakomybėmis susijusius procesus.

Šiuose dviejuose aspektuose ryškus asinchroninių integracijų pranašumas. Asinchroninių komunikacijų metu, tarnybos tiesiog siunčia pranešimus apie procesų pabaigą ir klausosi, kada juos pradėti. Sinchroninės komunikacijos reikalauja žinoti implementacijos detales ir kreiptis į kitas tarnybas, per klientus arba tiesiogiai. Dažniausiai asinchroninės komunikacijos visoje sistemoje veikia vienodai.

5.2. Efektyvumas

Vienas pagrindinių skirtumų tarp sinchroninių ir asinchroninių integracijų yra blokavimas. Kaip jau minėta anksčiau, sinchroninės integracijos yra blokuojančios, priešingai nei asinchroninės. Kaip minima įmonės „Microsoft“ publikuotame darbe apie mikroservisus ir jų kūrimą su .NET technologija [SWP⁺20] vykdant ilgus ir daug resursų reikalaujančius procesus pasitelkiant tokią technologiją kaip RESTful, dėl ilgo atsakymo laukimo galima patirti operacijos laiko pasibaigimus (angl. „*timeouts*“) ir dėl to sėkmingai neužbaigti procesų.

Kitas asinchroninių integracijų privalumas yra lygiagretiškumas. Priešingai nei sinchroniniai komunikavimai, pranešimus gali prenumeruoti ir vienu metu savo pareigybes vykdyti keli procesai, kas, žinoma, padidina sistemos efektyvumą ir sutrumpina procesų veikimo laiką.

5.3. Implementacijos kompleksiskumas

Dar vienas labai svarbus faktorius renkantis tarp technologijų yra kompleksiskumas. Kuriant programinę įrangą kompleksiški reikalavimai reiškia, kad atsiras daug vietos klaidoms. Klaidos labai stipriai didina gamybos kaštus ir sistemos vartotojų nepasitenkinimą. Asinchroninės integracijos yra žymiai sudėtingesnės ir dažniausiai naudojami pranešimų brokeriai reikalauja papildomų implementacijos detalių, kad pavyktų sujungti tarnybą su brokeriu ir pavyktų klausytis arba publikuoti pranešimus. Tokie darbai iš PĮ kūrėjų reikalauja aukštos kvalifikacijos, o tokių darbuotojų, kurie atitiktų reikalavimus, įmonėms yra daug sunkiau surasti ir įdarbinti. Kita vertus sinchroniniai tarnybų kvietimai yra paprasti ir tarnybų programų programavimo sąsajos būna išlaikomos

primityvios ir trivialios. Tas minima ir David S.Linthicum knygoje „Enterprise Application Integration“ [SLi00].

5.4. Veiksmų istorija

Kuriant dideles informacines sistemas, dažnai reikia atsižvelgti į įvykių registravimą. Dažnas reikalavimas didelėms sistemoms yra veiksmų istorija. Tokiems uždaviniams spręsti geriausiai tinka asinchroninės užklauskos, nes pasitelkus tokias technologijas kaip pranešimų brokeriai, žinutės ir įvykiai būna saugomi, ypač įvykiais paremtose (angl. „*event-based*“) architektūrose. Sinchroninių integracijų metu įvykių istorijos problemą reikia spręsti kitaip ir ji netampa tokia triviali.

5.5. Įvykių sekos užtikrinimas

Procesų vykdymai, kaip jau minėta studentų registracija, neretai reikalauja užtikrinimo, kad vienas procesas bus įvykdomas anksčiau už kitą. Sinchroninės integracijos yra blokuojamos ir vykdomos paeiliui, todėl šios problemos nelieka, tačiau asinchroninis komunikavimas gali būti lygiagretinamas ir siekiant užtikrinti korektišką eiliškumą reikia įgyvendinti papildomą logiką. Dėl tokių atvejų, lygiagretūs procesai tampa dar labiau komplikuoti. Tokiais atvejais asinchroninio komunikavimo metu tarnybos turi saugoti proceso būseną ir užtikrinti veiksmų eiliškumą.

5.6. Tarnybų kompozicija

Vienas iš mikroservisų architektūros privalumų yra išskaidytų tarnybų perpanaudojimas. Įmonės, kurios užsiima projektiniais darbais, neretai siekia kurti tokią PĮ, kuria būtų galima pasiremti arba jos komponentes panaudoti ateities projektuose. Mikroservisai, kurie naudoja sinchronines integracijas būna mažiau surišti su domeno logika ir norint integruoti tarnybą kitoje informacinėje sistemoje neprivaloma taip prisirišti prie technologijų. Tokios tarnybos įmonėms atneša daug naudos ir sutaupo kaštų ateityje.

Tyrimas

Toliau pateikta lentelė su lyginimo rezultatais tarp sinchroninių komunikacijų ir asinchroninių remiantis šiame skyrelyje apibrėžtais kriterijais ir analize (1 lentelė). Prie komunikavimo tipo surašyti kriterijai, kuriuose nurodytas komunikavimo tipas yra pranašesnis:

1 lentelė. Komunikavimo tipų palyginimo rezultatai.

Asinchroninis komunikavimas	Sinchroninis komunikavimas
Silpnas sujungimas ir stiprus sanglauda	Implementacijos kompleksiskumas
Efektyvumas	Įvykių sekos užtikrinimas
Veiksmų istorija	Tarnybų kompozicija

6. Rezultatai ir išvados

Šiame darbe buvo išanalizuoti pagrindiniai komunikavimo integracijų mikroservisų architektūrose tipai, pateiktos mikroservisų atsiradimo priežastys ir aptarti mikroservisų architektūrų pranašumai prieš monolitines sistemas. Taip pat įvardinta tinkamo integracijų pasirinkimo svarba. Buvo išskirti pagrindiniai mikroservisų komunikavimo tipai:

1. Tarnybų jungimas per duomenų bazę.
2. Sinchroninės užklausos/atsakymo (angl. „*request/response*“) integracijos.
3. Asinchroninės, įvykiais paremtos (angl. „*event-based*“) integracijos.

Darbe buvo detaliau aptarti sinchroniniai ir asinchroniniai komunikavimai, tačiau taip pat paminėtas ir apibūdintas komunikavimas per duomenų bazę tipas. Apžvelgti dviejų pagrindinių tipų principai. Išskirti skirtingų tipų komunikavimo privalumai ir trūkumai. Apibūdintos integracijų tipų panaudojimas modelinėse situacijose. Pabaigoje pateikti pagrindiniai pasirinkimo tarp skirtingų technologijų aspektai ir priežastys.

Pasinaudojus pateikta informacija, buvo įgyvendinti pagrindiniai darbo uždaviniai:

- Apibūdinta, kas yra mikroservisai, kuo jie ypatingi, kaip jie projektuojami, remiantis šaltiniais.
- Išskirti pagrindiniai integracijų ir komunikavimo tipai ir jų savybės.
- Remiantis šaltiniais išdėstyti trūkumai ir privalumai apibrėžtuose kriterijuose.
- Palyginti skirtingi komunikavimo būdai mikroservisų architektūrose.
- Pateiktos rekomendacijos ir scenarijai, kokiomis aplinkybėmis kokį tipą geriausia naudoti.

Darbo uždaviniams įgyvendinti buvo naudojamos ne tik teorinės žinios iš įvairių šaltinių, tačiau ir istoriniai faktai, padėję paaiškinti tam tikrus mikroservisų ir jų komunikavimo aspektus.

Pateikus darbo rezultatus galima suformuluoti tokias išvadas:

- Mikroservisų architektūrose pasirinkimas tarp skirtingų komunikavimo tipų yra labai svarbus, nes tai lemia sistemos veikimo stilių ir daug kitų aspektų.
- Sinchroninės integracijos yra paprastesnės ir lengviau įgyvendinamos. Jos pranašesnės siekiant greitai gauti rezultatus, be kurių, nebūtų galima vykdyti sekančių veiksmų informacinėje sistemoje.
- Asinchroninės integracijos žymiai geriau atitinka pagrindinius mikroservisų aspektus, tokius kaip: silpnas sujungimas ir stiprus sąryšis.
- Asinchroninės integracijos yra efektyvesnės už sinchronines integracijas, dėl savo lygiagrečio savybių.
- Asinchroninės komunikacijos gali įgyvendinti įvykiais paremtas architektūras, kuriose nereikėtų papildomų pastangų, siekiant gauti informacinės sistemos veiksmų istoriją.
- Sinchroninis komunikavimas yra sklandesnis ir paprastesnis, dėl to paliekama mažiau vietos klaidoms.
- Sinchroninis tarnybų komunikavimas suteikia lankstesnes galimybes tarnybų kompozicijai, perpanaudojimui.
- Iš įmonių pusės daug naudingiau yra kurti sinchronines sistemas dėl kaštų skiriamų darbuotojų kompetencijoms.

Taigi darytina išvada, jog projektuojant mikroservisų architektūromis paremtą PĮ yra labai svarbu atsižvelgti į sistemos poreikius ir gerai apsvarstyti, koku komunikavimo stiliumi bus pasitikiama. Atsižvelgiant į suformuluotas išvadas, galima teigti, kad nors asinchroninės komunikacijos ne visada yra lengviausias būdas, tačiau tokią komunikaciją apibūdina geriausias mikroservisų praktikas ir paradigmas.

- [AM18] Omar Al-Debagy and Peter Martinek. A comparative review of microservices and monolithic architectures. <https://arxiv.org/pdf/1905.07997.pdf>, 2018. tikrinta 2020-04-22.
- [DE17] PHILIPPE DOBBELAERE and KYUMARS SHEYKH ESMAILI. Kafka versus rabbitmq. <https://arxiv.org/pdf/1709.00333.pdf>, 2017. tikrinta 2020-05-10.
- [DGL⁺17] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: yesterday, today, and tomorrow. <https://arxiv.org/pdf/1606.04036.pdf>, 2017. tikrinta 2020-04-22.
- [Fea04] Micheal C. Feathers. *Working Effectively with Legacy Code*. Prentice Hall PTR, Upper Saddle River, United States of America, 2004. tikrinta 2020-04-22.
- [FL14] Martin Fowler and James Lewis. Microservices. <https://martinfowler.com/articles/microservices.html>, 2014. tikrinta 2020-04-15.
- [Fow17] Susan J. Fowler. *Production-Ready Microservices*. O'Reilly Media, Sebastopol, United States of America, 2017. tikrinta 2020-05-01.
- [HKL09] Volker Hockmann, Heinz D. Knoell ir Ernst L. Leiss. *Encyclopedia of Multimedia Technology and Networking, Second Edition*. 2009. tikrinta 2020-04-22.
- [JL17] Vineet John and Xia Liu. A survey of distributed message broker queues. <https://arxiv.org/pdf/1704.00411.pdf>, 2017. tikrinta 2020-04-22.
- [Joh18] Espen Johnsen. *Reliable Asynchronous Communication in Distributed Systems*. Magistrinis darbas, Bergensis Universitas, 2018. tikrinta 2020-05-10.
- [Loi18] J. Loisel. Soap vs rest (why comparing them is a nonsense. <https://octoperf.com/blog/2018/03/26/soap-vs-rest/>, 2018. tikrinta 2020-04-22.
- [Mak17] Joni Makkonen. *Performance and usage comparison between REST and SOAP web services*. Magistrinis darbas, Aalto University, 2017. tikrinta 2020-05-05.
- [New15] Sam Newman. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, Sebastopol, United States of America, 2015. tikrinta 2020-04-20.
- [Ray03] Eric Steven Raymond. *The Art of UNIX Programming*. Addison-Wesley, Upper Saddle River, United States of America, 2003. tikrinta 2020-04-22.
- [Sha19] Sowmya Nag K Sharvari T. A study on modern messaging systems – kafka, rabbitmq and nats streaming. <https://arxiv.org/pdf/1912.03715.pdf>, 2019. tikrinta 2020-05-10.
- [SLi00] David S. Linthicum. *Enterprise Application Integration*. Addison-Wesley, Upper Saddle River, United States of America, 2000. tikrinta 2020-04-22.

- [SWP⁺20] Sabah Shariq, Maira Wenzel, John Parente, Nish Anil, and Miguel Veloso. Communication in a microservice architecture. <https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture>, 2020. tikrinta 2020-04-22.