VILNIAUS UNIVERSITETAS
INFORMATIKOS INSTITUTAS
PROGRAMŲ SISTEMŲ KATEDRA

Automatinė konfigūracijos valdymo sistema

Automatic Configuration Management System

Bakalauro baigiamasis darbas

Atliko:	Martynas Struckus	(parašas)
Darbo vadovas:	J. Asist. Oleg Mirzianov	(parašas)
Darbo recenzentas:	Partn. Doc. Andrius Adamonis	(parašas)

Vilnius – 2020

Santrauka

Šio darbo metu buvo siekiama automatizuoti konfigūracijos valdymo procesą. Augantis programuotojų ir kuriamų produktų skaičius įmonėse kartu lemia ir didejantį konfigūracijos valdymo poreikį. Todėl šio proceso automatizavimas padėtų įmonėms sutaupyti laiko bei pinigų. Tam buvo sukurtas sistemos prototipas, gebantis automatiškai kurti ir konfigūruoti reikalingus konfigūracijos valdymo komponentus, naujiems projektams. Gautas automatizavimo procesas - labai greitas, tačiau ne visada kokybiškas. Sudėtingų architectūrinių poreikių aplikacijoms, automatinis konfigūracijos valdymas atliekamas prasčiau, todėl, tokiais atvejais, gali būti reikalingas DevOps specialisto įsitraukimas. Visgi, pakankamai išvystyta tokio pobūdžio sistema ateityje gali būti labai naudingas ir galingas įrankis.

Raktiniai žodžiai: automatizavimas, DevOps, CI/CD, debesų kompiuterija, konfigūracijos valdymas

Summary

The aim of this work was the automation of configuration management process. A growing number of software engineers and development products in companies also determines increasing need for configuration management. Therefore, the automation of this process would help companies to save their time and money. Because of that a system prototype was developed which is capable of creating and configuring required configuration management items automatically for new projects. The resulting automation process is inordinary fast but not always a good quality one. For applications that have complicated architectural needs automatic configuration management is performed worse, therefore, involvement of DevOps specialist is mandatory in these type of cases. However, a sufficiently developed system of this kind could be an extremely useful and powerful tool in the future.

Keywords: automation, DevOps, CI/CD, cloud computing, configuration management

TURINYS

Įvadas

Šiame darbe bus siekiama sukurti automatinę konfigūracijos valdymo sistemą, kuri gebėtų automatiškai kurti reikalingus konfigūracijos valdymo elementus, atsižvelgiant į reikalaujamus naujos aplikacijos resursus ir tipą. Praktikoje DevOps specialistams tenka atlikti panašaus pobūdžio programinės įrangos įdiegimo proceso konfigūravimo užduotis. Stambios įmonės, kuriančios programinę įrangą, tokios kaip „Bentley Systems“, būtent šiam procesui atlikti skiria nuo 2 iki 5 žmonių. Augantis programuotojų komandų ir kuriamų produktų skaičius generuoja vis daugiau darbo, todėl šio proceso automatizavimas padėtų sutaupyti organizacijoms brangaus laiko ir pinigų. Taip pat patys DevOps specialistai, turėtų daugiau laiko atlikti kitiems darbams, todėl tokia sistema turėtų atitinkamą dydį teigiamos įtakos konfigūracijos valdymo proceso efektyvumui, kiek organizacija skiria resursų programinės įrangos įdiegimo proceso palaikymui. Kadangi, 2020 metais, debesų kompiuterijos technologijos (angl. *cloud computing technologies*) toliau smarkiai populiarėja (prognozuojamas 17% debesijos paslaugų rinkos augimas per 2020 metus [Inc19]), kuriama sistema bus paremta debesies pagrindu, orientuota organizacijoms naudojančioms tam tikrą debesies tiekėją ir platformą, pritaikyta naudoti „Agile“ programų kūrimo procesui bei palaikys tinkamas CI/CD kompetencijas, taip užtikrinant sistemos aktualumą šiuolaikinėms tendencijomis. Tokio tipo sistema turėtų būti dinamiška ir lengvai plečiama, kadangi kiekviena organizacija užsiima skirtinga veikla ir kuria skirtingų tipų aplikacijas, todėl bus stengiamasi išlaikyti ir dinamiškumo aspektą.

Darbo **tikslas** - parinkti dinamišką konfigūracijos valdymo problemos sprendimo būdą ir pagal jį sukurti automatinės konfigūracijos valdymo sistemos prototipą, kuris gebėtų automatiškai kurti reikalingus konfigūracijos valdymo elementus bent kelių skirtingų tipų aplikacijoms, idealiu atveju be arba tik su minimalia DevOps specialisto pagalba ir būtų tinkamas naudoti organizacijoms.

Keliami darbo **uždaviniai**:

1. Išanalizuoti tinkamas debesies platformas ir pasirinkti vieną iš jų.
2. Sudaryti automatiškai kuriamų konfigūracijos valdymo elementų sąrašą.
3. Atlikti preliminarių sistemos technologinių poreikių analizę, pasirinkti naudojamus programinės įrangos karkasus ir kitas technologijas.
4. Išanalizuoti konfigūracijos valdymo elementų kūrimo detales ir sudaryti dinamišką automatizavimo sprendimą.
5. Sudaryti sistemos valdymo planą.
6. Išanalizuoti ir paruošti automatizavimui reikalingus resursus.
7. Sudaryti duomenų saugojimo schemą.
8. Implementuoti ankščiau sudarytą konfigūracijos elementų kūrimo automatizavimo sprendimą.
9. Realizuoti sistemos kontrolinius reikalavimus.
10. Sukurti minimalią naudotojo sąsaja.

11. Įvertinti automatizavimo kokybę, eksperimentuojant su skirtingais aplikacijų reikalavimais.
12. Išanalizuoti automatizavimo rezultatus ir sudaryti sistemos patobulinimų ir pagerinimų sąrašą.

Šio darbo pabaigoje, kaip rezultatą, numatoma gauti apibūdintos sistemos prototipą, tinkamą naudoti organizacijoms, kurios kuria debesų kompiuterijos pagrindu paremtą programinę įrangą.

Pradžioje bus analizuojami debesies tiekėjai ir platformos bei galiausiai pasirinktas vienas iš jų. Pasirinkta debesies platforma turės daug įtakos ir šiek tiek apribos galimus technologijų pasirinkimus, kadangi skirtingos debesies platformos, skirtingai integruojasi su įvairiomis išleidimo valdymo (angl. *release management*) sistemomis ir produktais tokiais kaip „Jenkins“, „GitLab“, „Azure DevOps“, „Bitbucket“ ir kt. Taip pat kadangi debesies platformų prenumeratos yra mokamos, todėl bus atsižvelgiama ir į jų kainą.

Sekantis žingsnis yra apibrėžti automatizavimo apimtį, todėl bus sudaromas automatiškai kuriamų abstrakčių konfigūracijos valdymo elementų sąrašas. Tada bus stengiamasi atrinkti tinkamas technologijas, leidžiančias kurti sąraše nurodytus elementus bei vertinamos jų integracijos galimybės. Parinkus tinkamas technologijas galima pereiti prie būsimos sistemos projektavimo, programavimo kalbos ir programinės įrangos karkasų pasirinkimo.

Turint pakankamai informacijos apie didžiąją dalį sistemos techninių detalių, galima pradėti sudaryti teorinį automatizavimo sprendimą. Išsiaiškinus kokios yra kiekvienam abstraktaus konfigūracijos valdymo elementui reikalingos įvestys ir reikalavimai, galima sudaryti automatizuojamų elementų sukūrimo tvarką ir privalomų įvesčių sąrašą. Toks sąrašas vėliau gali būti naudojamas kaip duomenų modelis automatizuojamam procesui. Galiausiai lieka išanalizuoti ryšius tarp konfigūracijos valdymo elementų ir apibrėžti automatizavimo procesą.

Tam, kad ši sistema išliktų valdoma ir ja nebūtų galima piknaudžiauti, būtina priimti tam tikrus kontrolės ir valdymo sprendimus. Siekiant palaikyti vieną iš pagrindinių šios sistemos naudų - padėti sutaupyti organizacijai pinigų, o ne juos išleisti, reikalinga apibrėžti tam tikrą patvirtinimo procesą, kuris padėtų atlikti nenaudingų ir neekonomiškų sprendimų filtravimo vaidmenį. Todėl bus apibrėžiami naudotojų tipai ir jiems suteiktos teisės bei leidimai, leidžiantys atlikti tam tikrus veiksmus. Priklausomai nuo naudotojų teisių realizacijos būtina įvertinti ir teisių valdymo detales bei apibrėžti teisiu valdymo procesą. Pasinaudojant priimtais sprendimais ir įžvalgomis, galiausiai lieka sudaryti bendrą kontrolės ir valdymo planą.

Atsižvelgiant į jau sudarytą automatizavimo proceso eigą ir kontrolės reikalavimus, galima atrinkti duomenis, kuriuos reikia saugoti. Įvertinus ryšius tarp duomenų bus galima sudaryti duomenų saugojimo schemą ir duomenų bazės tipą. Kadangi bus tiesiogiai dirbama su duomenų baze, tam bus pasirinktos papildomos technologijos leidžiančios dirbti su duomenimis, tokios kaip ORM („Object-relational mapping“).

Sistemos automatizavimo procesui bus reikalingi resursai, tokie kaip įdiegimo skriptai (angl. *deployment scripts*) ar baziniai išleidimo kanalai (angl. *base release pipelines*), todėl bus sudaromas reikalingų sistemos resursų sąrašas. Sekantis žingsnis - sistemos programavimas ir resursų paruošimas. Kadangi prieš tai buvo paruoštas teorinis automatizavimo sprendimas, galima pradėti jį realizuoti. Sukurtas algoritmas turėtų būti naudojamas kaip pagrindinis sistemos funkcionalu-

mas. Kad algoritmas būtų efektyvus tam reikės paruošti sistemai reikalingus resursus naudojimui. Kuomet automatizavimo algoritmas bus pilnai paruoštas ir tinkamai ištestuotas, bus galima pereiti prie kitų sistemos reikalavimų implementavimo. Taip pat, kad būtų galima tinkamai naudotis sistemos teikiamu funkcionalumu reikalinga sukurti ir naudotojo sąsają.

Galiausiai parengta sistema bus ištestuota automatiškai atliekant skirtingų tipų aplikacijų konfigūravimą, bus vertinama automatizavimo kokybė ir stebimi gauti rezultatai. Pasitelkiant gautus rezultatus bus galima pastebėti sistemos trūkumus, kuriuos būtų galima patobulinti, todėl kartu bus sudaromas patobulinimų ir pagerinimų sąrašas.

1. Pasirinktų technologijų ir apimties apibrėžimas

1.1. Debesies platformos pasirinkimas

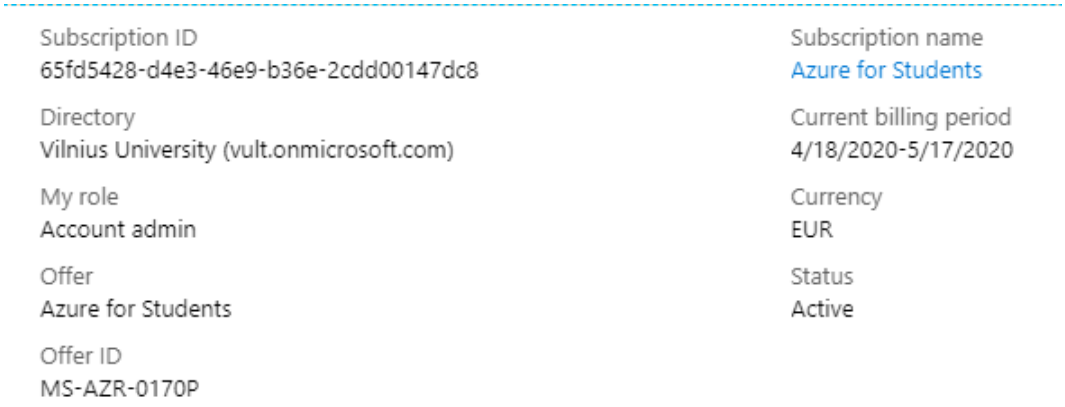
Debesies platformos pasirinkimas yra įtakingas sprendimas sistemos mąstu, nes tai apriboja tam tikras technologines galimybes. Tam, kad tyrimas būtų aktualus buvo nuspęsta rinktis tarp trijų populiariausių debesies platformų: „Amazon Web Services (AWS)“, „Microsoft Azure“ ir „Google Cloud Platform“. Remiantis 2019 metų duomenimis, „Amazon Web Services (AWS)“ bendra debesijos rinkos dalis siekė 32,4%, „Microsoft Azure“ - 17,6% bei „Google Cloud Platform“ - 6,0% [Cha20].

Vienareikšmiškai „Amazon Web Services“ yra pati populiariausia ir plačiausiai naudojama debesies platforma. AWS turi mokymosi programą „AWS Educate“, leidžiančią moksleiviams ir studentams plačiau pažinti pačią debesies platformą, teikia vaizdinę mokymosi medžiagą ir dokumentaciją. Kitą vertus, tai nėra naujovė, kiti konkurentai tarp jų ir „Microsoft Azure“ bei „Google Cloud Platform“ taip pat siūlo atitinkamas mokymosi programas. Visos trys platformos taip pat gali suteikti studentams prieiga prie resursų naudojimo su tam tikru kainos limitu. AWS ir „Microsoft Azure“ studentams nemokamai suteikia 100$ vertės prenumeratas [Ser20] [Cor20c], „Google Cloud Platform“ - 50$ vertės prenumeratą [LLC20a]. Be to, tam kad, prieiga prie resursų būtų suteikta, studento organizacija ar mokymosi įstaiga turi būti įtraukta į tam tikrą mokymosi įstaigų sąrašą. Vilniaus Universitetas yra įtrauktas tik į „Microsoft Azure“ mokymosi įstaigų sąrašą, todėl yra galimybė naudotis šios platformos studetams teikiamais privalumais. Dauguma kitų palyginimo kriterijų yra priklausomi nuo, kuriamos programinės įrangos pritaikymo detalių dėl to, šiuo konkrečiu atvėju, jie nėra pakankamai reikšmingi, kad įtakotų debesies platformos pasirinkimą, kadangi debesies platforma bus naudojama tik kaip automatizavimo aplinka (žiūr. 1 lentelę).

1 lentelė. Debesies tiekėjų suvestinė

Debesies platforma	**Prenumeratos vertė**	**Mokymosi programa**	**Privalumai VU studentams**
Amazon Web Services	100$	✓	✗
Microsoft Azure	100$	✓	✓
Google Cloud Platform	50$	✓	✗

Atsižvelgiant į šiuo atveju svarbiausią kriterijų - kainą, „Microsoft Azure“ platforma buvo patraukliausias pasirinkimas, todėl buvo nuspręsta toliau naudoti šią debesies platformą. Pasinaudojant, Vilniaus Universiteto studentams teikiamais „Microsoft Azure“ privalumais, buvo sukurta studento paskyra ir suteikta nemokama prenumerata, galiojanti vienerius metus ir leidžianti naudotis šios debesies platformos resursais (žiūrėti 1 pav.).



1 pav. „Microsoft Azure“ prenumerata

1.2. Automatiškai kuriami konfigūracijos valdymo elementai

Šiame skyriuje bus sudaromas automatiškai kuriamų konfigūracijos elementų sąrašas. Šis sąrašas bus naudojamas apibrėžti būsimos sistemos funkcionalumo apimtį. Sąrašo elementai bus atrenkami atsižvelgiant į CI/CD bendrąsias praktikas ir problematikos aktualumą.

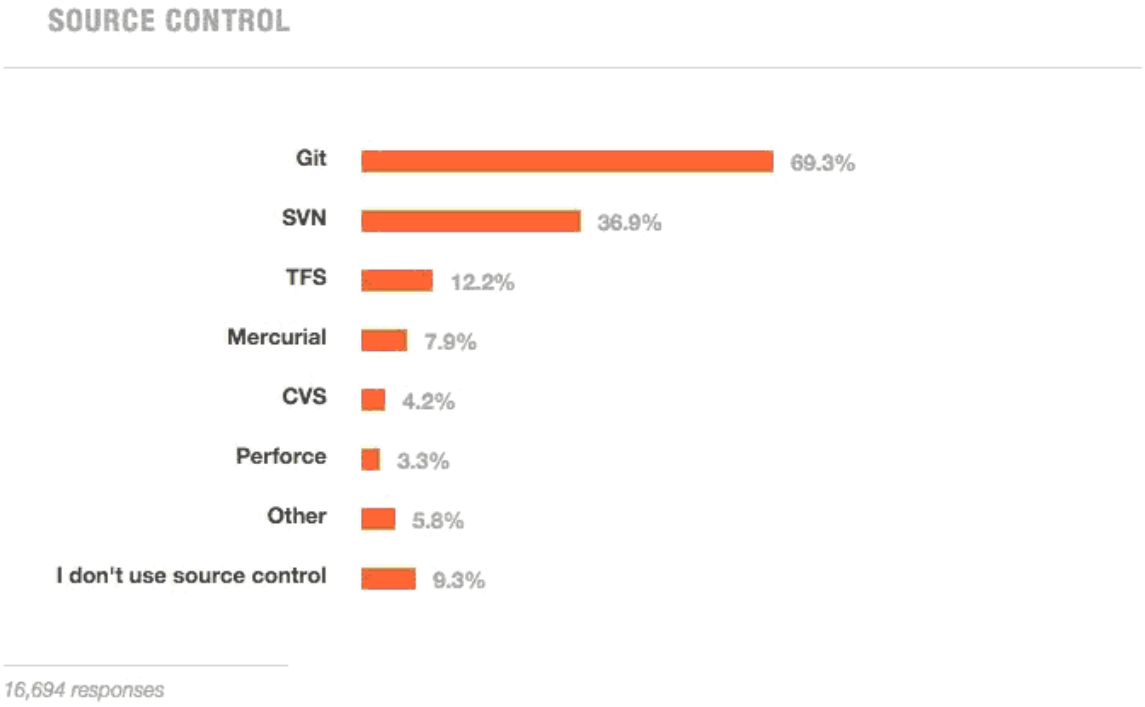
Tipinis CI/CD procesas susidaro iš šių abstrakčių fazių [LLC20b]:

1. Sudėjimas (angl. *build*).
2. Testavimas (angl. *testing*).
3. Išleidimas (angl. *deploy*).
4. Automatinis testavimas (angl. *automated testing*).
5. Įdiegimas į produkcinę aplinką (angl. *deploy to production*).

Šis automatinis procesas yra konfigūruojamas ir palaikomas DevOps specialistų stambiose organizacijose. Be to visos prieš tai išvardintos fazės reikalauja skirtingos konfigūracijos priklausomai nuo aplikacijos techninių detalių, pavyzdžiui sudėjimas skiriasi nuo naudojamos programavimo kalbos ir/ar naudojamų programinės įrangos karkasų, išleidimas - priklauso nuo aplikacijos tipo (internetinė aplikacija, mobili aplikacija). Tačiau automatizuoti tik CI/CD proceso konfigūravimą neužtenka. Tokio tipo procesas prasideda tik tuomet, kai programuotojas atlieka tam tikrus programinio kodo pakeitimus ir nusiunčia juos į versijavimo sistemą [LLC20b]. Bet net ir prieš pradedant rašyti programinį kodą, komandos taip pat planuoja būsimus darbus, kuriuos reikia atlikti rotacijos bėgyje (kalbant apie „Agile“ procesą). Šiam procesui dažnai naudojami tam tikri projektų valdymo įrankiai pavyzdžiui: Basecamp, Trello, Jira ir kt. [Glo17] leidžiantys sekti progresą ir valdyti užduotis, kurie taip pat yra konfigūruojami juos išmanačių specialistų, todėl yra galimas papildomas automatizavimas ir šioje srityje.

Sekant programų kūrimo „Agile“ procesų eiga, pirmasis žingsnis yra planavimas, kas atitinkamai automatizavimo ir kuriamo prototipo atžvilgiu būtų, darbų lentos/sąrašo (angl. *backlog*) sukonfigūravimas [Fag20]. Be abejo, tam bus naudojami specialūs įrankiai, kurių konfigūracijos automatizavimas potencialiai gali būti sudėtingesnis priklausomai nuo galimybių, todėl renkantis projektų valdymui tinkamus įrankius pagrindinis kriterijus bus automatizavimo galimybės ir prieі-

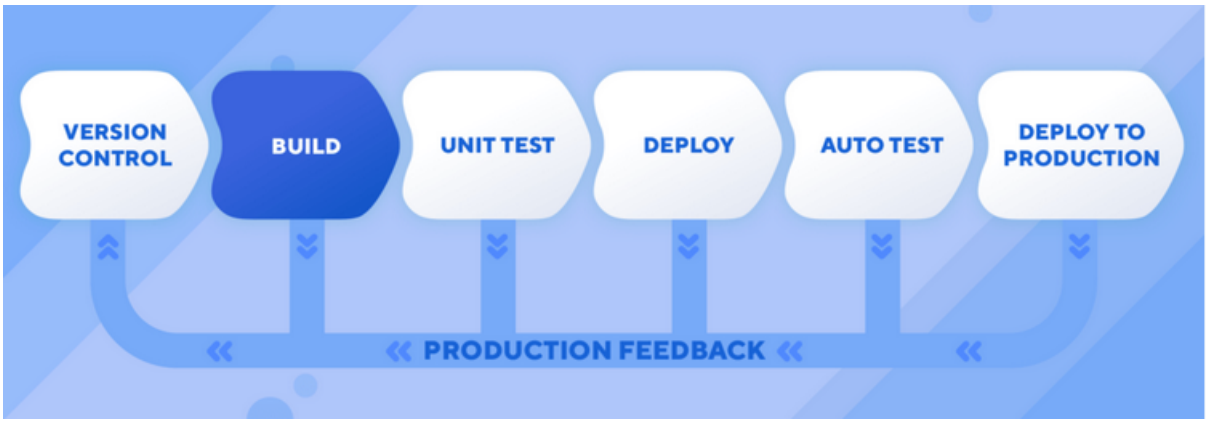
namumas. Toliau, pereinant prie programavimo dalies, būtinas ir bene svarbiausias komponentas, kurį pirmą pasiekia programuotojo atlikti programinio kodo pakeitimai yra versijavimo sistema [War18]. Vienintelis versijavimo sistemos reikalavimas yra tas, jog tai būtų paskirtyto tipo versijavimo sistema (angl. distributed version control). Taip yra todėl, nes šio tipo versijavimo sistemos yra naujausios, daugiausiai naudojamos pasauliniu mąstu, greitesnės bei labiau patikimos lyginant su centralizuotomis versijavimo sistemomis (angl. centralized version control sistemos) [Nag19]. Populiariausios šio tipo versijavimo kontrolės sistemos yra „Git“ ir „Mercurial“ [Sny15]. Vis dėl to renkantis vieną iš šių dviejų versijavimo kontrolės sistemų, pranašesnė yra „Git“, kadangi tai yra be abejonės pati populiariausia sistema (žiūrėti 2 pav.), o šiuo atveju populiarumo aspektas yra svarbiausias todėl, nes tai labiausiai įtakoja kuriamo prototipo aktualumą šiuolaikinėms tendencijoms. Kas liečia versijavimo sistemos konfigūravimą, darbo metu bus pasirinktas reikalavimus atitinkantis įrankis bei automatiškai kuriamos pirminio kodo repozitorijos (angl. *source code respositories*) kiekvienam naujam projektui.



2 pav. „Stackoverflow“ 2015 metais atliktos versijavimo kontrolės sistemų naudojimo apklausos rezultatai [Sny15]

Sekant tolimesne proceso eiga, po programinio kodo pakeitimų patekimo į versijavimo sistemą prasideda CI/CD procesas. Kiekvieną kartą atsiradus programos kodo pakeitimams versijavimo sistemoje, pagal CI/CD kompetencijas, automatiškai pradedamas kodo kompiliavimas ir sudėjimas (angl. *build*) (žiūrėti 3 pav.). Šis mechanzimas turėtų būti susietas su pirminio kodo repozitorija iš kur galėtų pasiekti programinį kodą, tinkamai apdorotį jį ir supakuoti tolimesniam naudojimui. Prieš atliekant tolimesnius veiksmus, būtina užtikrinti, kad programos vienetų testai (angl. *unit tests*) veikia pagal savo paskirtį ir grąžiną teigiamą rezultatą, priešingu atveju neverta toliau tęsti proceso [LLC20b]. Supakuotą programinį kodą, priklausomai nuo aplikacijos tipo,

reikia tinkamai įdiegti į „Microsoft Azure“ platformą. Tam reikalingas išleidimo kanalas (angl. *release pipeline*), gebantis automatiškai sukurti reikalingus „Microsoft Azure“ resursus bei panaudoti supakuotą programinį kodą įdiegimui į skirtingas aplinkas. Kokybišką perėjimą iš testavimo aplinkos į produkcinę aplinką padeda užtikrinti automatinis testavimas, todėl reikalinga atsižvelgti ir į šį reikalavimą, automatizuojant konfigūravimą. Išleidimo kanalas panašiai kaip ir sudėjimo kanalas turėtų būti automatiškai paleidžiamas tuo atveju, kuomet pateikiama nauja supakuoto kodo versija. Visi šie elementai tinkamai sukonfigūruoti sudaro pilną CI/CD darbo kelią, todėl to pakaktų palaikyti reikiamas kompetencijas.



3 pav. Tipinio apibendrinto CI/CD proceso struktūra [LLC20b]. Versijavimo kontrolė, sudėjimas, vienetų testai, įdiegimas, automatiniai testai, įdiegimas į produkcinę aplinką

Galiausiai tyrimo eigoje buvo sudarytas automatiškai kuriamų ar/ir konfigūruojamų komponentų užduočių sąrašas:

1. Sukonfigūruoti darbų lentą/sąrašą (angl. *backlog*).
2. Atlikti reikalingus versijavimo sistemos konfigūracijos veiksmus, sukurti pirminio kodo repozitoriją.
3. Sukurti programinio kodo sudėjimo (angl. *build*) sprendimą.
4. Pridėti atitinkamą vienetų testavimo (angl. *unit testing*) sprendimą.
5. Sukurti ir sukonfigūruoti išleidimo kanalą, gebantį įdiegti programinę įrangą į skirtingas aplinkas bei atlikti automatinį testavimą.

Visi sąraše nurodyti elementai apibrėžia kuriamo prototipo funkcines galimybes ir automatizavimo apimtį. Darbo metu, sėkmingai sukūrus tokios apimties automatizavimo procesą, būtų pagerinamas konfigūracijos valdymo proceso efektyvumas organizacijos mąstu.

1.3. Naudojami konfigūracijos valdymo įrankiai

Siekiant pradėti automatiškai kurti konfigūracinius komponentus, būtina pradžioje išsirinkti tam tinkamus įrankius. Galimi variantai priklauso nuo debesų platformos. Iš visų galimų įrankių variantų, kurie nesunkiai integruojasi su „Microsoft Azure“ platforma, remiantis „Microsoft“ dokumentais, daugiausiai tam siūlomi variantai yra: „Azure DevOps“, „Jenkins“ kartu su „GitHub“,

bei keletą kitų atviro kodo (angl. *open-source*) įrankių variacijų su prieš tai išvardintomis sistemomis [Cor20d].

„Azure DevOps“ yra konfigūravimo sistema, kuri yra tinkama diegti programinę įrangą į „Microsoft Azure“ debesies platformą, tačiau nebūtinai šiam tikslui ji yra sukurta [McL20]. Visgi ši sistema yra gan didelė ir susideda iš daugybės skirtingų servisų [Cor20b], kurių funkcionalumą būtų galima panaudoti konfigūravimo procese. Vienas iš jų yra „Azure Repos“ versijavimo sistema, palaikanti „Git“ tipo pirminio kodo repozitorijas. Be abejonės „Azure DevOps“ servisų rinkinyje yra ir „Azure Pipelines“ servisas, leidžiantis sukurti kodo sudėjimo, testavimo ir išleidimo darbo eigas (angl. *workflows*). Galiausiai šioje sistemoje yra ir integruotas projektų valdymo įrankis - „Azure Boards“, skirtas užduočių planavimui ir valdymui. Todėl šiam funckionalumui nereikėtų ieškoti papildomų automatizuojamų įrankių ar sistemų. Pats didžiausias šios sistemos privalumas yra tas, jog visi reikalingi komponentai yra „po vienu stogu“, nebūtina ieškoti papildomų integracijų bei visi prieš tai minėti servisai gali būti lengvai panaudojami konfigūravimui ir turi „REST API“ sąsajas [Cor16a].

„Azure DevOps“ pranašumai:

- Sistema turi pakankamai funkcionalumo padengti visą automatizuojamą procesą.
- Turi puikią „REST API“ sąsają, lengva automatizuoti.
- Suteikia prieiga prie „Microsoft Azure“ resursų įdiegimo skriptų šablonų.
- Yra galimybė naudoti nemokamai.

„Azure DevOps“ trūkumai:

- Sunku papildomai išplėsti esamą sistemos funkcionalumą.

„Jenkins“ yra atviro kodo automatizavimo serveris, leidžiantis automatiškai kompiliuoti ir įdiegti programinę įrangą į pasirinktą platformą, tarp jų ir „Microsoft Azure“. Tai yra vienas iš populiariausių, turintis virš tūkstančio papildymų, labai lankstus ir lengvai konfigūruojamas įrankis [Jen20]. Tačiau „Jenkins“ atlieka tik sudėjimo ar/ir išleidimo kanalo funkcijas, todėl renkantis šį įrankį bus būtina kartu pasirinkti ir keletą kitų įrankių likusiam automatizavimo procesui patengti. Likusias funkcionalumo spragas galėtų užpildyt „GitHub“ integracija [Blo17], suteikiant prieigą prie darbų lentos/sąrašo bei versijavimo sistemos. Todėl, „Jenkins” labiau tiktų naudoti kartu su kitomis reikalingomis integracijomis ir konfigūracijos valdymo sistemomis, bet ne atskirai. Be to, yra galimybė pasinaudoti „Jenkins“ funkcionalumu integruojant šį įrankį su pačia „Azure DevOps“ sistema [Ben18]. Pagrindinis trūkumas lyginant su „Azure DevOps“ yra tas, jog „Jenkins“ iš karto neturi „REST API“ sąsajos, kurią būtų galima pradėti naudoti automatizavimui, tokio tipo funkcionalumo reiktų ieškoti tarp sistemos plėtinių (angl. *extensions*), kas dar labiau komplikuotų automatizavimą.

„Jenkins“ pranašumai:

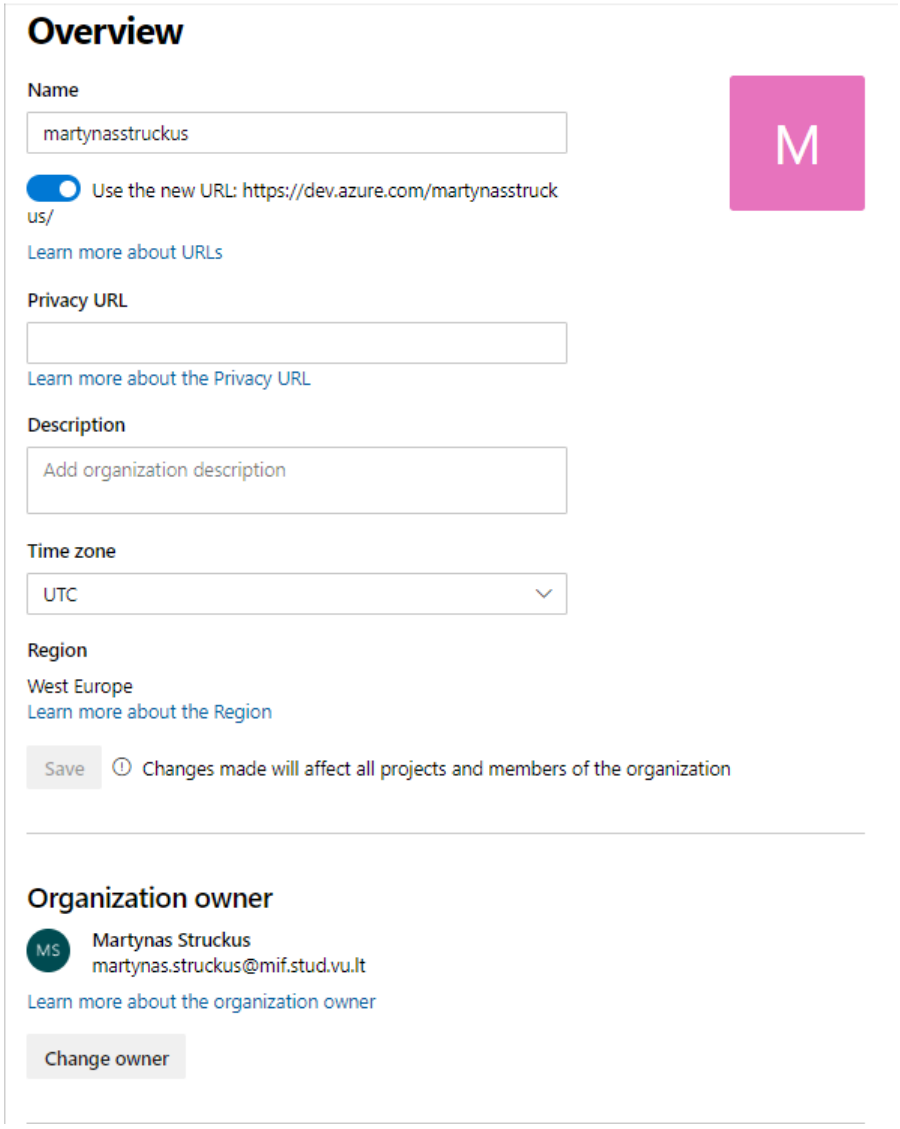
- Atviro kodo, lengvai plėčiama sistema.
- Turi daugiau negu 1000 plėtinių (angl. *extensions*).
- Yra galimybė naudoti nemokamai.

„Jenkins“ trūkumai:

- Turi mažai įmontuotų funkcijų (angl. *build-in features*).
- Reikalinga integracija su kitais įrankiais ar sistemomis.
- Sudėtingesnis automatizavimo procesas.

Apibendrinant, abu nagrinėti variantai „Azure DevOps“ ir „Jenkins“ kartu su „GitHub“ iš dalies buvo abu tinkami. Vis dėl to „Azure DevOps“ „REST API“ suteikia didelį pranašumą, lyginant su tuo ką gali pasiūlyti „Jenkins“, automatizavimo prasme. Tačiau „Jenkins“ sudaro puikias galimybės išplėsti sistemos funkcionalumą naudojant plėtinius, visgi automatizavimo kriterijus šiuo atvėju turi didesnį svorį, todėl buvo pasirinkta „Azure DevOps“ konfigūravimo sistema.

Galiausiai buvo pasinaudota prieš tai aktyvuota „Microsoft Azure“ naudotojo paskyra bei sukurta nemokama „Azure DevOps“ aplinka tolimesniam darbui (žiūrėti 4 pav.).



4 pav. „Azure DevOps“ testinės aplinkos apžvalga

Ši „Azure DevOps“ organizacija yra pasiekiama tik iš VU MIF domeno, todėl papildomų apsaugos priemonių imtąsi nebuvo.

2. Sistemos projektavimas ir pasiruošimas

2.1. Sistemos valdymo planas

Šiame poskyryje bus sudaromas sistemos valdymo planas, kurio pagrindinis tikslas yra užtikrinti, kad sukurta sistema būtų tinkamai kontroliuojama bei ja nebūtų piknaudžiaujama.

Pagrindinis automatinės konfigūracijos valdymo sistemos tikslas yra padėti organizacijai sutaupyti pinigų, tačiau kadangi sistema gebės automatiškai kurti resursus, kurie yra išlaikomi iš organizacijos biudžeto, šio tikslo įgyvendinimas gali nukrypti priešinga linkme, jeigu sistema nebus tinkamai valdoma. Greitai ir paprastai kuriami mokami resursai galėtų sparčiai padidinti organizacijos išlaidas, todėl buvo reikalinga surasti šios problemos sprendimo būdą.

Šios sistemos naudojimas, projekto gyvavimo cikle, turėtų vykti tuomet kai projekto detalės ir kita reikalinga informacija yra pateikta bei patvirtinta tam tikrų atsakingų asmenų organizacijoje, tačiau kol kas sistema neturi prieigos prie šios informacijos, dėl ko atsiranda valdymo spragų. Stambios programinę įrangą kuriančios įmonės naudoja standartizuotą užklausų procesą (angl. *standardized request process*), kuris padeda kontroliuoti įmonės išlaidas bei yra svarbi projektų valdymo dalis [Inc20b]. Remiantis „Wrike“, organizacijos teikiančios projektų valdymo programinę įrangą tokioms kompanijoms kaip „Google“, „Airbnb“, „Hootsuite“ ir kt. [Inc20c], pateikta informacija, užklausų formos, projektų valdymo procese yra gyvybiškai svarbi detalė, kadangi neužbaigti reikalavimai yra viena iš pagrindinių priežasčių, kodėl projektai žlunga [Inc20b].

Kadangi konfigūracijos valdymo automatizavimui yra reikalinga dalis informacijos apie naują projektą, pavyzdžiui prie projekto dirbanti komanda, projektui reikalingi resursai ir kt., sistemos valdymas bus paremtas standartizuotu užklausų procesu, to pasakoje bus papildomai implementuojama sistemos projektų valdymo dalis. Užklausų formos bus kuriamos ir valdomos pačioje sistemoje, patvirtina užklausa bus reikalinga norint pradėti konfigūracijos automatizavimo procesą. Taip remiantis gerosiomis projektų valdymo praktikomis [Mul17], bus užpildomos sistemos valdymo spragos.

Užklausos formos formatas labai priklauso nuo organizacijos poreikių [Mul17], visgi bus kuriamas sistemos prototipas, kuris neturi pilnai atitikti visų organizacijos reikalavimų, todėl bus tiesiog reikalaujama pateikti bendrą informaciją apie naują projektą, kuri nepriklauso nuo organizacijos struktūros. Pagal bendrąsias užklausų formų sudarymo praktikas [Han18] bei remiantis pavyzdiniais šablonais [Inc20a], buvo suformuotas reikalingos informacijos sąrašas:

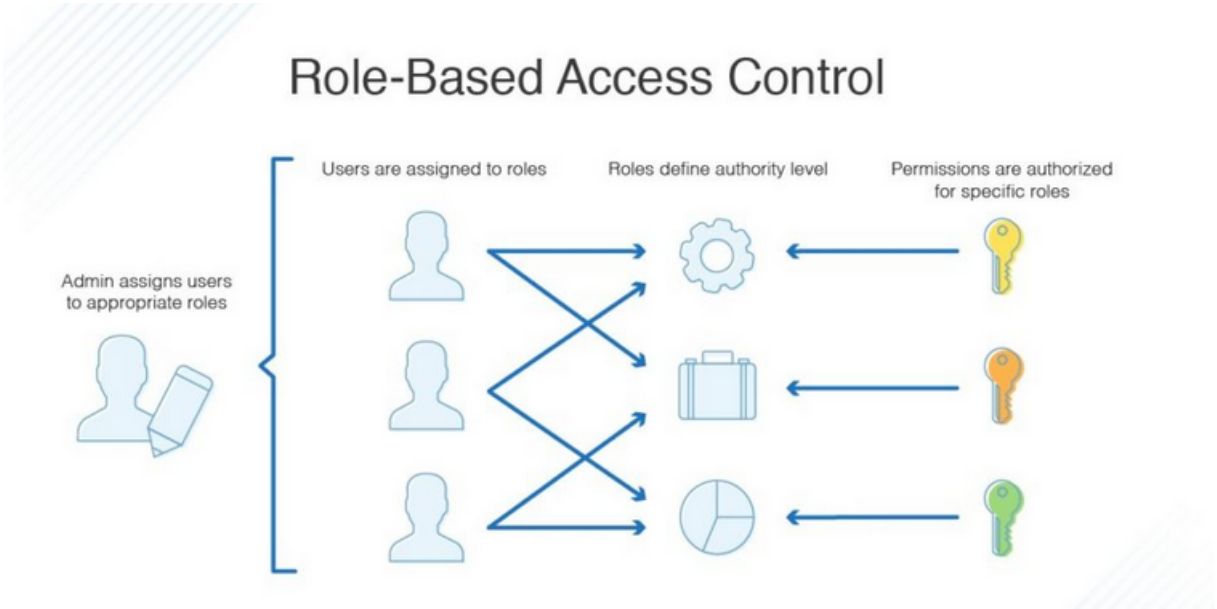
- Projekto pavadinimas.
- Projekto vadovo vardas ir pavardė.
- Numatoma mėnesinė kaina.
- Numatoma išleidimo į produkcinę aplinką data.
- Projekto aprašymas.

Kadangi pagal kuriamos sistemos naudojimo prigimtį yra žinoma, jog naudojanti organizacija, užsiima programinės įrangos kūrimu remiantis debesų kompiuterijos technologijomis bei diegia

programinę įrangą į „Microsoft Azure“ debesies platformą, ši informacija taip pat bus panaudota užklausų formos sudarymui.

Atsižvelgiant į faktą, kad sistemoje bus implementuotas standartizuotas užklausų procesas, tai reikškia, jog sistemos naudotojai privalės turėti skirtingas privilegijas. Todėl sistemos prototipui bus taikoma vienokia ar kitokia prieigos kontrolė (angl. *access control*). Tam yra galimi du skirtingi prieigos kontrolės modeliai RBAC („Role-Based Access Control“) ir ABAC („Attribute-Based Access Control“) [Car17]. RBAC modelis kontroliuoja prieigą atsižvelgdamas į naudotojui priskirtas sistemos roles ir kokią prieigą priskirtos rolės jam suteikia. Perfrazuojant, sistemos naudotojai yra suskirstomi į tam tikras grupes, tačiau vienas naudotojas gali priklausyti keletai skirtingų grupių. ABAC modelis interpretuoja prieigos kontrolę kitaip. Pastaruoju atveju naudotojų prieiga kontroliuojama atsižvelgiant į tam tikrus juos apibūdinančius atributus, pavyzdžiui naudotojamas, kurių tipas yra „darbuotojai“ ir departamentas yra „HR“ - suteikiama prieiga prie įmonės įdarbinimo informacijos. Toks prieigos interpretavimo stilius ABAC suteikia papildomo lankstumo, tačiau jis tampa sudėtingas ir komplikuotas [DNS19].

Taigi, renkantis prieigos kontrolės modelį, būtina įvertinti kuris modelis labiau tinka kuriamai sistemai. Remiantis „Identity Automation Systems“ kompanijos, teikiančios tapatumo ir priegos valdymo sprendimus (angl. *identity and access management solutions*) patirtimi, visada reikėtų pradėti nuo RBAC modelio [Car17]. RBAC modelis yra kur kas paprastesnis ir dažnu atveju to pakanka. Esant poreikiui, visada galima išskaidyti sistemos prieigos komponentus į smulkesnias dalis, taip galiausiai pereinant prie ABAC modelio. Apsvarsčius tai, jog kuriama sistema naudos prieigos kontrolę tik standartizuotam užklausų procesui įgyvendinti, šioje vietoje nereikalingas komplikuotas sprendimas, todėl bus naudojama RBAC modelio implementacija (žiūrėti 5 pav.).



5 pav. RBAC prieigos kontrolės modelio struktūra [DNS19]

Pasirinkus RBAC modelį, iš karto žinoma, jog bus būtina apibrėžti administratoriaus rolę, kuri leidžia priskirti kitiems naudotojam jų roles. Norint identifikuoti likusias naudotojų roles, reikia detaliau pažvelgti į patį užklausų procesą. Tokio tipo procese dalyvauja dvi skirtingos šalys:

užklausos siuntėjas ir užklausos patvirtinojas (angl. *approver*). Akivaizdžiai abi šalys turi skirtingas privilegijas, užklausos patvirtintojas yra pranašenis ir turi daugiau privilegijų, leidžiančių patvirtinti ar atmesti užklausas. Tai reiškia, kad reikalinga apibrėžti patvirtinojo rolę taip pat. Kitų atvejų, kuriuose dalyvautų daugiau negu vienas naudotojas, atstovaujantis skirtingas šalis nėra, todėl pakaks dviejų naudotojo rolių.

Galiausiai apibendrinant gautus rezultatus, buvo sudarytas sistemos valdymo planas:

1. Sistemoje bus implementuotas standartizuotas užklausų procesas.
2. Prieigos kontrolė sistemoje bus valdoma RBAC modelio pagalba.
3. Pradėti automatizavimo procesą bus reikalinga patvirtinta užklausos forma.
4. Kiekviena užklausa turės priskirtą patvirtintoją.

2.2. Naudojami programinės įrangos karkasai

Šiame poskyryje bus nagrinėjami programinės įrangos karkasai su tikslu nuspręsti, kuriuos iš jų naudoti kuriant automatinės konfigūracijos valdymo sistemos prototipą.

Kuriamas automatinės konfigūracijos valdymo sistemos prototipas - debesies pagrindu sukurta programa (angl. *cloud-based application*), skirta vidiniam naudojimui. Jau yra žinoma, kad sistemai bus reikalinga naudotojo sąsaja, todėl bus nagrinejami priekinės dalies programinės įrangos karkasai (angl. *front-end frameworks*). Taip pat sistemai reikia duomenų bazės saugoti užklausų formas, todėl bus ieškomas sprendimas skirtas komunikacijai su duomenų baze. Už prieigos kontrolės valdymą, automatizavimo procesą bei darbą su duomenų baze bus atsakinga programos galinė dalis (angl. *back-end*), sekant standartiniu kliento-serverio architektūros stiliumi.

„Azure DevOps“ sistemos automatizavimui yra sukurtos klientinės bibliotekos (angl. *client libraries*), palaikančios .NET, Golang, Node.js ir Python programavimo kalbas/karkasus [Cor16b]. .NET programinės įrangos karkasas yra žinomas kaip ypatingai patikimas pasirinkimas [Sen17]. Atsižvelgiant į tai, kad programuojama sistema yra pakankamai paprasta ir nereikalauja išskirtinių atlikimo (angl. *performance*) ar apdorojimo privalumų, kuo pasižymi kitos programavimo kalbos, be to vidinėms sistemomis patikimumas yra ypač naudingas ir padeda minimalizuoti palaikymui skirtus resursus, todėl .NET šiuo atveju yra puikus programos galinės dalies variantas. Kartu su .NET programavimo karkasu bus naudojamas ir „Entity Framework“ įrankis, dėl savo integracijos su „LINQ“ užklausų kalba bei dėl galimybės automatiškai sugeneruoti duomenų bazės modelį iš programos kodo („Code-first approach“) kas leidžia sutrumpinti programavimo laiką [Kum19].

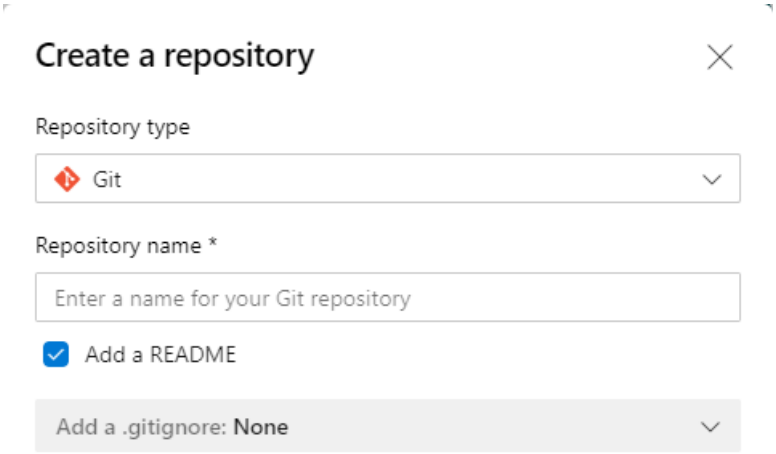
Priekinė programos dalis bus sukurta naudojant „React.js“ programavimo karkasą. Šis programinės įrangos karkasas pasižymi savo paprastumu ir virtualiu DOM („Document Object Model“) modelio perteikimu. Apskaičiuojami duomenų ir/ar būsenos pakeitimai leidžia atnaujinti tik tą puslapio dalį, kuriai jie yra aktualūs [Min16]. Šis funkcionalumas bus labai naudingas atvaizduojant automatizavimo progresą ir būsenos pasikeitimus naudotojui realiu laiku.

2.3. Automatiškai kuriamų komponentų analizė

Šiame poskyryje aprašoma automatiškai kuriamų „Azure DevOps“ komponentų analizės eiga. Pagrindinis analizės tikslas yra sužinoti kokios kiekvieno komponento sukūrimo įvestys ir kokios priklausomybės vyrauja tarp skirtingų komponentų. Pasinaudojant šia informacija galima sudaryti automatinio proceso darbo eigą.

Darbų lentos/sąrašo funkcionalumą suteikia „Azure Boards“ servisas „Azure DevOps“ sistemoje. Darbų lentos yra asocijuotos su projekto komandomis ir jeigu šis servisas yra aktyvus, darbų lenta yra sukuriama automatiškai kiekvienai projekto komadai [Cor19a]. Remiantis tuo, jog darbų lenta yra automatiškai asocijuojama su projekto komanda, norint sukurti nauja darbų lentą - reikia sukurti naują projekto komandą, todėl būtina sistemoje automatizuoti projekto komandos sukūrimą. Pagal „Azure DevOps REST API“, komandos sukūrimas yra paprastas, vienintelė privaloma įvestis - komandos pavadinimas. Kad automatiškai sukurta darbų lenta būtų tinkama naudojimui, reikalinga papildomai ją sukonfigūruoti, nustatant produkto darbų sritį bei apibrėžiant taikomą iteracinį procesą [Cor19a]. Šie konfigūraciniai pakeitimai taip pat turi būti automatizuojami, priešingu atveju, darbų lenta bus galutinai nebaigta ir netinkama naudojimui.

„Azure Repos“ yra servisas suteikiantis versijavimo kontrolės sprendimą. „Azure Repos“ palaiko dviejų tipų versijavimo sistemas: TFVC („Team Foundation Version Control“) ir „Git“. Pagal prieš tai nustatytus reikalavimus, bus naudojama tik „Git“ versijavimo sistema, be to, remiantis TFVC naudotojų atsiliepimais, ši versijavimo sistema neturėtų būti toliau naudojama naujiems projektams, kadangi „Git“ sistema yra pranašesnė daugeliu aspektų bei patys „Microsoft“ TFVC versijavimo sistemos kūrėjai, daug dirba ties „Git“ palaikymu [Mou19]. „Azure Repos“ pirminio kodo repozitorijos (angl. *source code repository*) sukūrimas reikalauja tik dviejų įvesčių: repozitorijos tipo ir pavadinimo (žiūrėti 6 pav.). Kadangi anksčiau buvo nuspręsta naudoti „Git“ versijavimo sistemą, šis tipas bus parenkamas pagal nutylėjimą, todėl lieka tik viena privaloma įvestis - pavadinimas.



6 pav. „Azure Repos“ pirminio kodo repozitorijos sukūrimo langas

Už likusių CI/CD darbo eigos komponentų realizavimą atsakingas „Azure Pipelines“ servisas [Cor19b]. Programos sudėjimo (angl. *build*) procesui „Azure Pipelines“ atitikmuo yra sudėjimo kanalai (angl. *build pipelines*). Sudėjimo kanalo sukūrimas reikalauja jau egzistuojančios pirminio

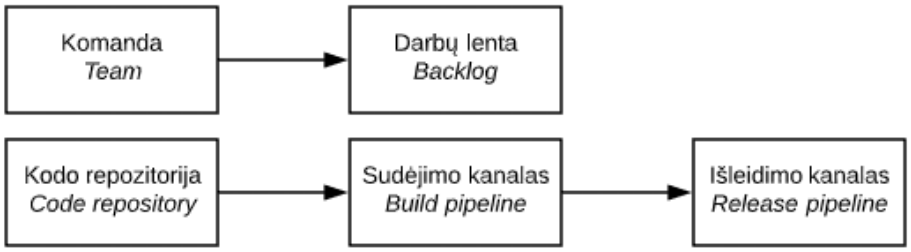
kodo repozitorijos ir kanalo pavadinimo. Tačiau net ir suteikus šias privalomas įvestis vis tiek yra gaunamas tuščias sudėjimo kanals iš kurio nėra jokio naudos, todėl kuriant sudėjimo kanalą reikia jį sukonfigūruoti taip, jog jis gebėtų apdoroti gaunamą programinį kodą. Universalus sudėjimo kanalas neegzistuoja, nes sudėjimo kanalo turinys ir procesas priklauso nuo programavimo kalbos ar programinės įrangos karkasų, todėl prieš atliekant automatizavimą, būtina žinoti šias detales. Remiantis „Azure Pipelines“ naudotojų patirtimi ir atsiliepimais [Mad20], šią problemą galima išspręsti naudojant kanalų šablonus pagal programinės įrangos reikalavimus. Šablonai gali būti perpanaudojami ir centralizuojami pačioje „Azure DevOps“ sistemoje, todėl tai neriboja kuriamojo sistemos prototipo funkcionalumo ir suteikia lankstumo.

Programinės įrangos įdiegimui procesui į tam tikras aplinkas naudojamas atitikmuo - išleidimo kanalai (angl. *release pipelines*), taip pat valdomi „Azure Pipelines“ serviso [Cor19b]. Išleidimo kanalo sukūrimas reikalauja egzistuojančio programos sudėjimo (angl. *build*) sprendimo bei kanalo pavadinimo. Išleidimo kanalo turinys priklauso nuo aplikacijos tipo, todėl šios problemos prigimtis ir jos sprendimas yra tokie pat kaip prieš tai nagrinėto sudėjimo kanalo.

Vadovaujantis „Azure DevOps“ naudotojų patirtimi, automatinių testų paleidimas turėtų būti sudėjimo/išleidimų kanalų dalis [Sel19]. Todėl testų paleidimo logika bus įtraukiama kartu į centralizuotų šablonų turinį. Skirtingos sudėjimo ir išleidimo kanalų šablonų kombinacijos leis atlikti skirtingų aplikacijų tipų ir reikalavimų konfigūracijos valdymą. Sukūrus daugiau šablonų, sistema bus atitinkamai pajėgesnė.

Kiekvienas iš minėtų konfigūruojamųjų komponentų, analizės metu, buvo sukurtas ir sukonfigūruotas rankiniu būdu testinėje „Azure DevOps“ aplinkoje. Tai padėjo geriau įsivaizduoti kaip atrodys galutinis, po automatizavimo gautas rezultatas. Kartu pasinaudojus internetinės naršyklės galimybėmis buvo stebiamos siunčiamos užklausos, kuriomis būtų galima pasinaudoti kaip pavyzdžiu, kuriant sistemos galinę dalį.

Analizės metu buvo pastebėta, kad tuščios pirminio kodo repozitorijos nepakanka susieti su sudėjimo kanalu, nes tuščia kodo repozitorija neturi egzistuojančios pagrindinės „master“ gijos. Ši gija yra automatiškai sukuriama po pirmo „commit” komandos paleidimo, kurios metu į repozitoriją yra įkeliami pradiniai programinio kodo pakeitimai. „Azure DevOps“ suteikia galimybę kuriant repozitoriją kartu paleisti ir „commit” komandą, kuri patalpina pradinį repozitorijos aprašymo failą (žiūrėti 6 pav.). Pasinaudojus šiuo funkcionalumu bus išspręsta pirminio kodo repozitorijos susiejimo su sudėjimo kanalu problema, todėl tai bus įtraukiama į konfigūracijos automatizavimo procesą.



7 pav. „Azure DevOps“ komponentų priklausomybių grandinė

Panaudojus atlikos konfigūruojamų komponentų analizės rezultatus buvo nustatytos kiekvieno komponento priklausomybes, kuriomis remiantis buvo sudaryta priklausomybių grandinę (žiurėti 7 pav.). Priklausomybių grandinė nusako konfigūracijos automatizavimo seką.

2.4. Duomenų saugojimo schema

Sistemos valdymo planavimo metu buvo prieita prie išvados, jog sistemoje bus implementuotas standartizuotas užklausų procesas bei naudojamas RBAC prieigos kontrolės modelis. Remiantis šiais faktais buvo identifikuotos trys duomenų esybės, kurių duomenys turi būti saugomi atskirose lentelėse:

- Naudotojas.
- Naudotojo rolė.
- Užklausos forma.

Tolimesni duomenų bazės schemos formavimo veiksmai buvo atlikti vadovaujantis duomenų bazės dizaino proceso gairėmis [Ngu17].

Pirmas žingsnis buvo nupręsti kokie duomenys bus saugomi kiekvienoje lentelėje. Naudotojo duomenų saugojimas nėra bendrai apibrėžtas dalykas, todėl naudotojo duomenys yra saugomi visada skirtingai, priklausomi nuo to, ko reikalauja aplikacija. Šiuo atveju, kuriamai sistemai pakanka naudotojo asmeninės informacijos duomenų: elektroninio pašto adreso ir pilno vardo, kurie apibūdina naudotojo tapatybę. Naudotojo rolė turi tik pavadinimą (administratorius, patvirtintojas), kitų papildomų duomenų nėra. Užklausos formos reikalingi stulpeliai buvo numatyti 2.1. poskyryje, papildomai prie jų buvo pridėti: užklausos būsena, projekto komandos pavadinimas, sudėjimo šablono identifikatorius ir išleidimo šablono identifikatorius, todėl kad šie duomenys turi įtakos patvirtinimo procese.

Kiekvienai iš prieš tai identifikuotų lentelių, buvo sukurti „integer“ tipo identifikatoriai atliekantys pirminio rakto funkciją. Kadangi duomenų struktūroje yra daugiau nei viena esybė, reikia nustatyti ryšius tarp jų [Ngu17]. Naudotojo esybė turi egzistuojančius ryšius tarp kitų dviejų esybių, remiantis RBAC modelio struktūra (žiūrėti 5 pav.) kiekvienas naudotojas gali turėti keletą naudotojo rolių taip pat, pagal standartizuotą užklausų procesą, kiekvienas naudotojas gali turėti daugiau negu vieną pateiktą užklausos formą. Viena naudotojo rolė gali būti priskirta keliems naudotojams, kadangi naudotojo rolė yra apibrėžiama kaip grupė, todėl laisvai galime nustatyti, jog tarp šių esybių egzistuoja „daugelis su daugeliu“ (angl. *many-to-many*) ryšys. Tuo tarpu užklausos forma turi tik vieną savininką, tačiau turi ir užklausos patvirtintoją. Tiek savininkas, tiek patvirtintojas abu priklauso naudotojo esybei, todėl šis atvejis yra šiek tiek nestandartinis. Visgi „du su daugeliu“ ryšys neegzistuoja ir kadangi daugiau negu vienas naudotojas yra asocijuojamas su užklausos forma, buvo nustatytas „daugelis su daugeliu“ ryšys. Tiesioginių ryšių tarp naudotojo rolės ir užklausos formos identifikuota nebuvo.

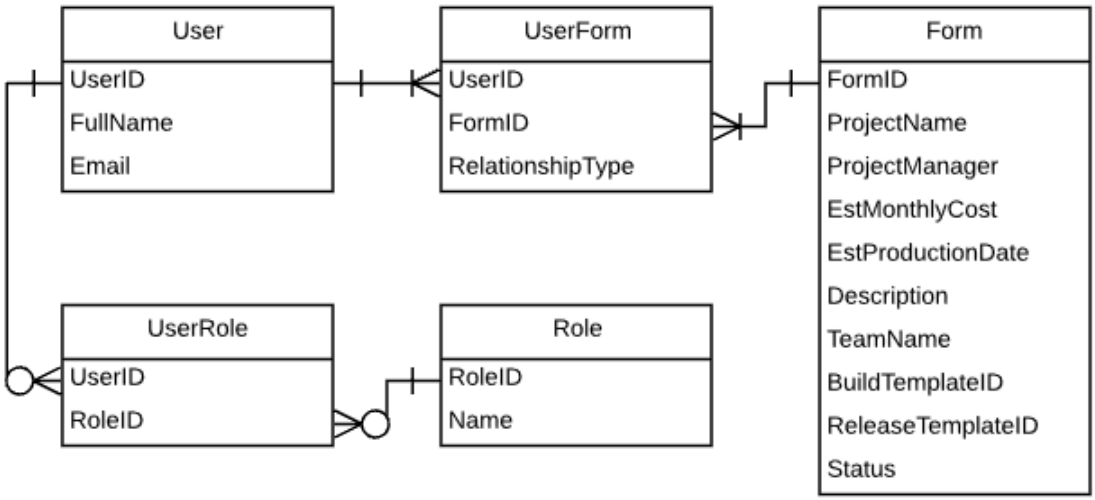
Darbo eigos metu buvo gauti šie ryšiai tarp esybių:

- Tarp naudotojo ir naudotojo rolės „daugelis su daugeliu“ ryšys.

- Tarp naudotojo ir formos „daugelis su daugeliu“ ryšys.
- Tarp naudotojo rolės ir formos ryšių nėra.

„Daugelis su daugeliu“ ryšiai duomenų bazėse yra valdomi sukuriant po papildomą tarpinę lentelę kiekvienam ryšiui, susidedančią iš ryšį sudarančių esybių identifikatorių [Bru17]. Šiuo atveju yra reikalinga apibrėžti dar dvi papildomas lenteles. Kadangi ryšys tarp naudotojo ir užklausos formos yra sudėtingesnis (naudotojas gali būti formos sąvininkas arba patvirtintojas), todėl šio ryšio tarpinėje lentelėje būtinas dar vienas papildomas stulpelis apibūdinantis ryšio tipą. Galiausiai, buvo numatyta turėti penkias duomenų bazės lenteles.

Panaudojus lentelių, jų stulpelių apibrėžimus ir ryšius tarp skirtingų lentelių buvo sudaryta duomenų bazės schema (žiūrėti 8 pav.).



8 pav. Duomenų bazės schema

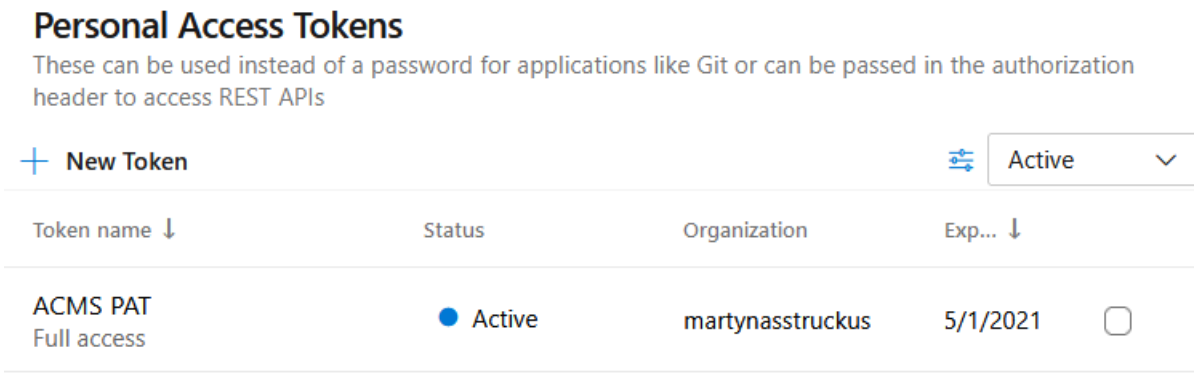
3. Sistemos prototipas

Šiame skyriuje bus aprašoma sistemos prototipo kūrimo ir programavimo darbų eiga bei gauti rezultatai. Siekiant sumažinti programavimui atlikti reikalingą laiką, buvo panaudotas „ASP.NET Core React“ techninius reikalavimus atitinkantis projekto šablonas. Visa programos biznio logika buvo apdorojama galinėje dalyje (angl. *back-end*), o priekinė dalis (angl. *front-end*) buvo atsakinga už informacijos atvaizdavimą.

3.1. Konfigūracijos valdymo automatizavimas

Konfigūracijos valdymo automatizavimui buvo naudojamos, 2.2. poskyryje paminėtos „Azure DevOps“ klientinės bibliotekos. Šios bibliotekos padengia visus „Azure DevOps REST API“ sąsajos prieigos taškus (angl. *endpoints*) ir palengvina užklausų į „Azure DevOps“ siuntimą. Prototipo programinis kodas bei naudoto sąsaja buvo kuriami anglų kalba, kadangi sistema ir tinkama naudoti ne tik Lietuvos įmonėms ir organizacijoms, bet ir užsienio.

„Azure DevOps“ yra apsaugota sistema, todėl tam, kad siunčiamos užklausos būtų apdorojamos reikia praeiti autentifikacijos procesą. „Microsoft“ šiuo atveju, kai interaktyvus prisijungimas prie „Azure DevOps“ nėra galimas, rekomenduoja naudoti PAT „Personal Access Token“ prieigos raktą [Cor20a]. Šį prieigos raktą gali sukurti bet kuris „Azure DevOps“ sistemos naudotojas, tačiau raktas negali suteikti daugiau privilegijų negu turi pats naudotojas. Taip pat PAT prieigos raktas turi savo galiojimo laiką, todėl pasibaigus jam, būtina raktą atnaujinti. Nelaimės atveju, pavyzdžiui duomenų ar informacijos nutekėjimui, visada galima panaikinti sukurtą raktą, taip jis tampa iš karto daugiau nebegaliojantis. „Azure DevOps“ autentifikacijos procesui buvo sukurtas PAT prieigos raktas (žiūrėti 9 pav.), turintis pakankamai privilegijų atlikti reikiamus automatizavimo veiksmus, o prieigos rakto reikšmė patalpina kuriamojo sistemos prototipo konfigūracijoje.



9 pav. Prototipui sukurtas PAT prieigos raktas

Automatizavimo implementacijai buvo panaudotos įgytos žinios apie kiekvieną iš 2.3 poskyryje nagrinėtų komponentų, buvo renkama informacija apie siunčiamas užklausas į „Azure DevOps“ sistemą, sekant interneto srautą (angl. *internet traffic*) naršyklėje, analizės metu. Buvo pasitelkta gauta informacija, identifikuoti ir atrinkti reikalingas bibliotekas, kurios padengia siunčiamų užklausų įvairovę.

Tam, kad automatizavimo procesas būtų lankstus bei, esant reikalui, jį būtų galima išskaidyti, kiekvienas iš automatizuojamųjų ar konfigūruojamųjų komponentų sistemos prototipo galinėje dalyje yra apdorojamas atskirai (žiūrėti 10 pav.). Taip suteikiama galimybė pilnai kontroliuoti automatizavimo procesą iš sistemos priekinės dalies. Galiausiai kiekvienam iš komponentų buvo dedikuoti aplikacijos sąsajos prieigos taškai, priimantys ir apdorojantys HTTP užklausas.

```
1  using ACMS.Requests;
2  using Microsoft.TeamFoundation.Build.WebApi;
3  using Microsoft.TeamFoundation.Core.WebApi;
4  using Microsoft.TeamFoundation.SourceControl.WebApi;
5  using Microsoft.VisualStudio.Services.ReleaseManagement.WebApi;
6  using System.Collections.Generic;
7  using System.Threading.Tasks;
8
9  namespace ACMS.Managers
10 {
11     public interface IAzureDevOpsManager
12     {
13         Task ConfigureBoardAsync(ConfigureBoardRequestDto request);
14         Task<BuildDefinition> CreateBuildPipelineAsync(CreateBuildPipelineRequestDto request);
15         Task<GitRepository> CreateGitRepositoryAsync(CreateGitRepositoryRequestDto request);
16         Task<WebApiTeam> CreateTeamIfNotExistsAsync(CreateTeamRequestDto request);
17         Task<ReleaseDefinition> CreateReleasePipelineAsync(CreateReleasePipelineRequestDto request);
18         Task<IList<WebApiTeam>> GetAllTeamsAsync();
19         Task<IList<BuildDefinition>> GetAllTemplateBuildPipelinesAsync();
20         Task<IList<ReleaseDefinition>> GetAllTemplateReleasePipelinesAsync();
21     }
22 }
23
```

10 pav. Sistemos prototipui sukurta „Azure DevOps“ integracijos sąsaja

Sistemos funkcionalumo dynamiškumui užtikrinti, buvo remiamasi sudėjimo ir išleidimo kanalų šablonais. Visi šablonai buvo talpinami tam tikroje „Azure DevOps“ direktorijoje, kuri vėliau buvo nurodyta sistemos prototipo konfigūracijoje. Tokiu būdų nėra prisirišama prie specifinių detalių, kadangi viskas yra konfigūruojama. Šablonų nuskaitymui ir grąžinimui buvo sukurti dar du aplikacijos sąsajos prieigos taškai, po vieną skirtingų paskirčių šablonams (sudėjimo kanalams ir išleidimo kanalams).

Užtikrinti, jog sistemos prototipo galinė dalis veikia tinkamai, buvo naudojama „Postman“ programa, leidžianti modifikuoti ir siųsti HTTP užklausas. „Postman“ pagalba buvo ištestuotas visų sukurtų aplikacijos sąsajos priegos taškų veikimas bei grąžinami JSON tipo struktūros rezultatai.

Kadangi sudėjimo ir išleidimo kanalai automatiškai kuriami ir konfigūruojami naudojant šablonus, buvo rankiniu būdų sukurta keletas šablonų, kurie yra tinkamai paprasčiausiems diegti programinę įrangą naudojant paprasčiausius „Microsoft Azure“ debesies resursus.

3.2. Kontrolinių reikalavimų implementacija

Šiame poskyryje bus aprašomos sistemos prototipo kontrolinių reikalavimų, numatytų sudarant sistemos valdymo planą 2.1. poskyryje, implementacijos detalės ir kitos įžvalgos.

Kontrolinių reikalavimų tenkinimui, ankščiau buvo sudaryta duomenų bazės schema, į kurią atsižvelgiant buvo sukurtas „Entity Framework“ sprendimas programos galinėje dalyje, atsakingas už duomenų bazės sukūrimą bei operacijų atlikimą. Vadovaujantis „Code-first approach“ taktika,

programoje buvo sukurtos klasės atitinkančios visas duomenų esybes (žiūrėti 11 pav.), tada pasinaudojant „Entity Framework“ karkasu, buvo automatiškai sugeneruotas likęs programinis kodas, atsakingas už komunikaciją su duomenų baze. Ką tik sukurtoje duomenų bazėje buvo patalpinti testavimui sukurti duomenys, bei prieš tai numatytos bazinės naudotojo rolės.

```
using ACMS.Constants;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace ACMS.Database.Entities
{
    public class Form
    {
        public int FormID { get; set; }

        [Required]
        public string ProjectName { get; set; }

        [Required]
        public string ProjectManager { get; set; }

        [Required]
        public int EstMonthlyCost { get; set; }

        [Required]
        public string EstProductionDate { get; set; }

        public string Description { get; set; }

        [Required]
        public string TeamName { get; set; }

        [Required]
        public int BuildTemplateId { get; set; }

        [Required]
        public int ReleaseTemplateId { get; set; }

        [Required]
        public FormStatus Status { get; set; }

        public List<UserForm> Users { get; set; }

        public Form()
        {
            Status = FormStatus.Waiting;
        }
    }
}

```

11 pav. Užklausos formos esybės apibrėžimas programoje

Turint sukurtą duomenų bazę ir testinius duomenis buvo pereita prie standartizuoto užklausų proceso logikos bei supaprastinto RBAC modelio implementacijos. Pradedant nuo RBAC modelio buvo sukurtos funkcijos atsakingos už sistemos naudotojų ir jų rolių valdymą:

- Naudotojo sukūrimas.
- Naudotojo ištrynimas.
- Visų naudotojų grąžinimas.
- Konkretaus naudotojo grąžinimas.
- Naudotojo rolių grąžinimas.
- Naudotojo rolių priskyrimas.

Standartizuoto užklausų proceso implementavimo metu buvo sukurtos funkcijos skirtos užklausų formų valdymui:

- Formos sukūrimas.
- Formos ištrynimas.
- Visų formų grąžinimas.
- Konkrečios formos grąžinimas.
- Formos būsenos pakeitimas.

Kiekvienai išvardintai funkcijai buvo sukurtas aplikacijos sąsajos prieigos taškas, naudojamas iš programos priekinės dalies arba potencialiai kitų aplikacijų integracijai su šia sistema.

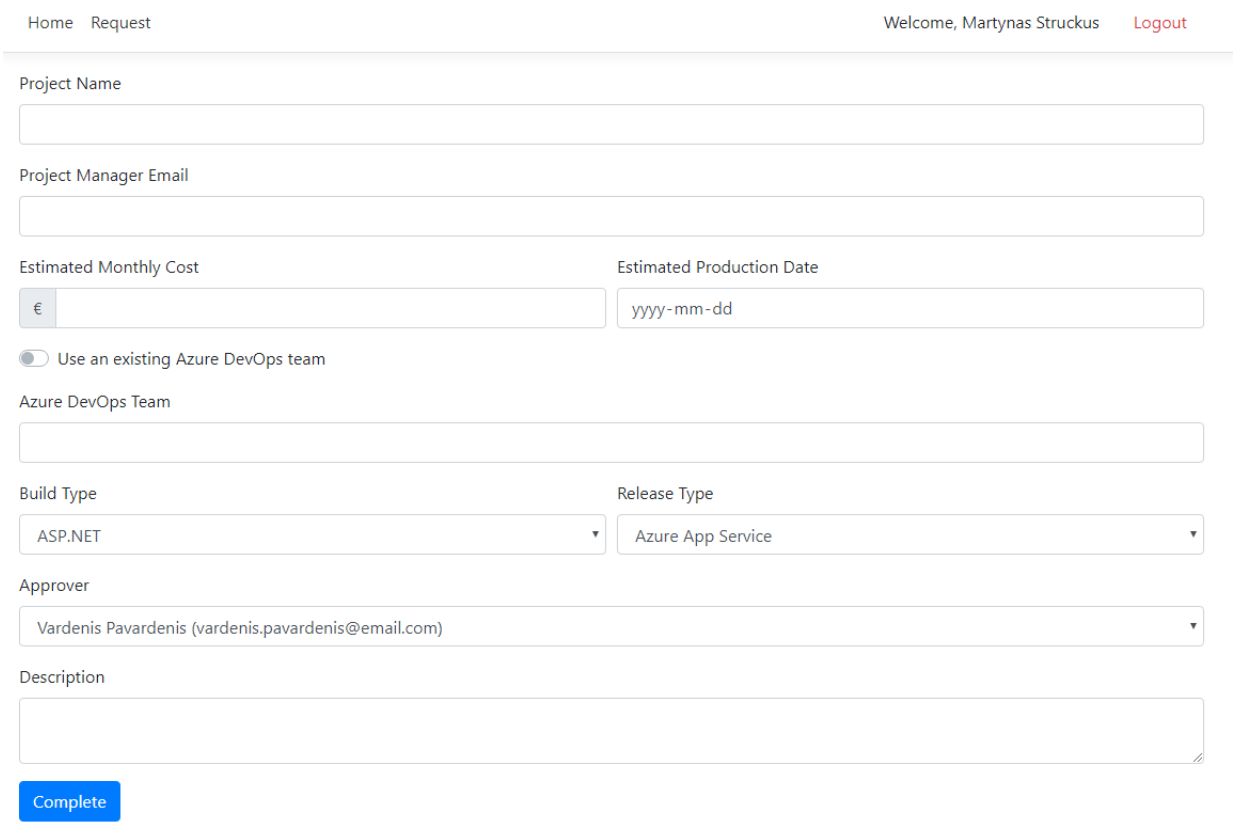
3.3. Naudotojo sąsajos kūrimas

Naudotojo sąsajos kūrimui buvo nuspręsta naudoti „React.js“ programinės įrangos karkasą, leidžianti generuoti dinamišką puslapio turinį. Naudotojo sąsaja reikalinga, užklausų formų kūrimui ir valdymui bei automatizavimo proceso iniciavimui ir progreso atvaizdavimui. Remiantis RBAC modeliu, papildomai buvo reikalinga įgyvendinti naudotojų rolių valdymo funkcionalumą. Išskiriant pagrindines naudotojo sąsajos funkcijas pagal tai buvo apibrėžti reikalingi puslapiai.

- Užklausos sukūrimo puslapis.
- Užklausų valdymo puslapis.
- Naudotojų rolių valdymo puslapis.
- Automatizavimo puslapis.

Visi išvardinti puslapiai turi bent dalį dinaminio turinio, kuris priklauso nuo naudotojo duomenų ar privilegijų. Reikiamiems duomenims gauti buvo siunčiamos užklausos į sistemos galinę dalį, kur jos yra apdorojamos bei grąžinami atitinkami rezultatai. Tam, kad sistema būtų kuo labiau efektyvesnė ir spartesnė, buvo papildomai realizuotas gautų duomenų saugojimas laikinojoje atmintyje (angl. *caching*).

Kadangi naudotojo sąsajos turinys skiriasi priklausomai nuo paties naudotojo ir sistemai būtina žinoti kas yra naudotojas, buvo papildomai sukurtas registracijos ir prisijungimo puslapis. Visgi tai yra tik sistemos prototipas, todėl buvo sukurtas paprastas autentifikacijos sprendimas - sistema patikrina pagal įvesta elektroninį paštą ar naudotojas egzistuoja duomenų bazėje ir grąžina atitinkamą rezultatą.



12 pav. Užklausos formos sukūrimo langas

Naudotojo sąsajos pradiniame puslapyje buvo realizuotas užklausų valdymo funkcionalumas, leidžiantis naudotojui matyti ir valdyti savo sukurtas užklausas bei jeigu naudotojas yra patvirtintojas - patvirtinti arba atmesti gautas užklausas. Atskirai buvo sukurtas užklausos sukūrimo puslapis (žiūrėti 12 pav.), kuriame yra pateikta pildoma forma. Taip pat buvo sukurtas naudotojų rolių valdymo puslapis, prieinamas ir rodomas tik sistemos administratoriams, kuriame pateikiama informacija apie visus sistemos naudotojus bei realizuotas rolių valdymo funkcionalumas. Galiausiai, buvo sukurtas automatizavimo puslapis, kuris yra pasiekiamas tik turint patvirtiną užklausos formą. Automatizavimo puslapyje yra papildomi įvesties laukai, neįtakojantys patvirtintojo sprendimo, mygtukas, leidžiantis inicijuoti automatizavimą bei pats automatizavimo progresas (žiūrėti 13 pav.). Sėkmingai baigus pilną automatizavimo procesą, užklausos forma pažymima kaip užbaigta. Automatizavimo puslapyje esančios įvestys taip pat yra dinamiškos ir priklauso nuo pasirinktų sudėjimo ar išleidimo kanalų.

Kuomet yra inicijuojamas automatizavimo procesas iš naudotojo sąsajos, pradedamas internetinių užklausų siuntimas į galinę programos dalį, komponentų priklausomybės grandinės (žiūrėti 7 pav.) tvarka. Nesėkmės atveju, atlika automatizavimo apimtis priklauso nuo to, kurioje grandinės vietoje įvyko klaida, tačiau kadangi automatizavimo procesas nebuvo įvykdytas iki galo, galima inicijuoti jį iš naujo.

13 pav. Sėkmingai atlikas automatinis konfigūracijos valdymo procesas

3.4. Automatizavimo apžvalga

Šiame poskyryje bus aptariami sukurto sistemos prototipo automatizavimo rezultatai bei kokybė.

Sistemos prototipas atlieka automatinį konfigūracijos valdymą, naudodamas „Azure DevOps“ sistemą, todėl automatizavimo galimybė priklauso nuo šios sistemos prieinamumo (angl. *availability*) bei funkcionalumo. Nepaisant to, dalis automatiškai konfigūruojamųjų kompopentų nesudėtingi, todėl problemų juos automatizuojant nėra. Tačiau, kita dalis komponentų (sudėjimo ir išleidimo kanalai) yra konfigūruojami sudėtingai, todėl šiuo atveju šių komponentų kokybė priklauso tik nuo naudojamų šablonų kokybės. Be to, sudėtingų architektūrų aplikacijos reikalauja unikalių sprendimų, todėl šablonų tobulinimas negali išspręsti šios problemos.

Taip pat kol kas nėra galimybės pridėti papildomų resursų kaip duomenų bazės, failų laikmenos ir kt., kurie nevisada reikalauja naujo projekto inicijavimo, todėl nėra automatiškai konfigūruojami. Tokiais atvejais yra būtinas DevOps specialisto įsikišimas tam, kad konfigūracijos valdymas būtų atliktas pilnai, kadangi tai yra jau už programos galimybių ribos.

Buvo aptikta dar viena problema automatizavimo procese. Įvykus klaidai automatizavimo grandinės viduryje arba gale, procesas būna nutrauktas ir iki galo nebaigtas, tačiau dalis komponentų jau yra sukurti ir sukonfigūruoti. Šioje vietoje atsiranda problema, jog užklausos forma dar nėra užbaigta, kadangi automatizavimo procesas neįvyko iki galo, bet ir nėra galimybės užklausos užbaigti, kadangi bandant iš naujo, automatizavimas pradedamas vėl nuo pradžių, tačiau pradiniai komponentai jau yra sukurti ir automatizavimas vėl vyksta nesėkmingai.

Nepaisant keletos minėtų problemų, automatizavimas 90% atvejų vyksta sklandžiai ir užtrunka mažiau negu 10 sekundžių, tačiau konfigūracijos atitikimas priklauso nuo aplikacijos architektūros sudėtingumo bei naudojamų įrankių galimybių.

3.5. Sistemos prototipo patobulinimų sąrašas

Kuomet buvo išbandytas sistemos prototipo funkcionalumas ir išnagrinėti rezultatai, buvo sudarytas aktualiausių ir būtiniausių sistemos pagerinimų sąrašas:

- Realizuoti papildomų resursų pridėjimo ir apdorojimo funkcionalumą.
- Sukurti galimybę, įvykus klaidai automatizavimo metu, pakartotinai pradėti automatizavimą nuo tos vietos, kurioje įvyko klaida.
- Realizuoti kitų CI/CD įrankių ir jų komponentų palaikymą.
- Implementuoti užklausos formos atšaukimo ir redagavimo funkcijas.
- Sukurti elektronių pranešimų siuntimo sprendimą, pranešančių apie užklasos būsenos pasikeitimus.
- Pridėti galimybę, kuri leistų atsisakyti kai kurių elementų automatizavimo/konfigūravimo.
- Realizuoti aktyvaus katalogo (angl. *active directory*) naudotojų autentifikaciją ir autorizaciją.

Sudarytame sąraše buvo išvardinti aptikti potencialūs sistemos pagerinimai nepriklausantys nuo organizacijos naudojančios tokią sistemą kompetencijų ar poreikių.

3.6. Sistemos prototipo ir kitų esančių sprendimų palyginimas

Šiame poskyryje bus lyginamas sukurtas sistemos protitipas su kitais, jau rinkoje esančiais konfigūracijos valdymo automatizavimo sprendimais.

3.6.1. Palyginimo eiga

Palyginimui buvo pasirinktas „Azure DevOps Starter“ įrankis (žiūrėti priedą nr. 1), kuris buvo išleistas 2020 metų kovo mėnesį [Cor20e], atliekantis automatinį konfigūravimą naudodamas „Azure DevOps“ sistemą, panašiu principu kaip ir sukurtas prototipas. Kitų panašaus pobūdžio įrankių rasta nebuvo.

Pagal bendrą sistemos protitipo ir „Azure DevOps Starter“ funkcionalumą, buvo sudarytas pagrindinių palyginimo kriterijų sąrašas:

1. Funkcionalumas.
2. Dinamiškumas.
3. Plėčiamumas.
4. Valdymas.

Taip pat buvo atsižvelgta ir į papildomą funkcionalumą, kurio neturi vienas iš minėtų įrankių.

„Azure DevOps Starter“ įrankis yra tinkamas automatizuoti visą CI/CD procesą [Cor20e]. Automatizuojamieji komponentai yra: pirminio kodo repozitorija, sudėjimo ir išleidimo kanalai, darbų lenta. Prieš pradedant automatizavimą būtina pasirinkti programinės įrangos karkasus bei kitus reikalingus resursus, tai parodo, kad įrankis veikia panašiu principu kaip ir sukurtas proto-

tipas. Pagrindiniai pranašumai yra tie, jog automatizavimo metu atliekamas versijavimo sistemos „git commit“ komandos paleidimas, kurios metu į naujai sukurtą pirminio kodo repozitoriją patalpinamas šabloninis naujos programos kodas, atitinkantis pasirinktą programinęs įrangos karkasą. To pasakoje, kartu yra ir inicijuojamas, automatiškai sukurtas, CI/CD procesas, todėl šiuo aspektu, „Azure DevOps Starter“ funkcionalumas yra šiek tiek pranašesnis.

Abu įrankiai naudoja tam tikrus šablonus skirtus automatizavimui bei reikalauja atitinkamų įvesčių, priklausomai nuo pasirinktų šablono. Tačiau „Azure DevOps Starter“ įrankis yra šiek tiek dinamiškesnis, kadangi yra galimybė pridėti ir papildomus resursus, tokius kaip duomenų bazė, ko neturi sukurtas prototipas. Be to, pasirenkamų resursų sąrašas yra atfiltruojamas pagal prieš tai pasirinktą programinės įrangos karkasą, taip yra apsisaugoma nuo negalimų ar nepalaikomų parinkčių. Todėl, iš dinamiškumo pusės, „Azure DevOps Starter“ taip yra pranašesnis įrankis.

Sukurtas sistemos prototipas turi aplikacijos sąsają (API), kas sudaro potencialias integracijos galimybes kitoms sistemoms bei užtikrina sistemos plėčiamumą. Tuo tarpu, „Azure DevOps Starter“ neturi atviros aplikacijos sąsajos, kurią būtų galima naudoti iš išorės, kadangi šiuo įrankiu galima naudotis tik iš „Microsoft Azure“ portalo [Cor20e]. Be to, atlikus bandymus buvo pastebėta, jog „Azure DevOps Starter“ suteikia fiksuota šablonų sąrašą naudotojo pasirinkimui, kurio negalima papildyti. Sukurtas prototipas, informaciją apie šablonus, atsisiunčia iš tam skirtos direktorijos, prie kurios prieigą turi „Azure DevOps“ sistemos naudotojų grupė. Tai leidžia, esant reikalui, plėsti sistemos prototipo galimybęs pridedant vis naujų šablonų, kuo negali pasigirti „Azure DevOps Starter“ įrankis.

Tiek sukurtas prototipas, tiek „Azure DevOps Starter“ yra valdomi panašiu principu. Iš pradžių yra supildoma reikalinga informacija, tik po to inicijuojamas automatizavimas. Tačiau „Azure DevOps Starter“ nepasižymi papildomomis saugos priemonėmis, bet kuris vartotojas turinti prieigą prie šio įrankio gali akimirksniu pradėti automatiškai kurti mokamus „Microsoft Azure“ resursus. Ši problema yra puikiai apdorojama, sukurto sistemos prototipo atžvilgiu, kadangi ten yra implementuotas patvirtinimo mechanizmas bei RBAC prieigos kontrolė, kas padeda kontroluoti ir deramai valdyti sistemą.

3.6.2. Palyginimo apibendrinimas

Palyginimo apibendrinimo išvados buvo sudarytos pagal gautus rezultatus palyginimo eigoje:

1. Funkcionalumas. „Azure DevOps Starter“ ir sukurto prototipo funkcionalumo apimtis yra panaši, tačiau „Azure DevOps Starter“ yra šiek tiek pranašesnis, kadangi automatizavimo metu yra patalpinamas naujos programos kodas į pirminio kodo repozitoriją, pagal pasirinktą programinės įrangos karkasą. Šio funkcionalumo sukurtas prototipas neturi.
2. Dinamiškumas. „Azure DevOps Starter“ yra dinamiškesnis įrankis, kadangi yra papildomai realizuota pasirinkimų suderinamumo logika bei suteikiama galimybė pasirinkti papildomus resursus.
3. Plėčiamumas. Sukurtas sistemos prototipas didesnį potencialą plėčiamumo galimybėms, kadangi buvo sukurta aplikacijos sąsaja, leidžianti integruotis kitoms sistemos bei plečiamas

šablonų pasirinkimas. Tuo nepasižymi „Azure DevOps Starter“ įrankis.

4. Valdymas. „Azure DevOps Starter“ neturi jokių papildomų kontrolės ar prieigos valdymo priemonių, tuo tarpu sukurto prototipo valdymas yra paremtas užklausų procesu bei RBAC prieigos kontrolės modelio realizacija.

Rezultatai

1. Sudarytas automatinis konfigūracijos valdymo problemos sprendimas, apimantis visas CI/CD proceso dalis. Išanalizuoti konfigūracijos valdymui skirti įrankiai bei atrinkti automatizuojamieji komponentai.
2. Sudarytas sistemos valdymo planas, paremtas užklausų formomis bei RBAC naudotojų prieigos kontrole.
3. Atlikta automatizuojamųjų konfigūracijos valdymo komponentų analizė. Analizės metu buvo sudaryta automatinio proceso darbo eiga pagal komponentų priklausomybes bei identifikuotos kiekvieno komponento reikalingos įvestys.
4. Sukurti konfigūracijos valdymo automatizavimui skirti sudėjimo ir išleidimo kanalų šablonai. Šablonai buvo panaudoti klonavimui atliekant automatizavimą.
5. Realizuotas sistemos prototipas, atliekantis automatizavimą paremtą „Azure DevOps“ sistemos integracija. Sukurta naudotojo sąsaja skirta sistemos valdymui. Įgyvendinti valdymo plane numatyti sistemos kontroliniai reikalavimai.
6. Atlikti sistemos prototipo naudojimo bandymai. Bandymų metu buvo atliekamas konfigūracijos automatizavimas, taikant skirtingus architektūrinius reikalavimus. Išanalizavus bandymų rezultatus buvo sudarytas sistemos pagerinimų sąrašas.

Išvados

1. Konfigūracijos valdymo automatizavimas vykdomas greitai, 90 procentų atvejų neviršyja 10 sekundžių. Tai leidžia suprasti, jog gerai išvystyta tokio pobūdžio sistema gali būti labai naudinga programinę įrangą kuriančioms įmonėms.
2. Automatizavimo kokybė priklauso nuo aplikacijos architektūros sudėtingumo bei šablonų atitikimo. Kuo šablonas yra universalesnis ir lankstesnis, tuo kokybiškesnis automatizavimas, kadangi tai suteikia sistemai dinamiškumo. Tačiau, šablonų kokybė ne visada gali padėti, vienas iš „Microsoft Azure“ resursų - „Azure Data Factory“ yra pakankamai sudėtingas, kad sistema gebėtų pilnai jį sukonfigūruoti, todėl šiuo atveju yra būtinas DevOps specialisto įsitraukimas. Tai parodo, jog pilnai pasitikėti tik sistemos atliekamu darbu, kol kas negalima.
3. Automatizavimui reikalinga didelė kitų įrankių ir integracijų įvairovė: versijavimo sistemos sprendimas, CI/CD komponentų palaikymas, projektų valdymo įrankiai, todėl šis procesas gali būti nestabilus bei netinkamas reguliariems atnaujinimams.
4. Greitas ir nevaldomas konfigūracijos valdymo elementų kūrimas potencialiai gali sugeneruoti daug papildomų ir nereikalingų išlaidų, todėl būtina sistemą tinkamai kontroliuoti.
5. Lyginant su esamu automatiniu konfigūracijos valdymo įrankiu - „Azure DevOps Starter“, sukurtas protitipas nesuteikia tokios didelės funkcionalumo apimties, automatizavimo prasme, tačiau yra pranašesnis savo valdymo bei plėčiamumo galimybėmis.

Šaltiniai

[Ben18] Brian Benz. Azure devops and jenkins in perfect harmony, 2018. URL: https://medium.com/@bbenz/azure-devops-and-jenkins-in-perfect-harmony-8c92ff980723 (tikrinta 2020-05-04).

[Blo17] Jonathan Block. Integrating github with jenkins for continuous integration and deployment, 2017. URL: https://medium.com/@DoorDash/integrating-github-with-jenkins-for-continuous-integration-and-deployment-7cae2c2161cb (tikrinta 2020-05-15).

[Bru17] Ben Brumm. How to handle a many-to-many relationship in database design, 2017. URL: https://dzone.com/articles/how-to-handle-a-many-to-many-relationship-in-datab (tikrinta 2020-05-04).

[Car17] Samuel Carter. Rbac vs abac access control models - iam explained, 2017. URL: https://blog.identityautomation.com/rbac-vs-abac-access-control-models-iam-explained (tikrinta 2020-05-16).

[Cha20] Jay Chapel. Aws vs azure vs google cloud market share 2020: what the latest data shows, 2020. URL: https://medium.com/@jaychapel/aws-vs-azure-vs-google-cloud-market-share-2020-what-the-latest-data-shows-9afd4accf8d7 (tikrinta 2020-05-14).

[Cor16a] Microsoft Corporation. Azure devops services rest api reference, 2016. URL: https://docs.microsoft.com/en-us/rest/api/azure/devops/?view=azure-devops-rest-6.0/ (tikrinta 2020-05-15).

[Cor16b] Microsoft Corporation. Azure devops services rest api reference, 2016. URL: https://docs.microsoft.com/en-us/rest/api/azure/devops/?view=azure-devops-rest-5.1 (tikrinta 2020-05-16).

[Cor19a] Microsoft Corporation. Start using your kanban board, 2019. URL: https://docs.microsoft.com/en-us/azure/devops/boards/boards/kanban-quickstart?view=azure-devops (tikrinta 2020-05-16).

[Cor19b] Microsoft Corporation. What is azure pipelines?, 2019. URL: https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops (tikrinta 2020-05-16).

[Cor20a] Microsoft Corporation. Authenticate access with personal access tokens, 2020. URL: https://docs.microsoft.com/en-us/azure/devops/organizations/accounts/use-personal-access-tokens-to-authenticate?view=azure-devops%5C&tabs=preview-page (tikrinta 2020-05-16).

[Cor20b] Microsoft Corporation. Azure devops, 2020. URL: https://azure.microsoft.com/en-us/services/devops/ (tikrinta 2020-05-15).

[Cor20c] Microsoft Corporation. Azure for students, 2020. URL: https://azure.microsoft.com/en-us/free/students/ (tikrinta 2020-04-19).

[Cor20d] Microsoft Corporation. Devops solutions, 2020. URL: https://azure.microsoft.com/en-in/solutions/devops/ (tikrinta 2020-05-15).

[Cor20e] Microsoft Corporation. Overview of azure devops starter, 2020. URL: https://docs.microsoft.com/en-us/azure/devops-project/overview%5C#devops-starter-and-azure-devops-integration (tikrinta 2019-05-27).

[DNS19] DNSstuff. Rbac vs. abac: what's the difference?, 2019. URL: https://www.dnsstuff.com/rbac-vs-abac-access-control (tikrinta 2020-05-16).

[Fag20] Julia Fagelman. 10 simple steps to agile project management with scrum, 2020. URL: https://monday.com/blog/scrum/agile-project-management-scrum/ (tikrinta 2020-05-14).

[Glo17] Globalluxsoft. The 7 best project management tools for software development, 2017. URL: https://medium.com/globalluxsoft/the-7-best-project-management-tools-for-software-development-6ed9c1012ec5 (tikrinta 2020-05-14).

[Han18] Brianna Hansen. 5 best practices for managing incoming work requests, 2018. URL: https://www.wrike.com/blog/5-best-practices-managing-incoming-work-requests/ (tikrinta 2020-05-16).

[Inc19] Gartner Inc. Gartner forecasts worldwide public cloud revenue to grow 17% in 2020, 2019. URL: https://www.gartner.com/en/newsroom/press-releases/2019-11-13-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-17-percent-in-2020 (tikrinta 2020-05-14).

[Inc20a] JotForm Inc. Request forms, 2020. URL: https://www.jotform.com/form-templates/category/request-form (tikrinta 2020-05-16).

[Inc20b] Wrike Inc. Why should i use request forms in project management software?, 2020. URL: https://www.wrike.com/project-management-guide/faq/why-should-i-use-request-forms-in-project-management-software/ (tikrinta 2020-05-16).

[Inc20c] Wrike Inc. Wrike customers, 2020. URL: https://www.wrike.com/customers/ (tikrinta 2020-05-16).

[Jen20] Jenkins. Jenkins user documentation, 2020. URL: https://www.jenkins.io/doc/ (tikrinta 2020-05-15).

[Kum19] Mukesh Kumar. What is entity framework and how entity framework core is different, 2019. URL: https://www.c-sharpcorner.com/article/what-is-entity-framework-how-entity-framework-core-is-different/ (tikrinta 2020-05-16).

[LLC20a] Google LLC. Google cloud platform education, 2020. URL: https://edu.google.com/programs/students/benefits/ (tikrinta 2020-04-19).

[LLC20b] Katalon LLC. Ci/cd pipeline: what, why & how to build the best one, 2020. URL: https://www.katalon.com/resources-center/blog/ci-cd-pipeline/ (tikrinta 2020-05-14).

[Mad20] Krzysztof Madej. Build templates on azure devops, 2020. URL: http://thecodemanual.pl/2020/04/02/build-templates-on-azure-devops.html (tikrinta 2020-05-16).

[McL20] Mark McLaren. Jenkins vs azure devops pipelines, 2020. URL: https://devops.stackexchange.com/questions/5157/jenkins-vs-azure-devops-pipelines (tikrinta 2020-05-15).

[Min16] Chris Minnick. The real benefits of the virtual dom in react.js, 2016. URL: https://www.accelebrate.com/blog/the-real-benefits-of-the-virtual-dom-in-react-js/ (tikrinta 2020-05-16).

[Mou19] Thomaz Moura. Tfs version control is dead, 2019. URL: https://medium.com/@thomaz.moura/tfs-version-control-is-dead-6d475e247389 (tikrinta 2020-05-04).

[Mul17] Ben Mulholland. How to create a project request form (and why your company needs one), 2017. URL: https://www.business2community.com/strategy/create-project-request-form-company-needs-one-01873522 (tikrinta 2020-05-16).

[Nag19] Chris Nagele. An introduction to version control, 2019. URL: http://guides.beanstalkapp.com/version-control/intro-to-version-control.html (tikrinta 2020-05-15).

[Ngu17] Kim Nguyen. Relational database schema design overview, 2017. URL: https://medium.com/@kimtnguyen/relational-database-schema-design-overview-70e447ff66f9 (tikrinta 2020-05-16).

[Sel19] Ihor Seleznov. 3 ways to run automated tests on azure devops, 2019. URL: https://blog.techfabric.io/3-ways-to-run-automated-tests-on-azure-devops/ (tikrinta 2020-05-16).

[Sen17] Pradip Sengupta. 10 reasons asp.net application is reliable for good websites, 2017. URL: https://www.ipstechnologyservices.com/blog/10-reasons-asp-net-application-is-reliable-for-good-websites/ (tikrinta 2020-05-16).

[Ser20] Amazon Web Services. Aws educate, 2020. URL: https://aws.amazon.com/education/awseducate/students/ (tikrinta 2020-04-19).

[Sny15] Russell Snyder. Version control systems: keep your code in order!, 2015. URL: https://webinerds.com/version-control-systems-keep-your-code-in-order/ (tikrinta 2020-05-15).

[War18] Chris Ward. What is continuous integration?, 2018. URL: https://www.exoscale.com/syslog/what-is-continuous-integration/ (tikrinta 2020-05-14).

Santrumpos

CI - Continious Integration
CD - Continious Delivery
RBAC - Role-Based Access Control
ABAC - Attribute-Based Access Control
DOM - Document Object Model
TFVC - Team Foundation Version Control
PAT - Personal Access Token

Priedas nr. 1

„Azure DevOps Starter“ įrankis