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. *„Technology Heterogeneity“*) 4
1.2.2. Atsparumas (angl. *„Resilience“*) 5
1.2.3. Plečiamumas (angl. *„Scaling“*) 5
1.2.4. Lengvas diegimas (angl. *„Ease of Deployment“*) 5
1.2.5. Organizacinis pasiskirstymas (angl. *„Organizational Allignment“*) 5
1.2.6. Kompozicija (angl. *„Composability“*) 5
1.2.7. Optimizuotas pakeičiamumas (angl. *„Optimizing for Replaceability“*) 5
1.3. Monolitinių sistemų skaidymas į mikroservisus 6

2. MIKROSERVISŲ SISTEMOS VIDINIŲ INTEGRACIJŲ TIPAI 7
2.1. Servisų komunikavimas per duomenų bazę 8

3. SINCHRONINĖS (ANGL. “SYNCHRONOUS”) INTEGRACIJOS 9
3.1. Sinchroninių integracijų principas 9
3.2. Sinchroninių integracijų technologijos 10

4. ASINCHRONINĖS (ANGL. “ASYNCHRONOUS”) INTEGRACIJOS 12
4.1. Asinchroninių integracijų principas 12
4.2. Asinchroninių integracijų technologijos 14

5. SKIRTINGŲ INTEGRACIJŲ TIPŲ PRIVALUMAI IR TRŪKUMAI 16
5.1. Silpnas sujungimas ir stiprus sąryšis (angl. *„Loose Coupling and High Cohesion“*) 16
5.2. Efektyvumas 16
5.3. Implementacijos kompleksiškumas 16
5.4. Veiksmų istorija 17
5.5. Įvykių sekos užtikrinimas 17
5.6. Servisų perpanaudojimas 17

6. REZULTATAI IR IŠVADOS 18

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 kokias technologijas pasirinkti, pradedant kurti naują informacinę sistemą. Vis daugėja skirtingų technologijų ir programavimo kalbų, kurios yra pranašesnes už kitas tik siaurose srityse, dėl to dažnu atveju neužtenka pasirinkti vieną technologiją ar programavimo kalbą norint sukurti kokybišką ir tvarią PĮ. Kai kurios technologijos yra pranašesnės resursų taupyme, kitos pranašesnės daug skirtingų bibliotekų palaikymu ir lengvai naudojama aplikacijų programavimo sąsają (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 technogolijas, atsiranda problema kaip jas apjungti, kad veiktų vieningai. Tokiu atveju, galima naudotis saitynų tarnybų pagalba (angl. ”Web services”), tačiau to neužtenka, nes kas, jeigu norime, skirtingus funkcionalumus įgyvendinti skirtingų technologijų pagalba. Tada dažnu atveju naudojamas mikroservisų (angl. *„Microservices“*) architektūrinis modelis. Remiantis šiuo modeliu būtų kuriami atskiri moduliai (taip pavadinsime atskirus sitemos vienetus), kurių kiekvienas būtų atsakingas už savo funkcionalumą. Problema su šia architektūra, kad reikia priversti šiuos skritingus modulius bendrauti tarpusavyje, o tai nebūna taip paprasta.

Problema

Kokį integracijų tipa rinktis mikroservisų architektūrose, norint įgyvendinti komunikaciją tarp skirtingų servisų.

Darbo tikslas

Palyginti skirtingus komunikacijų tipus, apžvelgti jū pranašumus ir trūkumus, pateikti pavyzdžių, kur vieni tipai yra pranašesni už kitus, ir pateikti technologijų šiem tipam realizuoti pvyzdžių.

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 Mazzara publikuotą straipsnį „Microservices: yesterday, today, and tomorrow“ [DGL+17] jau 1960-iais buvo susiduriama su problemomis susijusiomis su dižiulio masto PĮ kūrimu ir kaip tai projektuoti. Buvo kuriama daug būdų kaip tai daryti, ir daug toerijų kaip tūrėtų atrodyti PĮ kodas, ir kaip projektuoti informacines sitemas. Daug veliau apie 2000 metus, susiformavo sąvokos „Service-Oriented Computing“ (toliau SOC) ir „Service-Oriented Architecture” (toliau SOA), kurių idėja buvo ir paradigmos buvo apie tai, kad aplikacija, arba kitaip pavadinus servisas, turėtų būti atsakingas už konkretaus resursų ar verslo logikos informaciją. Šį servisą tūri 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 dirbutvėse (angl. *„workshop“*). Po metų ta pati grupė architektų nusprendė, labiausiai tinkantis pavadinimas šiam architektūriniam tipui yra mikroservisai (angl. *„microservices“*). Po šiuo terminu slypi daug idėjų ir paradigmų, tačiau pagrindinė mintis yra skaidymas dideles sistemos į mažas „granules“ ir mažus servisus. Taip pat norint apibūdinti senas sistemas, kurios buvo nedalomos ir sistemą yra paleidžiamas kaip vienas vienetas, atsirado terminas „monolitas“ (angl. *„monolith“*). Nors 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 šios apibūdinti informacinių sistemų architektūriniams tipams.

1.2. Mikroservisų pranašumai prieš monolitus

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

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

Kiekvienas servisas 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 servisa galim projektuoti skirtingom technologijom, 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 vieno serviso veikimas. Tokiu atveju kiti servisai apie tai nežinotų ir veiktų toliau, o norint ištaisyti problemą, užtektų sutaisyti ir perkrauti vieną servisą.

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

Stambioje monolitinėje sistemoje visos plečiamumo ir efektyvumo problemos spendžiamos kartu. Mikroservisų sistemoje kievieną tokio tipo problemą sprendžiame atskiruose servisuose. Tokiu atveju servisus, kurie reikalauja mažiau resursų galima suteikti mažiau resursų, o sunkesniems ir mažiau efektyviems servisams išskirti daugiau. Tačiau verta paminėti, kad mikroservisų sistemos plečiamumas ne visada yra geras, tai aprasšyta 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 tuos servisus, kurie yra susiję su pakeitimais.

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

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

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

Įmonės dažnai susiduria su problema, kai tas pats funckionalumas reikalingas keliose informacinėse sistemose. Mikroservisų architektūroje, kadangi sistema susideda iš mažų autonomiškų servisų, juos 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 papraščiau, kai užtenka perrašyti konkretų servisą, neliečiant likusios informacinės sitemos.

1.3. Monolitinių sistemų skaidymas į mikroservisus

Paskutinį dešimtmetį tapo gan populiaru stambaus mąsto monolitus skaidyti į mikroservisus, tačiau tai nėra taip paprasta. Visų pirma skaidant monolitą į atskirus servisus labai svarbu identifikuoti, kokie mažesni servisai bus. Serviso riboms apibrėžti panaudosime Micheal C. Feahters knygoje „Working Effectively with Legacy Code“ [Fea04] apibrėžtą terminą „siūle“ (angl. *„seam“*).Siūle šiuo atveju yra kodo dalis, kuri yra izoliuota ir autonomiška. Siūles 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 aplikaciją paleisti su tiek kopijų kiek turėsime siūlių.
2. Išskirstyti kvietimus į kopijas, pagal tai kokias siūles kopijos reprezentuoja.
3. Išvalyti aplikacijos kopijas, paliekant tik siūlių funkcionalumą.

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

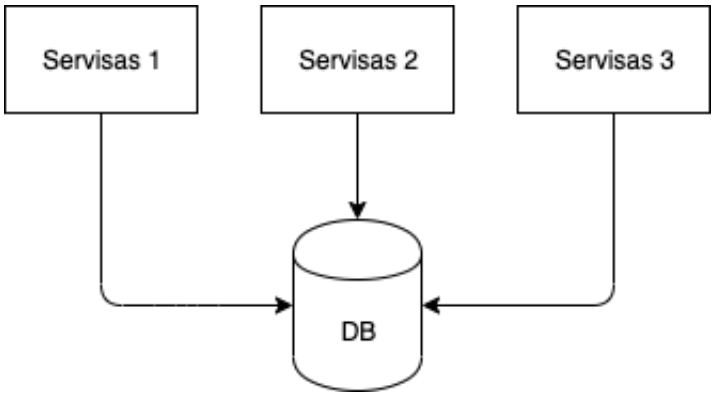
2. Mikroservisų sistemos vidinių integracijų tipai

Pagal Sam Newman knygą „Building Mircroservices: 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, kiti servisai 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 išskyria keletą skirtingų mikroservisų komunikavimo tipų, kuriuos ir aptarsime:

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

2.1. Servisų 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.):



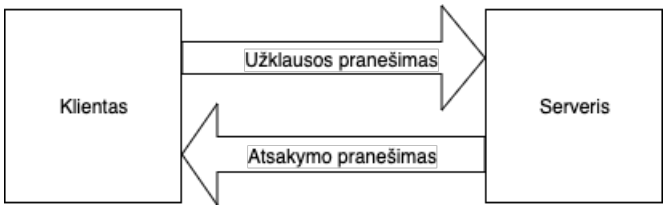
1 pav. Mikroservisų komunikavimas duomenų bazės pagalba.

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 užklausos/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 būti arba svetainės turinys, arba grąžinama klaida, pavyzdžiui kad toks resuras neegzistuoja arba klientas yra neautorizuotas šios svetainės lankytojas. Šis bendravimas yra perteiktas pateikta schema (2 pav.):



2 pav. Sinchroninio komunikavimo schema.

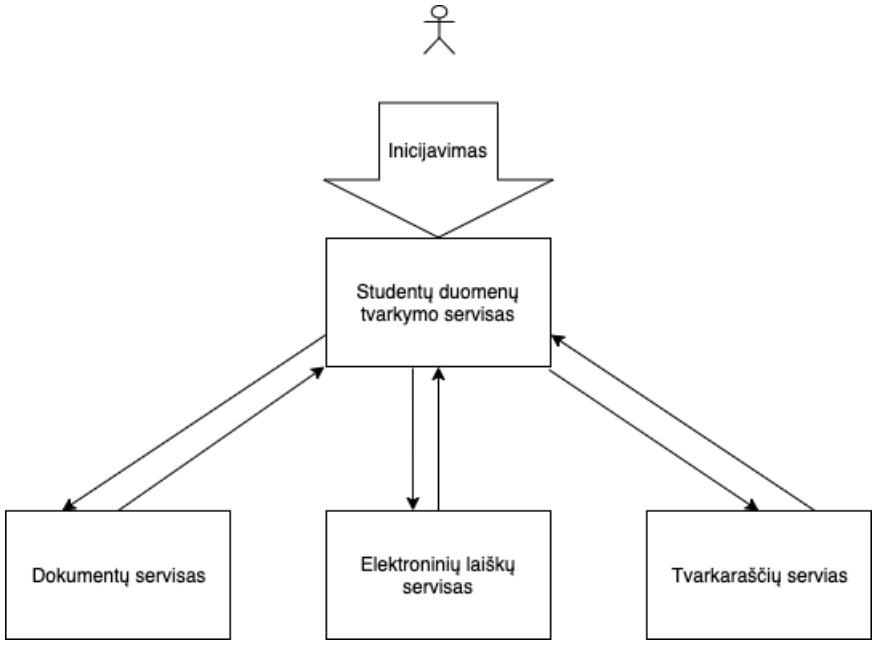
Taigi šis modelis puikiai gali veikti ir mikroservisų sistemoje. Vienas servisas siunčia užklausą į kitą servisą norėdamas gauti informaciją arba inicijuoti, kokį nors veiksmą. Šis modelis ypatingas tuo, kad yra primityvus ir iškart užklausą išsiuntęs servisas žinos ar sėkmingai pavyko atlikti norimą veiksmą. Šis modelis yra labai geras, kai tik įvykus sėkmingai užklausai 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 architektiniu modeliu. Šioje sistemoje yra atskiri servisai atsakingi už studentų duomenų tvarkymą, dokumentų generavimą, elektroninių laiškų siuntimą, tvarkaraščių sudarymą ir saugojimą. Be šių servisų sistema turėtų turėti ir daug kitos paskirties mikroservisų, bet dabar aptarsime tik šiuos. Taigi, sinchroniniu komunikavimo modeliu servisas atsakingas už studentų duomenų tvarkymą bus informuotas apie naujo studento kūrimo inicijavimą. Šis servisas savo duomenų bazėje sukuria naują įrašą ir vykdo tokius veiksmus eilės tvarka:

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

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

Tokią veiksmų seką vaizdžiai parodo tokia 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 ju nelabai galima lyginti [Loi18]. Tačiau remiantis Joni Makkonen magistrinio darbo „REST ir SOAP saityno tarnybų efektyvumo ir naudojamumo palyginimas“ [Mak17] galima teigti, kad REST yra pranašesne ir šiame darbe išsiplėsime tik su šiuo architektūriniu stilium.

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“*.

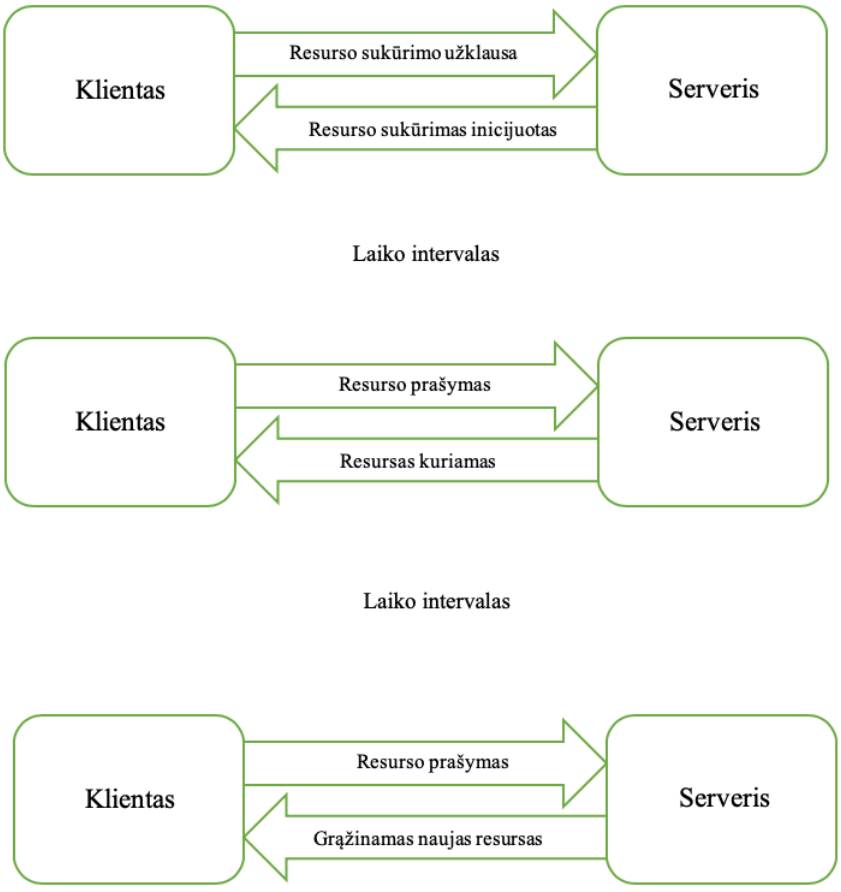
REST atsakymo struktūra skiriasi nuo užklausos tik tuo, kad vietoje HTTP metodo, gaunamas HTTP statuso kodas. Tokio komunikavimo per REST saityno tarnybas pavyzdį galime pamatyti šioje schemoje (4 pav.):

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 užklausa 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 užregistraves 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 asynchroninio komunikavimo principas. Kuriant mikroservisų architektūra paremtą informacinę sistemą, neretai susiduriama su tokiu atveju, kai norint sukurti resursą arba įgyvendinti procesą neužtenka tik dviejų servisų įsikišimo. Tokiais atvejais pasinaudojant asinchroninėmis integracijų technologijomis, galima perduoti žinutę ne vienam servisui, o keliems iš karto. Tada keli servisai gali klausytis šių žinučių ir į jas reaguoti vienu metu vykdant savo atsakomybes. Visiem servisams 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. Įgyvendti tokius procesus nebūna paprasta ir dažnu atveju reikalingos papildomos technologijos įgyvendinimui. Taip pat turi būti servisas atsakingas už tikrinimą ar visi savo procesus atliekantys servisai baigė darbą. Šis servisas turi saugoti proceso būsena (angl. *„state“*) realiu laiku. Ši būsena procesus vykdantiems servisams turėtų būti nežinoma. Toks pavyzdys yra vadinamas „publikavimo/prenumeravimo“ (angl. *„publish/subscribe“*) mechanizmu. Šiame mechanizme yra servisai, kurie prenumeruojasi prie žinučių apie naujo resurso sukūrimo inicijavimą ir patys paskelbia pranešimus apie savo procesų pabaigą. Servisas, kuris saugo proceso būseną, prenumeruojasi prie šių pranešimų ir atnaujiną 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.

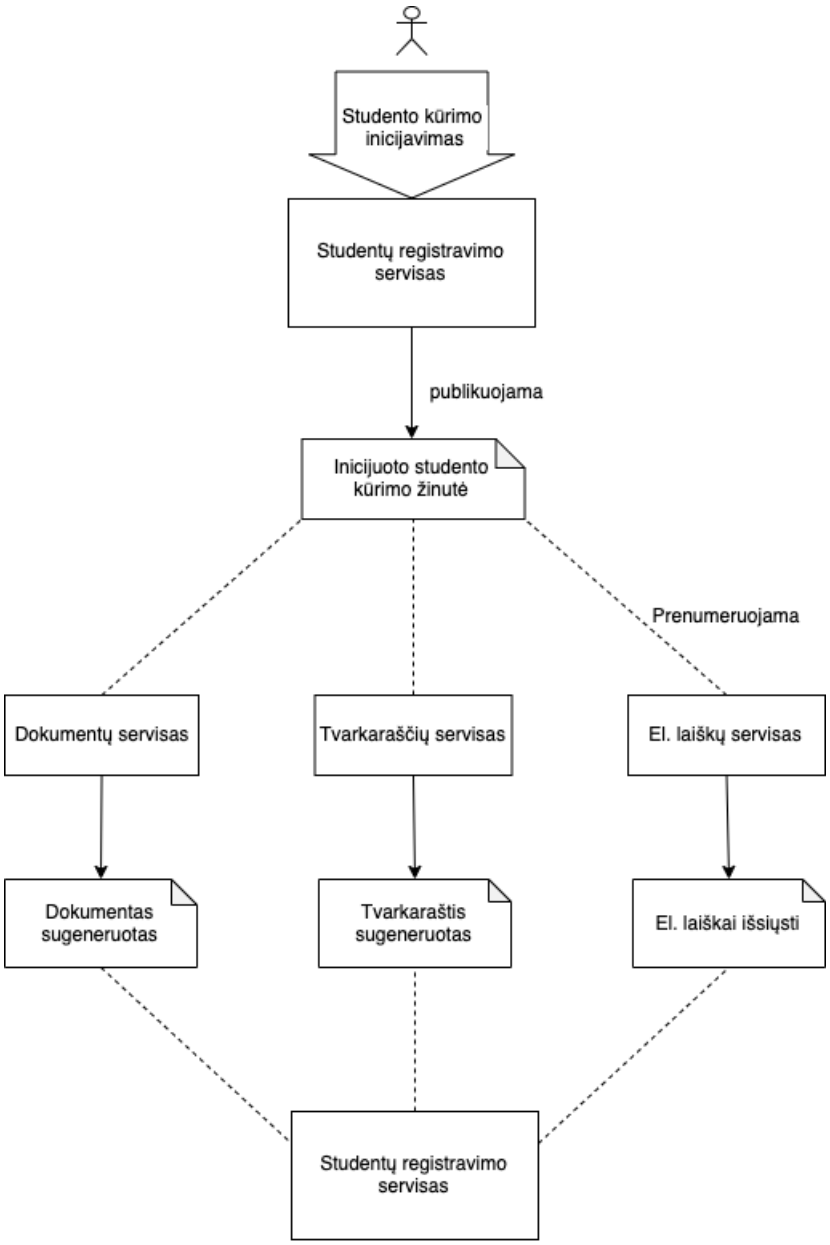
Verta paminėti, kad toks komunikavimo tipas yra neblokuojantis (angl. *„non-blocking“*). Tai reiškia, kad skirtingai nei sinchroninių komunikacijų metu, servisas išsiuntęs užklausą, gali toliau vykdyti kitus procesus. Vienas iš pavyzdžių būtų bekuriant naują resursą leisti vartotojui modifikuoti kitus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

Siekant geriau išaiškinti, kaip veikia asinchroninis komunikavimas mikroservisų sistemose, išanalizuosime modelinė situacija, kur komunikavimas vyktų būtent minėtu būdu. Patį sistemos modelį pasirinksime tokį patį, koks buvo naudojamas apibūdinti sichroninį komunikavimą. Taigi, asinchroniniu komunikavimo modeliu servisas atsakingas už studentų duomenų tvarkymą bus informuotas apie naujo studento kūrimo inicijavimą. Šis servisas savo duomenų bazėje sukuria naują įrašą ir vykdo tokius veiksmus eilės tvarka:

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

servisai. Gaudamas šiuos pranešimus, servisas pas save išsisaugo esama kuriamo studento situaciją. Jeigu buvo gautas pranešimas apie nesėkmingą kažkurio iš tarpinių procesų darbą, tada servisas pasižymi, kad tokio studento sukurti nepavyko. Kitu atveju, jeigu viskas įvyko sėkmingai, tai servisas patvirtina sėkmingą studento užregistravimą ir baigią procesą.

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



6 pav. Asinchroninio komunikavimo mikroservisuose schema.

Toks modelis yra tik vienas iš būdų kokiu 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

JAVA nuo 8 versijos savo aplikacijų programavimo sąsajoje jau pristatė „Completable Future“ modelį, kuriame pagalbinių metodų ir klasių pagalba galima įgyvendinti asinchroninius komunikavimus. Remiantis Vineet John ir Xia Liu publikacija „Apklausa apie paskirstytas pranešimų brokerius“ (angl. *„A Survey of Distributed Message Broker Queues“*) [JL17] pranešimų brokeriai (angl. *„Message Brokers“*) yra populiaurias komunikavimo pasirinkimas paskirstytose sistemose. Bendravimas per pranešimų brokerius iš savo principo yra asinchroninis. Dėl pranešimų brokerių populiarimo, kaip technologini tipą šiame darbe aptarsime būtent šią technologiją.

Dar 2019 metų gale publikuotame leidinyje apie modernius pranešimų brokerius [Sha19] lyginamos 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“ (anlg. *„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. Neišimtis yra ir su NATS. Nor visi brokeriai veikia panašiu mechanizmu, skirtingiem 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 sąryšis (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 servisas yra tinkamai suprojektuotas yra [New15]:

- Silpnas sujungimas (angl. *„Loose Coupling“*).Tai reiškia, kad kiekvienas servisas turėtų kuo labiau būti nepriklausomas. Jeigu yra daromas pakeitimas servisui, tai turėtų tik jam vienam ir būt atliekamas, nes vienas is pagrindinių mikroservisų privalumų yra galėjimas įdiegtį vieną servisą, nediegiant kitų.
- Stiprus sąryšis (anlg. *„High Cohesion“*).Tai reiškia, kad kievkienas servisas tūretų savo logiką laikyti pas save ir atlikti visus su savo atsakomybėmis susijusius procesus.

Šiuose dviejuose aspektuose ryškus asinchroninių integracijų pranašumas. Asinchroninių komunikacijų metu, servisai tiesiog siunčia pranešimus apie procesų pabaigą ir klausosi, kada juos pradėti. Sinchroninės komunikacijos reikalauja žinoti implementacijos detales ir kreiptis į kitus servisus, 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 ankščiau 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 kompleksiškumas

Dar vienas labai svarbus faktorius renkantis tarp technologijų yra kompleksiškumas. 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 servisą 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 sinchroninės servisų kvietimai yra paprasti ir servisų applikacijų programavimo sąsajos būna išlaikomos

primityvios ir trivelios. Tas minima ir David S.Linthicum knygoje „Enterpise 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 užklausos, nes pasitelkus tokias technologijas kaip pranešimų brokeriai žinutės ir įvykiai būna saugomi, ypač įvykiais paremtose (angl. *„event-based“*) architektūrose. Sinchroninių integracijų metų įvykių istorijos problemą reikia spręsti kitaip ir ji netampa tokia triveli.

5.5. Įvykių sekos užtikrinimas

Procesų vykdymai, kaip jau minėta studentų registracija, neretai reikalauja užtikrinimo, kad vienas procesas bus įvykdomas ankščiau 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 papildomos logiką. Dėl tokių atvejų, lygiagretūs procesai tampa dar labiau komplikuoti. Tokiais atvejais asinchroninio komunikavimo metu servisai turi saugot proceso būseną ir užtikrinti veiksmų eiliškumą.

5.6. Servisų perpanaudojimas

Vienas iš mikroservisų architektūros privalumų yra išskaidytų servisų 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 servisą kitoje informacinėje sistemoje neprivaloma taip prisirišti prie technologijų. Tokie servisai įmonėms atneša daug naudos ir sutaupo kaštų ateityje.

6. Rezultatai ir išvados

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

1. Servisų 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 pateiktos pragrindiniai pasirinkimo tarp skirtingų technologijų aspektai ir priežastys. Pasinaudojus pateikta informacija, buvo įgyvendinti pagrindiniai darbo uždaviniai:

- Apibūdinti, kas yra mikroservisai, kuo jie ypatingi, kaip jie projektuojami, remiantis šaltiniais.
- Išskirti pagrindinius integracijų ir komunikavimo tipus ir jų savybes.
- Palyginti skirtingus komunikavimo būdus mikroservisų architektūrose.
- Pateikti rekomendacijas, kokias atvejais 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 sitemos veikimo stilių ir daug kitų aspektų.
- Sinchroninės integracijos yra paprastesnės ir legviau įgyvendinamos. Jos pranašesnės siekiant greitai gauti rezultatus, be kurių, nebūtų galima vykdyti sekančių veiksmų informacinėje sistemoje.
- Asinchroninės integracijos yra technologiškai pranašesnės ir žymiai geriau atitinką pagrindinius mikroservisų aspektus, tokius kaip: silpnas sujungimas ir stiprus sąryšis.
- Asinchroninės integracijos yra efektyvesnės už sinchronines integracijas, dėl savo lygiagretinimo savybių.
- Sinchroninis komunikavimas yra sklandesnis ir paprastesnis, dėl to paliekama mažiau vietos klaidoms.
- Iš įmonių pusės daug naudingiau yra kurti sinchronines sistemas dėl kaštų.

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

Šaltiniai

[AM18] Omar Al-Debagy and Peter Martinek. A comparative review of microservices and monolithic architectures. https://arxiv.org/pdf/1905.07997.pdf, 2018.

[DE17] PHILIPPE DOBBELAERE and KYUMARS SHEYKH ESMAILI. Kafka versus rabbitmq. https://arxiv.org/pdf/1709.00333.pdf, 2017.

[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.

[Fea04] Micheal C. Feathers. *Working Effectively with Legacy Code*. Prentice Hall PTR, Upper Saddle River, United States of America, 2004. 29 psl.

[FL14] Martin Fowler and James Lewis. Microservices. https://martinfowler.com/articles/microservices.html, 2014.

[Fow17] Susan J. Fowler. *Production-Ready Microservices*. O'Reilly Media, Sebastopol, United States of America, 2017. 1 psl.

[HKL09] Volker Hockmann, Heinz D. Knoell ir Ernst L. Leiss. *Encyclopedia of Multimedia Technology and Networking, Second Edition*. 2009. 173 skyrius.

[JL17] Vineet John and Xia Liu. A survey of distributed message broker queues. https://arxiv.org/pdf/1704.00411.pdf, 2017.

[Joh18] Espen Johnses. *Reliable Asynchronous Communication in Distributed Systems*. Magistrinis darbas, Bergensis Universitas, 2018.

[Loi18] J. Loisel. Soap vs rest (why comparing them is a nonsense. https://octoperf.com/blog/2018/03/26/soap-vs-rest/, 2018.

[Mak17] Joni Makkonen. *Performance and usage comparison between REST and SOAP web services*. Magistrinis darbas, Aalto University, 2017.

[New15] Sam Newman. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, Sebastopol, United States of America, 2015. 39 psl.

[Ray03] Eric Steven Raymond. *The Art of UNIX Programming*. Addison-Wesley, Upper Saddle River, United States of America, 2003. 1 psl.

[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.

[SLi00] David S.Linthicum. *Enterprise Application Integration*. Addison-Wesley, Upper Saddle River, United States of America, 2000. 1 psl.

[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.