VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

API kūrimas – geriausios praktikos

Development of API – Best Practices

Bakalauro baigiamasis darbas

Atliko:	Patricija Buškevičiūtė	(parašas)
Darbo vadovas:	lekt. Tomas Plankis	(parašas)
Darbo recenzentas:	prof. Aldas Glemža	(parašas)

Vilnius – 2017

SANTRAUKA

Šis rašto darbas yra skirtas nustatyti geriausias API kūrimo praktikas. Darbe pristatomi gero API bruožai, jų formavimo metodai ir formavimui naudojami įrankiai. Pateikiama įrankių palyginamoji analizė. Toliau identifikuojamos API kūrimo strategijos, jos lyginamos tarpusavyje pagal kiekvienos galimybes sukurti gerą API produktą.

Darbe pateikiamos API publikacijai naudojamos technologijos, jos siejamos su identifikuotomis strategijoms. Atliekama palyginamoji technologijų analizė pagal gerų API bruožų formavimą, bei suderinamumą su formavimui naudojamais įrankiais.

Atliktų palyginimų rezultatai siejami tarpusavyje ieškant jų dermės ir rekomenduotinos naudojimui kombinacijos.

Raktiniai žodžiai: API, gerosios praktikos, API kūrimas, API produktas, strategija, technologija, įrankiai.

SUMMARY

This report is dedicated to distinguish methods of API creation. In this paper good characteristics of API, their formation methods and tools used for formation are introduced. Comparative analysis of used tools is presented. Further, API creation strategies are identified, they are being compared between each other according to each ones possibilities to create a good API product.

This research presents technologies that are used for externalization, they are linked with identified strategies.

Going further comparative technology analysis according to good API characteristics formation and compatibility with tools for formation is carried out. The findings of comparative analysis are connected in search of harmony and recommended combination for use.

Key words: API, best practices, development of API, API product, strategy, technology, tools.

TURINYS

ĮVADAS ... 5
Temos aktualumas ... 5
Darbo tikslas ... 6
Uždaviniai tikslams pasiekti ... 6
Naudojami metodai ... 6
1. API APŽVALGA ... 7
1.1. Apibrėžimas ... 7
1.2. Privalumai ir trūkumai ... 7
1.3. API kaip produktas ... 9
1.4. Pirkti ar kurti ... 10
2. GERO API BRUOŽAI ... 11
2.1. Dokumentacija ... 11
2.2. Stabilumas ... 11
2.3. Lankstumas ... 12
2.4. Saugumas ... 13
2.5. Panaudojamumas ... 13
3. API KŪRIMO STRATEGIJOS ... 15
3.1. Apžvalga ... 15
3.2. Palyginimas ... 16
4. GERO API KŪRIMO METODAI ... 19
4.1. Dokumentacija ... 19
4.2. Stabilumas ... 21
4.3. Lankstumas ... 25
4.4. Saugumas ... 26
4.5. Panaudojamumas ... 28
5. API PUBLIKACIJA ... 29
5.1. Apžvalga ... 29
5.2. „REST“ ir „SOAP“ ... 29
5.2.1. Apžvalga ... 29
5.2.2. Palyginimas ... 30
REZULTATAI IR IŠVADOS ... 32
ŠALTINIAI ... 35
SĄVOKŲ APIBRĖŽIMAI IR SANTRUMPOS ... 39

ĮVADAS

Temos aktualumas

Informacinių technologijų sektorius pasaulyje sparčiai vystosi, o naudotojai reikalauja vis didesnio mobilumo. Tuo tarpu API leidžia kurti lanksčią programinę įrangą (toliau PĮ), pritaikytą skirtingiems įrenginiams. Taigi, tapo populiaru API servisus naudoti kaip alternatyvą įmonės siūlomo funkcionalumo publikavimui. Taip publikuotas funkcionalumas yra lengvai ir paprastai plečiamas, leidžia integruotis su trečiųjų šalių platformomis. Šios priežastys kelia API vertę rinkoje, o įmonės ima žiūrėti į API nebe kaip į servisus, o produktą [TW17].

Naujų produktų kūrimas ir paleidimas rinkoje yra nemažas iššūkis. Rezultatai turi atitikti klientų reikalavimus, o tuo pačiu – būti patrauklūs ir sėkmingai konkuruoti su varžovų siūlomomis paslaugomis. Produktų kūrimui ir palaikymui yra sugalvota daug skirtingų strategijų ir praktikų. Jos nuolat kinta, nes bėgant laikui tobulėja technologijos ir senas strategijas bei tendencijas keičia naujos.

Siekiant sukurti gerą produktą mažiausiomis laiko ir pinigų sąnaudomis svarbu pasirinkti esamai situacijai optimalius metodus. Tą padaryti įmanoma tik pakankamai laiko investavus į procesų ir įrankių tyrimą, skirtingų strategijų palyginimą ir tokiu būdu lengviau numatyti galimus rezultatus, kokių nuokrypių tikėtis arba paprasčiausiai identifikuoti, kas yra madinga.

Eksperimentuojant, vadovaujantis skirtingais strateginiais sprendimais kyla rizika patirti nuostolius – vartotojams ar užsakovams išreiškus nepasitenkinimą gali tekti PĮ kurti iš naujo. Siekdami greitesnio problemų sprendimo PĮ kūrėjai linkę taisyti turimą produktą „lipdydami ant viršaus“. Toks pasirinkimas kenkia darbo procesui – jis tampa nenuoseklus, sutrinka problemų atsekamumas, nukenčia PĮ plečiamumas ir jos palaikymas tampa sudėtingesniu. Dėl to ypač svarbu jau produkto kūrimo pradžioje identifikuoti kokios yra geriausios kūrimo praktikos ir apibrėžti, kurios iš jų bus taikomos produkto kūrimo metu.

Pastaruoju metu vis dažniau girdimas strategijos „Pirmiausia API“ (angl. „API First“) pavadinimas. Tai gan „šviežias“ API produktų kūrimo būdas, sulaukęs daug teigiamų atgarsių ir po truputį užkariaujantis PĮ kūrėjų širdis. Kita vertus, ši strategija dar nėra plačiai taikoma, vis dar palaikomas tradicinis, į paslaugas orientuotas, architektūros kūrimo būdas (angl. „SOA – service oriented architecture“), pagrįstas strategija „Pirmiausia UI“.

Darbo tikslas

Nustatyti, kurie iš taikomų API kūrimo strategijų, metodų, įrankių bei publikacijai naudojamų technologijų gali būti laikomi geriausiomis API kūrimo praktikomis.

Uždaviniai tikslams pasiekti

1. Identifikuoti gero API bruožus.
2. Identifikuoti API kūrimo strategijas.
3. Atlikti API kūrimo strategijų palyginamąją analizę.
4. Identifikuoti metodus, kuriais formuojamas geras API.
5. Atlikti gero API formavimui naudojamų priemonių palyginamąją analizę.
6. Identifikuoti technologijas, naudojamas API publikacijai.
7. Atlikti naudojamų technologijų palyginamąją analizę.
8. Identifikuoti rekomenduotiną strategijų, įrankių ir technologijų kombinaciją.

Naudojami metodai

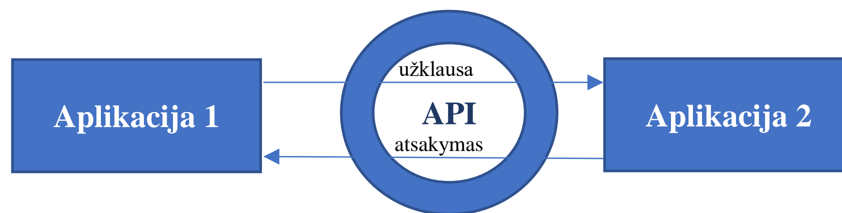
1. Teoriniai tyrimų metodai:
   - analizės metodas;
   - lyginamosios analizės metodas;
   - modeliavimo metodas;
   - apibendrinimo metodas.
2. Atvejo studija [MM13].

1. API APŽVALGA

1.1. Apibrėžimas

Aplikacijų programavimo sąsaja (angl. „Application Programming Interface, API“) veikia kaip rinkinys aiškiai apibrėžtų metodų, skirtų komunikacijai tarp PĮ ar jos komponentų. PĮ neturi jausmų ir emocijų, todėl šiuo atveju nėra aktualu sukurti intuityvią ar patrauklią sąsają – orientuojamasi į tai, kad būtų paprasta pasiekti reikiamą informaciją ar naudotis specifiniais PĮ funkcionalumais.

Paprasčiau API sąvoką paaiškinti gretinant ją su vartotojo sąsajos (angl. „User Interface, UI“) terminu. Taip kaip UI yra orientuotas į žmones, kad šiems būtų kuo paprasčiau naudotis programomis, taip API yra skirtas PĮ ar jos komponentams bendrauti tarpusavyje. Tarkime aplikacija 1 nori naudotis kitos aplikacijos 2 teikiamu API. Tuomet aplikacija 1 siunčia savo užklausą aplikacijai 2, o ši savo ruožtu grąžina reikiamą informaciją užklausos rezultate, atsakyme (1 pav.).



1 pav. Supaprastinta API veikimo schema

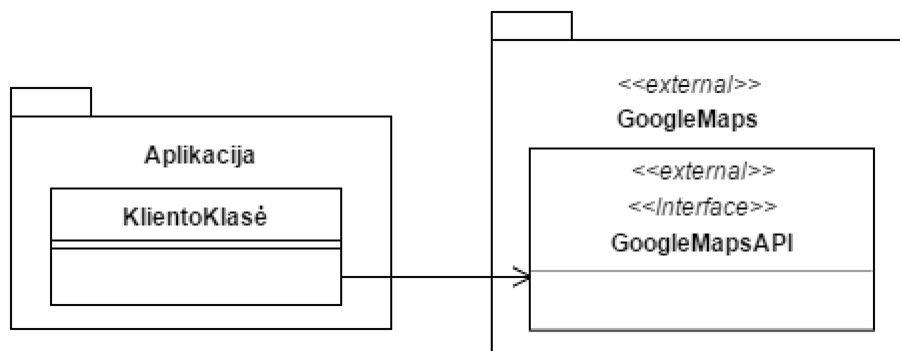
API veikia kaip aukštesnio lygio abstrakcija leidžianti paslėpti detalės apie tai, kaip funkcionalumas buvo įgyvendintas. Sąsaja yra formuojama pagal specialų projektavimo modelį paviešinant tik objektus ir funkcijas, kurių gali reikėti programuotojams. Šie savo ruožtu pasitelkdami papildomas technologijas, kuria naujas aplikacijas [Ber15].

1.2. Privalumai ir trūkumai

Paprastai įvairūs programų elementai yra įmontuojami (angl. „hardwired“) į PĮ. Toks funkcionalumo įkodavimas (angl. „hard coding“) mažina PĮ lankstumą, plečiamumą ir perpanaudojamumą – esant glaudiems ryšiams tarp kelių PĮ elementų ir vienam jų taikant pakeitimus, nenoromis paveikiami ir likusieji.

Tuo tarpu API palengvina programuotojų darbą – suteikia PĮ kūrimo procesui lankstumo bei sutaupo laiko. Naudojantis API produktu programuotojams nereikia gilintis į žemesnio lygio abstrakcijas – įgyvendinimo detales.

Programuotojai gali lengvai ir greitai kurti naujus produktus kombinuodami API produktus tarpusavyje. Tai reiškia, kad esant poreikiui galima pasinaudoti rinkoje jau egzistuojančiu API produktu ir išvengti pakartotino darbo paprasčiausiai kuriant integracijas su aktualia trečiųjų šalių PĮ – duomenų bazėmis ar programų kūrimo įrankiais (angl. „Software Development Kit, SDK“). Pavyzdžiui, socialinis tinklalapis „Facebook“ išleido SDK leidžiančią programuotojams kurti žaidimus skirtingoms platformoms [Seg17]. Tuo tarpu „Google Maps“ API žymiai pagreitina aplikacijų kūrimo procesą programuotojams – naudojant papildomas technologijas, tokias kaip „JavaScript“ ar „Flash“, galima lengvai ir greitai integruoti žemėlapių funkcionalumą į pasirinktą internetinį puslapį ar aplikaciją (2 pav.). Šis API pilnai suderinamas tiek su mobiliaisiais įrenginiais, tiek darbalaukio naršyklėmis [Bea17]. Pasak „Webopedia“ puslapio, tai vienas populiariausių naudojamų API.



2 pav. Klasių diagramos fragmentas rodantis prilausomybinį ryšį tarp aplikacijos ir jos naudojamo API serviso

API ne tik pagreitina naujos PĮ kūrimą, bet ir padidina programuotojų galimybes. Sujungus kelis skirtingus servisus galima sukurti visiškai unikalią ir paklausią rinkoje programą. Be to, teikti API servisus trečiosioms šalims apsimoka – gaunamos papildomos pajamos ir garsinamas prekinis ženklas. Pavyzdžiui, 2012 m. duomenimis net 60% „eBay“ pelno buvo sukaupta teikiant API servisus kitų įmonių PĮ kūrėjams [Lak14].

Šiame technologijų amžiuje įmonės siekia optimizuoti savo procesus. API suteikia galimybę rankinį žmonių darbą pakeisti mechaniniu, kuomet vietoje rankinio darbo (mygtukų spaudinėjimo) veiksmai yra atliekami automatiškai – vykdomas už juos atsakingas programinis kodas. Tai ypač

praverčia, kai reikia atlikti ekvivalenčius veiksmus su daugeliu skirtingų objektų. Automatizavus šiuos procesus sumažėja žmogiškųjų resursų poreikis, juos galima paskirstyti veiksmingiau.

1.3. API kaip produktas

Kad sėkmingai konkuruotų rinkoje organizacijos nuolatos turi kurti naujus produktus. Tokiu būdu yra patenkinama vis daugiau klientų poreikių, plečiasi jų gretos, o tuo pačiu didėja gaunamas pelnas.

Vis daugiau organizacijų į kasdienę savo veiklą įtraukia naršykles ir mobiliasias aplikacijas. Tuo tarpu API leidžia šiose skirtingose platformose sėkmingai vykdyti duomenų mainus – taip yra tobulinami egzistuojantys produktai, sistemos, operacijos, bei kuriamos galimybės versle. Dėl to pradėta API interpretuoti kaip produktą tiek išoriniams naudotojams, tiek esantiems įmonės viduje – kolegoms programuotojams bei vidaus sistemoms. Kita vertus, API palikus antrame plane ir susikoncentravus ties aplikacijos įgyvendinimu – vidiniu (angl. „back-end“) ir išoriniu (angl. „front-end“) programavimu – prarandama galimybė publikuoti turimą funkcionalumą ir padidinti gaunamą pelną.

Tiekėjų siekiamybė yra geri naudotojų atsiliepimai apie jų teikiamą produktą bei gaunamas grįžtamas pelnas. Keletas labiausiai dominuojančių [Kas16] tą užtikrinančių produktų politikų yra:

1. Siūlyti produktus žemiausia kaina. Ši politika reiškia, kad už pavienius produktus bus uždirbama mažiau. Tačiau tikėtina, kad bus įvykdomas didesnis pardavimų kiekis, o tai lems didesnį bendrąjį pelną.
2. Siūlyti aukščiausios kokybės produktus. Klientai vertina kokybę – ji kuria pasitikėjimą tiekėjais ir gaunamomis paslaugomis, atsiranda nuolatiniai naudotojai.
3. Siūlyti saugius produktus. Saugumo kriterijus yra vienas esminių – nesaugūs produktai griauna klientų pasitikėjimą – kyla rizika praradus duomenų privatumą patirti nuostolius.

Komandos, kuriančios API, turėtų aktyviai analizuoti klientų poreikius ir remiantis surinktais duomenimis formuoti kuriamų produktų savybes. API kaip produktai turėtų būti aktyviai palaikomi ir lengvai panaudojami, klientų problemos sprendžiamos, o teikiamos paslaugos nuolatos tobulinamos. Tinkamu metu atlikta rinkos metodų, naudojamų technologijų ir įrankių analizė, geriausių praktikų identifikavimas padeda atrasti tinkamiausią verslui ar įmonei API kūrimo strategiją. Jos laikantis bus sukurti aukštos kokybės, lengvo pritaikymo API produktai atitinkantys anksčiau minėtas gerų produktų politikas.

1.4. Pirkti ar kurti

Yra du keliai, kuomet kalbama apie API produktus. Pirmas kelias – nuo pagrindų kurti juos patiems, o antras – naudoti trečiųjų šalių siūlomus API servisus. Abu variantai turi savų privalumų ir trūkumų, kurie priklauso nuo pasirinkto API strategijos ir atsispindi tiek per verslo, tiek kasdienio PĮ kūrėjų darbo prizmę laiko, piniginiu ir saugumo aspektais.

Įmonės, siūlančios API servisus, orientuojasi į gerokai platesnius rinkos segmentus. Jų paslaugos tampa aktualios ne tik pagrindinio funkcionalumo naudotojams, tačiau ir programuotojams, kurie neturi pakankamai laiko ir pinigų kurti panašių servisų nuo pagrindų arba paprasčiausiai siekia sukurti inovatyvią PĮ kombinuodami keletą skirtingų funkcijų. Be to, sklandžios integracijos su trečiųjų šalių PĮ prisideda prie įmonės prekinio ženklo populiarinimo, padeda konkuruoti rinkoje.

Kita vertus API kūrimas yra brangus ir ilgas procesas, tad reiktų gerai apsvarstyti ar tokia investicija atsipirks. Visgi vien tai, kad servisai bus sukurti nereiškia, kad programuotojai būtinai jais naudosis – tai priklauso nuo API panaudojamumo, stabilumo, patogumo ir pan. Kad API atitiktų visus šiuos kriterijus būtina nuolatinė servisų priežiūra, pagalbos klientams teikimas, išsami dokumentacija. Be to, publikavus įmonės funkcionalumą API servisais atveriamas papildomas langas kibernetinei atakai atsižvelgiant į tai, kad servisų naudotojai gali jungtis per nepatikimas aplinkas. Todėl svarbu papildomai investuoti į servisų saugumo užtikrinimą.

Turint ribota biudžetą ir programuotojų resursus labiau apsimoka naudotis jau sukurtais API servisais – tikėtina, kad rinkoje egzistuojantys servisai yra geresni, nei įmonė galėtų pati susikurti. Pasirinkus trečiųjų šalių API lieka daugiau laiko koncentruotis į pamatinių paslaugų tobulinimą ir įmonės augimą, be to išvengiama servisų palaikymo išlaidų.

Kita vertus, pasirinkus trečiųjų šalių API sukuriamas stiprus priklausomybinis ryšys su išoriniais serveriais, todėl svarbu įsitikinti, kad pasirinkti tiekėjai yra patikimi, o jų PĮ stabili. Taip pat sudėtingiau išvengti saugumo spragų, nes paslaugų įgyvendinimo detalės lieka nežinomos [Kuo16].

2. GERO API BRUOŽAI

2.1. Dokumentacija

Dokumentacija yra tarytum API viršelis, kuriantis produktui tam tikrą įvaizdį – su ja susiduriama pirmiausia, dar prieš pradedant naudotis teikiama paslauga. Jeigu ji nepilna, pateikiama be aiškios struktūros, tai suponuoja, kad ir API buvo sukurtas nepakankamai kokybiškai. Tuo tarpu tvarkinga ir aiški dokumentacija padeda instruktuoti klientus, visapusiškai pristatyti ir parduoti produktą.

Taigi, aiški ir išsami dokumentacija rodo produkto kokybę, kelią jo vertę.

Vertinimo kriterijai: aiškumas, pilnumas.

2.2. Stabilumas

Kaip jau buvo minėta – API produktai skirti naudoti ne rankiniu būdu, tačiau mechaniškai, kuomet kuriamos įvairios integracijos tarp skirtingų PĮ. Servisai kombinuojami tarpusavyje papildomo programinio kodo pagalba, o procesai automatizuojami. Vadinasi, sukūrus API produktus ir juos publikavus išoriniams naudotojams, jie lieka visam laikui vieši. Dėl to yra labai svarbu, kad pirminė servisų publikacija būtų vykdoma tik tada, kai yra atlikti visi kokybės užtikrinimo darbai ir yra įsitikinta, kad servisai veikia be trukdžių, pvz.: platformoje vykdant lygiagrečius procesus nekyla konfliktų. Tokiu būdu bus išvengta papildomų produkto versijų keitimo.

Produkto tobulinimo darbai yra neišvengiami, kadangi tobulėja technologijos, keičiasi paslaugų pasiūla. Pavyzdžiui, socialinio tinklo „Facebook“ API kūrėjai nuolatos perrašo savo API servisus ir išjungia senus. Tai nėra taktiškas požiūris į savo API naudotojus, tačiau „Facebook“ populiarumas neblėsta – ne dėl savo PĮ, bet dėl didelės naudotojų ir gerbėjų gausos. Tačiau naudotojai yra suinteresuoti gauti stabilias ir patikimas paslaugas, todėl toks dažnas versijų keitimas yra rizikingas mažesniems prekiniams ženklams. Naudotojų grupė tikėtina yra mažesnė, o vykdant dažnus ir drąstiškus pakeitimus rizikuojama netekti jų pasitikėjimo, dėl to kris produkto paklausa, mažės pelnas. Taigi, svarbu kontroliuoti produkto versijavimą ir užtikrinti jo pastovumą.

Automatizacija reikalauja stabilumo ir nuoseklumo. Jei siūlomas API ir jo ypatybės keičiasi dažnai, nėra laikomasi gairių ir standartų – šių bruožų netenkama. Programinis kodas ir naudojami

įrankiai, priešingai nei realūs sistemų naudotojai, neturi intuityvaus suvokimo, todėl visos detalės turi būti aiškios, apibrėžtos ir tinkamai apdorotos.

Konfliktai, kylantys vykdant lygiagrečius procesus, dažniausiai atsiranda monolitiškuose produktuose, esant priklausomybiniams ryšiams (angl. „dependencies“) tarp servisų. Tokia praktika yra vengtina – kiekvienas API produktas turėtų būti orientuotas į specifinį funkcionalumą ir jį valdyti be trukdžių. Jei kuriamas monolitinis produktas, atsiradus problemoms gali sutrikti visų kanalų ir funkcinių objektų veikimas. Tokius sutrikimus sunkiau taisyti, patiriami dideli nuostoliai, neturima alternatyvų, kurias būtų galima pasiūlyti naudotojams, pavyzdžiui – sutrikus API veikimui kurį laiką naudotis sistemos UI.

Kuo daugiau trikių galima nuspėtų, tuo anksčiau galima pradėti juos spręsti arba apskritai planuoti, kaip jų išvengti. Tokiu atveju, kai trikiai atsiranda, svarbu turėti kuo palankesnes sąlygas jiems spręsti. Tai suteikia API robustiškumo, atsparumo sutrikimams ir stabilumo.

Stabilumas užtikrinamas kontroliuojant serverių apkrovą.

Vertinimo kriterijai: nuoseklumas, pastovumas, monolitiškumas, atsparumas – trikių vengimas ir šalinimas.

2.3. Lankstumas

Gerieji API bruožai yra formuojami skirtingomis technologijomis ir metodais. Tuo tarpu lankstumas apima bruožų visumą. Jis apibrėžiamas naudotojo pasirinkimo laisve technologijų ir metodų atžvilgiu.

Neįmanoma nuspėti visų galimų integracijų variantų, t.y. kaip naudotojas nuspręs naudoti API produktą. Be to, nėra garantijos, kad kiekvieno kliento platforma yra stabili, palaiko tuos pačius autorizacijos standartus ar perduodamų duomenų formatus. Potencialūs API naudotojai – trečiųjų šalių PĮ kūrėjai – dažniausiai jau turi dalį savo PĮ, o tuo pačiu ir technologinius prioritetus. Dėl to yra svarbu, kad API produktas būtų kuo tolerantiškesnis naudotojo aplinkai. Yra siekiama kurti moduliarų API, kuris veiktų kaip statybos blokai (angl. „building blocks“) įmontuoti kiekviename aplikacijos sluoksnyje [Yea16].

Vertinimo kriterijai: kūrimo laisvė, moduliarumas, plečiamumas, struktūros keičiamumas, lankstumas formuojant struktūrą, suderinamumas (su skirtingais įrenginiais ir technologijomis).

2.4. Saugumas

API produktas yra papildomas langas kibernetinei atakai, todėl saugumo užtikrinimas yra neatsiejama API kūrimo dalis. Kuo servisai saugesni, tuo jie patikimesni, stabilesni ir patrauklesni naudoti. Klientui yra aktualu, kad būtų užtikrintas jo duomenų saugumas, o naudojamas produktas būtų atsparus atakoms.

Saugumui užtikrinti yra kuriami autentifikacijos ir autorizacijos mechanizmai. Jiems pasirenkami metodai atsižvelgiant į tai ar reikės identifikuoti:

1. Kas yra užklausos siuntėjas.
2. Ar siuntėjas prisistatė savo vardu.
3. Ar siuntėjui leidžiama pasiekti norimą šaltinį.

API kūrėjai patys nusprendžia, kaip bus implementuotas servisų saugumas, todėl svarbu kartu su dokumentacija pateikti pasirinktos realizacijos instrukciją. Svarbu, kad autentifikacijos ir autorizacijos mechanizmas būtų patikimas ir draugiškas vartotojui – lengvai implementuojamas ir panaudojamas.

Gera realizacija laikoma tuomet, kai autentifikacija ir autorizacija yra aiški, lengvai vykdoma.

Vertinimo kriterijai: patikimumas (atsparumas atakoms, duomenų saugumas), implementacija.

2.5. Panaudojamumas

Žmonės iš prigimties yra linkę rinktis tai, su kuo dirbti paprasčiau ir patogiau – lengviau įgyvendinti, perprasti ir pritaikyti. Lygiai taip pat ir su API produktais – įmantri autorizacija, skirtingos URL nuorodų schemos, funkcionalumo publikacija neįprastais metodais apsunkina naudojimąsi produktu, prarandamas intuityvumo aspektas. Bendrųjų technologinių standartų laikymasis ir papildomi tyrimai padeda išvengti situacijos, kuomet programuotojai apkraunami sauja naujų technologijų ir nėra aišku, ką būtent su jomis galima nuveikti.

Kuo produktas realistiškesnis, tuo klientams lengviau jį pritaikyti, o kūrėjams – parduoti. Didelė tikimybė, kad dirbtinai sukurtas produktas (laiku neatlikus naudotojų poreikių analizės, apgalvojus panaudos scenarijų) nebus naudojamas.

Panaudojamumo matas ryškiai koreliuoja su pagalbos klientams teikimu. Tikėtina, kad gavę neaiškų klaidos pranešimą, radę dokumentacijos spragų, klientai nežinos kaip elgtis, laikys produktą nestabiliu, sudėtingu ir prastos vertės.

Vertinimo kriterijai: realistiškumas, implementacijos paprastumas, atitikimas naudotojo poreikiams.

3. API KŪRIMO STRATEGIJOS

3.1. Apžvalga

Siekiant sukurti gerą API produktą svarbu pasirinkti kūrimo strategiją ir jos laikytis. Strategija apima rinkos analizę, identifikavimą kam produktas bus skirtas ir naudojamas, jo apimties apibrėžimą, procesų, metodų ir technologijų pasirinkimą. Strategija padeda planuoti kūrimo darbus, išvengti procesų ir rezultatų padrikumo bei užtikrinti kuriamo produkto kokybę. Toks išankstinis planavimas reikalauja pradinės investicijos, tačiau ilgainiui ji atsiperka. Tuo tarpu laiku neapibrėžus siekiamos apimties ir produkto paskirties rizikuojama patirti nuostolius – nėra aiškus klientų segmentas, kuriam API produktas bus tinkamas naudoti.

Kol kas nėra apibrėžta konkrečių API kūrimo strategijų, tačiau tai, kad API vis dažniau yra interpretuojamas kaip produktas rodo poreikį jas formuoti.

Šiuo metu API produkto kūrimui taikomos PĮ kūrimo strategijos – „Pirmiausia UI" (dar vadinama tradicine) ir neseniai ėmusi populiarėti „Pirmiausia API". Tradicinis PĮ kūrimo būdas – tai kuomet pirmiau sukuriama veikianti aplikacija, baigiami vidinio ir išorinio (vartotojo sąsajos) programavimo darbai. Tuomet, kaip papildomas projektas inicijuojama dalies arba viso funkcionalumo publikacija API produktu. Yra taikoma „Pirmiausia programavimas" substrategija, kuomet API produktas yra pagrindžiamas aplikacijos programiniu kodu – generuojama pagal esamas klasių esybes bei aplikacijos konfigūraciją [EFT16]. Strategijos tikslas yra pateikti API kaip papildomą produktą rinkoje trečiosioms šalims naudoti, integracijoms palengvinti. Taikant šią strategiją, kuomet naudojama jau egzistuojanti PĮ struktūra (aprašytos klasės, esybes, objektai) API publikuojamas pasitelkiant „SOAP" technologiją, tačiau kuomet pradedama nuo projektavimo, dažniau naudojama ir „REST" [Nei12].

Tuo tarpu pagrindinis strategijos „Pirmiausia API" principas yra atvirkščias – PĮ kūrimas pradedamas nuo API produkto ir tik tuomet pereinama prie programos UI kūrimo, kurios veikimas pagrindžiamas sukurtu API produktu. Strategija „Pirmiausia API" yra dalinama į du etapus [Lan14]:

1. API projektavimas (angl. „design API first").
2. API realizavimas (angl. „develop API first").

Kaip matyti, strategija „Pirmiausia API" skatina API kūrimo darbus pradėti nuo produkto projektavimo ir tik tuomet pereiti prie jo realizacijos – kanalų, kuriais produktas bus naudojamas,

apibėžimo. Šis principas dar vadinamas „Pirmiausia projektavimas“ (angl. „Design First“). Taikant strategiją „Pirmiauia API“ produktas publikuojamas pasitelkiant „REST“ technologiją [Nei12].

3.2. Palyginimas

Abi apžvalgoje minėtos strategijos yra sėkmingai taikomos, tačiau „Pirmiausia API“ yra mažiau ištirta. PĮ kūrėjams trūksta argumentacijos, kodėl ją rinktis verta. Daugelis lieka ištikimi tradiciniams PĮ kūrimo būdams, dėl jaučiamos rizikos, kad strategijos „Pirmiausia API“ taikymas neatsipirks.

Toliau bus pateikiamas strategijų palyginimas atsižvelgiant į galimybes formuoti gero API bruožus ir nustatoma, ar „Pirmiausia API“ yra verta rizikos ir pradinės investicijos, o jos taikymas atsiperka (1 lentelė).

Paprastai dokumentacija yra rašoma produkto kūrimo metu arba jį užbaigus. Tačiau „Pirmiausia API“ siūlo ją parengti jau projektavimo metu. Ši praktika vadinama „Pirmiausia Dokumentacija“ (angl. „Documentation First“) [Flo15]. Dokumentacija projektavimo etape veikia kaip tarpininkavimo priemonė tarp kūrėjų ir būsimų naudotojų, kadangi yra pateikiami būsimo API eksperimentiniai maketai, panaudos scenarijai ir žadamos galimybės. Jie projektavimo metu gali būti performuojami ir koreguojami pagal gaunamas pastabas iš būsimų naudotojų – programuotojų. Kartu yra užtikrinama, kad kuriama išsami, lengvai suprantama dokumentacija.

Šiandieninis pasaulis orientuojasi ties daugiakanale PĮ, suderinama su visais galimais įrenginiais – kompiuteriu (darbastaliu ir naršykle), mobiliaisiais telefonais, planšetėmis. Tačiau tradicinė strategija verčia API realizaciją prisirišti prie tam tikro įrenginio, kuriam aplikacija buvo skirta. Tai lemia dvigubas resursų sąnaudas ir papildomą darbą, kadangi funkcionalumas yra kuriamas ir plėtojamas dvejais kanalais – aplikacijos ir API. Tai kartu apsunkina PĮ palaikymą, kadangi visus pakeitimus ir taisymo darbus reikia daryti lygiagrečiai – aplikacijoje ir API. Tuo tarpu strategija „Pirmiausia API“ padeda išvengti šių problemų. Jos dėka vidinis programinis kodas atskiriamas nuo išorinio [Dad17]. Tokiu būdu įgalinamas aplikacijų, tinkamų naudoti bet kokiame įrenginyje, kūrimas. Tereikia suprojektuoti ir realizuoti tokį API produktą, kuris įgalintų komunikaciją su pasirinktais įrenginiais [Bau11]. Taikant tradicinę API kūrimo strategiją šios dinamikos yra netenkama.

Projektavimo darbai, klientų poreikių, technologijų ir įrankių analizė reikalauja papildomo laiko ir resursų. Lyginant strategijas šiuo aspektu, „Pirmiausia UI“ yra pranašesnė – aplikacija, pagal kurią kuriamas API, atstoja klientų reikalavimų specifikaciją. Tikėtina, kad aplikacija atitinka būtinus verslo ir rinkos poreikius. Tai reiškia, kad tuos pačius poreikius atitiks ir API produktas, todėl galima jaustis

užtikrintai dėl jo apimties. Taigi, iš pirmo žvilgsnio atrodytų, kad kurti API labiau apsimoka pagal strategiją „Pirmiausia UI“.

Tačiau API produkto esmė yra teikti funkcionalumą skirtinguose įrenginiuose palaikomai PĮ. Tai reiškia, kad produktas turi tikti integracijoms su skirtingomis aplikacijomis, dėl to nėra prasminga pasirinkti vieną ir ja pagrįsti produkto veikimą. Tai įspraudžia API kūrėjus į savotiškus rėmus [Flo15]. Be to, yra tikimybė, kad aplikacijoje nėra įgyvendintos visos funkcijos, būtinos API kanale. Tuomet sukuriamas produktas, kurio naudojimo metu randama realizacijos ir veikimo spragų.

Tokias spragas galima užpildyti laipsniškai plečiant funkcionalumą. Iš vienos pusės, toks principas atitinka šiuo metu ypač paplitusios „Agile“ metodologijos principą, tačiau kita vertus produktas gali lengvai netekti rišlumo ir nuoseklumo, pvz.: tie patys funkciniai objektai ar jų ypatybės naujuose plėtiniuose pradedami vadinti kitomis sąvokomis. Tuo pačiu produktą yra sunkiau palaikyti, nes atsiranda daug versijų, kurias reikia kontroliuoti [Tai15].

API produktas yra skirtas programuotojams, todėl nuoseklumas yra svarbi jo dalis – rišlų ir nuoseklų API yra lengviau kontroliuoti, paprasčiau suprasti ir pritaikyti. Šiuo atžvilgiu laimi strategija „Pirmiausia API“, kadangi kuriant produktą nuo pagrindų, projektavimo metu į planą įtraukiami ir apibrėžiami aktualūs funkciniai objektai bei standartai, kurių bus laikomasi.

Palyginus su tradicinio API kūrimo strategija, „Pirmiausia API“ palengvina modulių separaciją, padeda vengti priklausomybinių ryšių tarp funkcinių objektų [Yea16]. Taip yra todėl, kad API projektavimo metu nėra prisirišima prie esamos aplikacijos struktūros. Todėl galima apibrėžti procesus, už kuriuos moduliai bus atsakingi ir tokiu būdu paversti juos statybos blokais būsimoms aplikacijoms. Modulių atskirtis ne tik daro produktą lankstesniu, tačiau kartu ir stabilesniu – išvengiama priklausomybinių ryšių, dėl kurių atsiradus sistemos trikiams gali įvykti grandininė reakcija.

Kita vertus, rizikinga kurti į API orientuota programinę įrangą, kadangi jos veikimas tampa priklausomu nuo API stabilumo. Kadangi kuriama viename kanale, atsiradus trikiui kyla pavojus, kad kurį laiką naudotojai negalės naudotis tiek API, tiek UI paslaugomis.

Nors jau egzistuojanti PĮ struktūra gali blaškyti API kūrėjus ir projektavimo metu bus prisirišama prie esamų įgyvendinimo detalių, matomas ryškus projektavimo etapo poreikis. Visų pirma, daug paprasčiau ir pigiau modifikuoti projektą, nei performuoti ar keisti produkto realizaciją. Visų antra, projektavimo metu analizuojami būsimų naudotojų – programuotojų – poreikiai. Projektas yra taisomas pagal jų pastabas, tokiu būdu sukuriamas realistiškesnis, labiau naudotojų poreikius atitinkantis ir dėl to lengviau pritaikomas API. Taigi, net pasirinkus tradicinę API kūrimo strategiją ir

turint aplikaciją kaip pagrindą produkto specifikacijos kūrimui, yra patartina inicijuoti projektavimo darbus ir tik tuomet pereiti prie realizacijos.

1 lentelė. Strategijų palyginimo vertinimas. Vertinimo skalė: 1 – prastas, 2 – vidutiniškas, 3 – puikus

<table>
<tr><th colspan="3">Kriterijus</th><th>„Pirmiausia API“ + „Pirmiausia projektavimas“</th><th>„Pirmiausia UI“ + „Pirmiausia programavimas“</th></tr>
<tr><td rowspan="2">Dokumentacija</td><td colspan="2">Aiškumas</td><td>3</td><td>2</td></tr>
<tr><td colspan="2">Pilnumas</td><td>3</td><td>2</td></tr>
<tr><td rowspan="5">Stabilumas</td><td colspan="2">Nuoseklumas</td><td>3</td><td>2</td></tr>
<tr><td colspan="2">Pastovumas</td><td>2</td><td>2</td></tr>
<tr><td colspan="2">Vengiama monolitiškumo</td><td>3</td><td>1</td></tr>
<tr><td rowspan="2">Atsparumas</td><td>Trikių vengimas</td><td>2</td><td>1</td></tr>
<tr><td>Trikių šalinimas</td><td>3</td><td>2</td></tr>
<tr><td rowspan="6">Lankstumas</td><td colspan="2">Kūrimo laisvė</td><td>3</td><td>1</td></tr>
<tr><td colspan="2">Moduliarumas</td><td>3</td><td>1</td></tr>
<tr><td colspan="2">Plečiamumas</td><td>3</td><td>2</td></tr>
<tr><td colspan="2">Struktūros keičiamumas</td><td>1</td><td>1</td></tr>
<tr><td colspan="2">Lankstumas formuojant struktūrą</td><td>3</td><td>1</td></tr>
<tr><td colspan="2">Suderinamumas</td><td>3</td><td>1</td></tr>
<tr><td rowspan="2">Saugumas</td><td colspan="2">Implementacija</td><td>1</td><td>2</td></tr>
<tr><td colspan="2">Patikimumas</td><td>3</td><td>3</td></tr>
<tr><td rowspan="3">Panaudojamumas</td><td colspan="2">Produkto implementacija</td><td>1</td><td>2</td></tr>
<tr><td colspan="2">Naudotojų poreikių atitikimas</td><td>3</td><td>2</td></tr>
<tr><td colspan="2">Produkto realistiškumas</td><td>3</td><td>1</td></tr>
<tr><td colspan="3">Rezultatas:</td><td>46</td><td>29</td></tr>
</table>

4. GERO API KŪRIMO METODAI

4.1. Dokumentacija

Pagrindinis dokumentacijos tikslas – padėti naudotojams priimti sprendimus, suprasti ir efektyviai spręsti problemas. Kad šis tikslas būtų sėkmingai pasiektas, yra laikomasi bendrųjų dokumentacijos standartų (angl. „Good Documentation Practices, GDP“) [Ric10] – duomenų rinkimo, palaikymo, modifikavimo ir validacijos. Šių standartų laikymąsis suteikia rezultatui nuoseklumo, apsaugos nuo galimų klaidų.

Svarbu identifikuoti, ko būtent reikia naudotojui ir laikytis pasirinktos struktūros. Kadangi dokumentavimo procesas yra sudėtingas, įmonės tam samdo kvalifikuotus specialistus – techninius rašytojus (angl. „Technical Writter“).

Dažnai surinkus informaciją apie dokumentuojamą produktą, šiuo atveju – API, yra pamirštama pridėti galimus panaudojimo scenarijus, užklausų ir atsakymų pavyzdžius. Tačiau tai yra viena svarbiausių dokumentacijos sudedamųjų dalių – naudotojas gali fiziškai pamatyti, ko tikėtis, kaip formuoti užklausas. Kuo daugiau detalių įtraukiama, tuo lengviau vėliau nuspėti servisų veikimo niuansus ir sukurti stabilesnę integraciją. Pvz.: Įmonės „Adform“ darbuotojai API dokumentaciją [Adf17] formuoja pagal įmonės teikiamų funkcijų hierarchiją, pradedant nuo abstrakčios struktūros ir sulig kiekvienu lygiu pereinant prie konkretesnės informacijos. Galiausiai kiekvienas metodas pristatomas trumpu aprašymu, galimų reikšmių tipais, pavyzdžiais, panaudojimo scenarijais, apribojimais ir kita būtina informacija. Tuo tarpu „Rackspace Cloud Files“ [RCF17] API kūrėjai dokumentacija pateikia paprastu sąrašu, kurio kiekviename elemente pateikiami visi galimi užklausų ir atsakymų pavyzdžiai, nėra aiškinami naudojami laukai ir jų reikšmės. Tokioje dokumentacijoje ne tik sunku naviguoti, tačiau ir suprasti servisų galimybes [Sim16].

Dokumentacijos turinys yra svarbu, tačiau taip pat svarbu, kad turinys būtų lengvai formuojamas, modifikuojamas, tinkamai pateiktas ir lengvai skaitomas. Dėl to pastaruoju metu ėmė ypač populiarėti dokumentavimo servisai, leidžiantys generuoti dokumentaciją automatiškai, pasitelkiant programinį kodą. Tokiu būdu dokumentuoto API dizainas gali būti testuojamas dar prieš jį realizuojant, generuojant eksperimentinius API maketus. Klientams iš anksto pateikiami panaudos scenarijai, todėl lengviau suprasti API produkto veikimo principus [Flo16]. Be to, servisai yra pritaikyti specifiškai API produkto dokumentavimui, todėl infomacijos kėlimas ir naujinimas yra daug

paprastesnis nei, pavyzdžiui, anksčiau minėtos „Adform“ API dokumentacijos, kuri yra palaikoma „Kunstmaan“ turinio valdymo sistemos (angl. „content management system, CMS“) pagalba.

„StackShare“ duomenimis dažniausiai turinio formavimui naudojami dokumentavimo servisai – „Swagger UI“ ir „Apiary“ [Sta17] (2 lentelė). Deja šie servisai yra suderinami tik su „REST“ technologija pagrįstu API produktu. „SOAP“ API dokumentacijai yra naudojamos turinio valdymo sistemos. Jos, palyginus su dokumentavimo servisais, yra ribotesnės, nes neleidžia kurti tokių pat realistiškų API maketų ar leisti klientams išbandyti produktą.

„Swagger“ yra tinkamas taikant API kūrimą iš apačios į viršų (strategiją „Pirmiausia UI“), tuo tarpu „Apiary“ – kuriant iš viršaus į apačią (taikant strategiją „Pirmiausia API“). Dėl to „Apiary“ dar vadinamas API projektavimo įrankiu, kartu suteikiančiu galimybę dokumentuoti, kurti ir testuoti eksperimentinius API objektus ir taip palengvinti tarpininkavimą tarp API produkto kūrėjų ir jo naudotojų. Dėl to rezultate gaunama pilnesnė ir labiau kliento poreikius atitinkanti, t.y. aiškesnė dokumentacija.

Tačiau iš pažiūros senovinė „Swagger UI“ įrankio žiūryklė leidžia sukurti galingus API užklausų karkasus, kurie yra labai artimi realioms naudotojų integracijoms. Yra galimybė validuoti nuorodos parametrus, kelyje nurodomų objektų pavadinimus, o dokumentuotas API užklausas galima testuoti tiesiai naršyklės lange (vykdyti funkcijas bei gauti realius atsakymus iš serverių). „Apiary“ žiūryklė yra draugiškesnė vartotojui – užklausos išdėstomos aiškia, minimalistine tvarka. Testavimo galimybės panašios į „Swagger UI“ – valdyti parametrus, užklausa perduodamus laukus bei išsiųsti ją tiesiai naršyklės lange, gauti atsakymus iš serverių. Kiekviena užklausa „Apiary“ servisuose turi komentarų sekcija, kurioje naudotojai gali palikti savo atsiliepimus.

Nagrinėjant dokumentavimo įrankius stabilumo aspektu (t.y. kiek įrankiai leidžia pademonstruoti sukurto API stabilumą, o konkrečiau – pastovumą ir atsparumą), „API Blueprint“ nusileidžia „Swagger“ specifikacijai, nes nepalaiko podėlio kaupimo ir versijavimo. API produkto saugumo perteikimo galimybės yra ekvivalenčios – abu dokumentavimo servisai palaiko skirtingo tipo saugumo standartus – bazinė autentifikacija bei „OAuth“ autorizacija.

„Swagger UI“ redaktorius palaiko „YAML“ ir „JSON“ formatus, dėl to dokumentacijos kūrimas šiuo įrankiu reikalauja bent minimalios programavimo patirties arba laiko resursų struktūros perpratimui. Šiuo aspektu į priekį išsiveržia „Apiary“, kadangi „API Blueprint“ specifikacija yra pagrįsta „Markdown“ sintakse. Ši sintaksė yra lengvai skaitoma ir modifikuojama, todėl dokumentacijos turinį valdančiam asmeniui nėra būtina programavimo patirtis. Tai suponuoja, kad „Apiary“ lankstumo klausimu įgyja pranašumą.

„Apiary“ servisai yra naujesni už „Swagger UI“ todėl natūralu, kad „Swagger UI“ naudojanti bendruomenė yra didesnė. Juolab „Swagger UI“ yra nemokamas, atviro turinio įrankis, prieinamas viešoje „GitHub“ repozitorijoje. Tuo tarpu pilna „Apiary“ įrankio versija yra mokama, o jo specifikacijos kalba „API Blueprint“ reikalauja pradinės instaliacijos. Tačiau panaudojamumo aspektu „Apiary“ lenkia „Swagger UI“, kadangi papildomai leidžia sugeneruoti automatinį kodą, kurį naudotojas gali nusikopijuoti į savo aplikaciją ir tokiu būdu akimirksniu integruotis reikiamą funkcionalumą.

Abu dokumentacijos servisai turi savų pliusų ir minusų, tad renkantis vertėtų atsižvelgti į esamą situaciją. Jei yra ribotas biudžetas ir yra galimybė dokumentacijos rengimui skirti žmones su patirtimi technologijų srityje, vertėtų rinktis „Swagger UI“. Kita vertus, esant galimybei investuoti į API projektavimą, vertėtų susimąstyti apie „Apiary“ bei „API Blueprint“ specifikaciją, kadangi pastarieji leidžia pateikti aiškią dokumentaciją, ją formuoti yra paprasčiau.

2 lentelė. „Apiary“ ir „Swagger UI“ palyginimas API savybių perteikimo aspektu. Vertinimo skalė: 1 – prastas, 2 – vidutiniškas, 3 – puikus

Kriterijus	**„Apiary“**	**„Swagger UI“**
Dokumentacija	3	2
Stabilumas	1	3
Lankstumas	3	2
Saugumas	3	3
Panaudojamumas	3	2
Rezultatas:	13	12

4.2. Stabilumas

Vykdant pakeitimus arba naujinimus rekomenduojama taikyti versijavimą. Jei pakeitimai bus vykdomi jų neidentifikuojant, klientų integracijos netikėtai „sugrius“, o nustatyti dėl kokios priežasties problemos ėmė kilti bus sudėtinga. Tuo tarpu versijavimas suteikia galimybę kontroliuoti API produkto naujinimus ir lengvai identifikuoti naudojamą produkto versiją.

API produkto vystyme naujinimų kontroliavimas užima svarbų vaidmenį. Atlikus serviso naujinimus kurį laiką tenka vienu metu palaikyti kelias versijas. Tokiu būdu siekiama užtikrinti, kad

klientų integracijos veiks be pertraukimų. Kita vertus, kelių versijų palaikymas gali užsitęsti net iki metų ir tai reikš, kad ilgą laiką serveriai bus papildomai apkrauti. Kuo labiau apkraunami serveriai – tuo daugiau stabilumo yra netenkama. Todėl labai svarbu pokyčius kontroliuoti, vengti pernelyg dažno versijų naujinimo ir rūpintis, kad žinios apie pokyčius sėkmingai pasiektų naudotojus. Prasta informacijos sklaida kelia riziką sulaukti klientų nepasitenkinimo ir teikiamų paslaugų atsisakymo.

Versijų pasikeitimus patartina kaupti pokyčių registre (angl. „change log“). Produkto versiją galima perduoti siunčiant antrašte (3 pav.) arba nurodyti užklausos nuorodoje (4 pav.) – URL kelyje (1 ir 2) bei parametro reikšme (3).

```
/*Įprasta antraštė*/
Accept: application/json
/*Antraštė su versijavimu*/
Accept: application/vnd.myapi.v2 + json
```

3 pav. Versijavimas užklausos antrašte

```
/*Pavyzdinės užklausų nuorodos:*/
1. http://www.example.com/Services/2017/02/18/ExampleService.svc/wsdl /*SOAP*/
2. https://www.example.com/v1/request /*REST*/
3. https://www.example.com/request?version=1 /*REST*/
```

4 pav. Versijavimas užklausos nuoroda

Renkantis būdą versijavimui reikėtų atsižvelgti į API naudotojus – smulkesni klientai teikia pirmenybę versiją perduoti nuoroda, tuo tarpu didesnės įmonės renkasi antraštes. Tai lemia skirtingi metodų privalumai ir trūkumai [Lev16a].

Nors versijavimas URL nuoroda yra aiškesnis, šis būdas pažeidžia pagrindinį „REST“ technologijos principą – vienas šaltinis turi būti identifikuojamas vienu URL keliu. Metodo parametrais yra priimta perduoti informaciją apie vykdomą funkciją, o API versija yra greičiau realizacijos atributas, taigi parametrų paskirtis jų neapima. Taigi, pranašumą įgyja versijavimas antraštėmis. Jį yra lengviau realizuoti ir palaikyti, be to – suteikiama daugiau stabilumo klientų integracijoms. Pamiršus pridėti versiją į antraštę arba ją nurodžius nekorektiškai klientus galima automatiškai nukreipti į iš anksto numatytą, bazinę API versiją. Tačiau toks sprendimas ne visuomet pasiteisina, kadangi kiekvieną versiją gali reikalauti skirtingo apdorojimo, todėl gali kilti nenumatytų veikimo problemų. Be to, versijos perdavimas antrašte apsunkina testavimą – prarandama galimybė

užklausos nuorodą testuoti mygtuko paspaudimu naršyklėje ir atsiranda būtinybė konstruoti pilną užklausą.

Produkto stabilumas koreliuoja su serverių apkrova. Apkrova yra mažinama įgalinus podėlio kaupimą (angl. „caching“), grupines (angl. „batch“) operacijas bei srautinį siuntimą (angl. „streaming“).

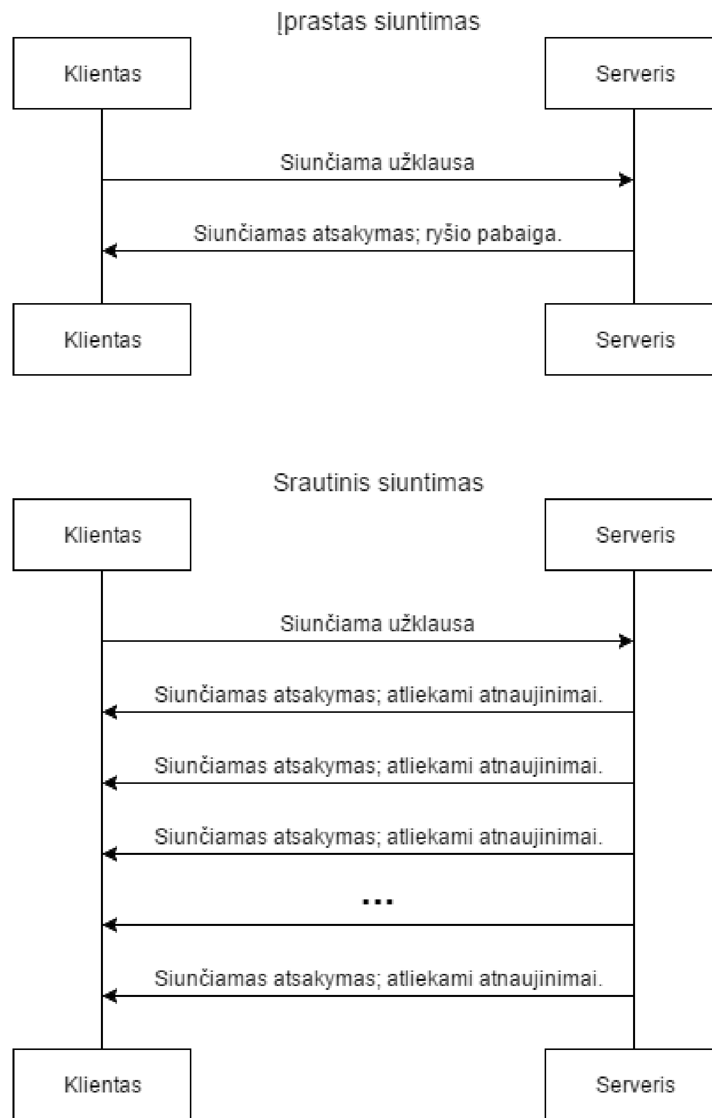
Podėlio kaupimas yra realizuojamas tinklų sietuve (angl. „gateway“) atvirkštinio įgalioto serverio metodu (angl. „reverse proxy“). Kai yra siunčiama saugi užklausa kokiu nors pasirinktu API serviso adresu, atvirkštinis įgaliotasis serveris įrašo grąžinamą atsakymą į podėlį ir tuomet, vykdant analogiškas užklausas pakartotinai, įrašytas atsakymas yra iš karto pateikiamas klientui. Tai reiškia, kad įgyvendinus podėlio kaupimą API servisams nebereikia iš naujo generuoti tų pačių rezultatų.

Grupinių operacijų veikimas yra pagrįstas procesų skirstymu į dalis ir tų dalių rikiavimu į specifinę vykdymo seką. Tai reiškia, kad procesai nebus vykdomi vienu metu ir serveriai nebus perkrauti. Šis funkcionalumas papildomai leidžia sumažinti tiesioginę naudotojo sąveiką, procesams galima nustatyti laikmačius ir gauti pranešimus, kuomet laikmačiai baigiasi. Ši funkcija padeda lengviau lokalizuoti kurio proceso metu įvyko gedimas. Tačiau grupinės operacijos turi ir trūkumų [Reh12] (3 lentelė).

3 lentelė. Grupinių operacijų privalumai ir trūkumai

Privalumai	Trūkumai
✓ Mažina serverių apkrovą. ✓ Pasikartojantys procesai atliekami greičiau, nes nereikia tiesioginio naudotojo įsiterpimo. ✓ Palaikymui nėra reikalinga speciali PĮ. ✓ Padeda lokalizuoti gedimus. ✓ Gali veikti atsijungus nuo serverio.	× Naudotojai turi būti specialiai apmokyti. × Tam tikrais atvejais implementacija yra brangi. × Jei procesas užtrunka per ilgai, sekantys po jo procesai irgi yra atidedami arba iš viso nevykdomi.

Srautinio siuntimo metu „HTTP“ ryšys yra kaip įmanoma ilgiau paliekamas atviru. Tokiu būdu yra įgalinami realaus laiko informacijos mainai (5 pav). Toks pastovusis ryšys (angl. „persistent connection“) suteikia galimybę mainytis informaciją iš karto po jos sugeneravimo, t.y. nereikia papildomai kreiptis į serverį, kad būtų atsiųsti naujausi duomenys.



5 pav. Įprasto ir srautinio siuntimo veikimo principas

Kad būtų išlaikytas servisų nuoseklumas ir granuliuotumas, reikėtų laikytis vieningų parametrų ir turinio laukų vardinimo standartų – apsibrėžti pagrindines reikšmes globaliai ir vėliau, reikiamose klasėse, jas paveldėti [Amb14]. Tokiu būdu sumažėja rizika maišyti laukų reikšmes tarpusavyje ir jas panaudoti netinkamai. Granuliuotumas yra aktualus kuomet dirbama su užklausomis, grąžinančiomis didelį rezultatą, pavyzdžiui – kelis objektus. Jis suteikia galimybę filtruoti, rikiuoti (angl. „sorting“) bei skirstyti rezultatus į puslapius (angl. „pagination“). Taigi, jo dėka išvengiama nereikalingos serverių apkrovos, nes gaunamas siauresnis rezultatas.

4.3. Lankstumas

Produktas įgyja lankstumo, kuomet naudotojui yra paliekama galimybė rinktis, kokiu būdu bus nustatomi užklausos atributai, tokie kaip metodo versija ar turinio duomenų formatas. Kai kurie API kūrėjai leidžia klientams patiems pasirinkti formą užklausos atsakymui. Pavyzdžiui, „Google Plus“ leidžia programuotojams gauti pilną, neformatuotą užklausos rezultatą ir tuomet patiems įvardinti reikiamus laukus. Tuo tarpu komandų bendradarbiavimui PĮ kurianti įmonė „Asana“ leidžia programuotojams perrašyti (angl. „override“) užklausos atsakymą ir patiems nuspręsti, kurie laukai turėtų būti įtraukti. „Asana“ grąžina tik identifikacinio numerio lauką, visus kitus laukus pasirenka programuotojas [DuV17].

Moduliarumas arba kitaip – API paskirstymas statybos blokais – įgyvendinamas pirmiau projektuojant API ir tik tuomet produktą įgyvendinant. Taip yra paprasčiau produktą pagal atliekamas operacijas suskirstyti į funkcinius objektus. Toks paskirstymas suteikia klientui aiškumu apie integracijos galimybes bei padeda orientuotis ieškant informacijos apie reikalingos srities užklausas.

Papildomo lankstumo ir dinamikos API produktui suteikiama įgalinus API grandinės kūrimą (angl. „API chaining“). API grandinė leidžia kombinuoti kelis API metodus į vieną užklausą ir gauti vieną, bendrą rezultatą. Rezultatas gaunamas paeiliui vykdant kiekvieną metodą ir jo rezultatą persiunčiant sekančiam. API grandinė gali turėti tik vieną deklaratyvų metodą, t.y. gali būti sudaryta vien tik iš „GET“ arba papildomai vieno deklaratyvaus – „PUT“, „POST“ arba „DELETE“ metodo. Deklaratyvus metodas laikomas grandinės šeimininku, juo grandinė arba pradedama, arba užbaigiama.

Galimybė kurti API grandinę leidžia kiekvieną užklausą paversti servisu, sumažinti kliento integracijos kodo apimtį. Dėl to, kad yra ryškiai sumažinama procesų vykdymo trukmė, API tampa pritaikomesnė mobiliesiems telefonams – veikimo greitis padidėja 50-75% [Rub15]. Kita vertus, svarbu atsižvelgti į turimus resursus ir programuotojų gebėjimus, kadangi prasta šio funkcionalumo implementacija gali sukelti rimtų problemų. Dėl šios priežasties PĮ kūrėjų požiūris į API grandinės kūrimą išlieka kontroversiškas (4 lentelė).

4 lentelė. API gradinės kūrimo privalumai ir trūkumai [San17]

Privalumai	Trūkumai
✓ Serveriui reikia apdoroti tik vieną užklausą ir išsiųsti tik vieną atsakymą vietoje keleto, todėl API veikia greičiau. ✓ Efektyviau kombinuoti užklausas siuntimo metu ir iš karto gauti atsakymą, nei po kelių užklausų įvykdymo filtruoti skirtingus rezultatus ir vykdyti paiešką. ✓ Paprastesnė operacijų automatizacija.	× Būtinai podėlio kaupimo ir grupinių operacijų palaikymas. × Prastas įgyvendinimas gali lemti begalines, kilpines operacijas, kuomet kreipiamasi į tuos pačius API objektų šaltinius. × Kenkėjiškos programos gali pasinaudoti galimybę siųsti kombinuotas užklausas, kuriomis bus siekiama griauti sistemą.

4.4. Saugumas

Saugumui palaikyti yra implementuojami autentifikacijos ir autorizacijos mechanizmai. Autentifikacija yra prisijungimo būdas, kurio metu (naudojant autentifikacijos protokolą) serveriui perduodami šifruoti arba nešifruoti naudotojo duomenys. Tuo tarpu autorizacija – yra verifikacija, kad toks sujungimas yra galimas. Autorizacija vykdoma po autentifikacijos [Lev16b]. Autentifikacijai ir autorizacijai naudojami įvairūs metodai: bazinis, „Digest“ ir „OAuth“.

Pagrindinis bazinės autentifikacijos principas – naudotojo slapyvardį ir slaptažodį perduoti naudojant „HTTP“ arba „HTTPS“ agentus. „HTTP“ agento realizacija yra elementari, ji nereikalauja slapukų (angl. „cookies“) kaupimo, sesijos identifikavimo ar prisijungimo puslapių. Užtenka standartinių „HTTP“ antraštės laukų. Perdavimo metu duomenys užkoduojami „Base64“ formatu, tačiau priešingai nei „HTTPS“ jie nėra papildomai šifruojami (angl. „encrypted“) ir maišomi (angl. „hashed“). Tai reiškia, kad jie gali būti lengvai pavogti bei kenksmingai panaudoti.

Naudojant „HTTP“ prisijungimo duomenys yra perduodami antrašte, o naršyklė kuriam laikui turi juos kaupti podėlyje. Podėlio kaupimo standartai kiekvienai naršyklei yra skirtingi, pvz.: „Microsoft Internet Explorer“ kaupia ir saugo duomenis 15 minučių. Tuo tarpu kenkėjiškos programos gali įsilaužti ir nutekinti sukauptus duomenis. Be to, „HTTP“ nesuteikia galimybės klientui atsijungti. Kita vertus yra specialūs metodai podėlio išvalymui, vienas jų – nukreipti naudotoją į nuorodą tame pačiame domene prisijungimo duomenis pakeičiant neteisingais. Deja šis metodas nėra vienodai pritaikomas skirtingose naršyklėse, todėl visapusiška realizacija yra sudėtinga.

„Digest“ autentifikacija leidžia aplikacijai identifikuoti, ar siuntėjas prisistatė savo vardu. Lyginant su baziniu, šis autentifikacijos būdas yra pranašesnis, nes prieš persiunčiant vartotojo slapyvardį ir slaptažodį jie yra šifruojami maišymo algoritmu. Kita vertus duomenų integralumą ir privatų ryšį bazinės autentifikacijos metu galima užtikrinti „HTTP“ kombinuojant su „TSL“ is „SSL“ kriptografiniais protokolais, t.y. naudoti „HTTPS“. Šie protokolai papildomai įgalina viešo rakto autentifikaciją, tačiau jų kombinacija sulėtina veikimą. Tuo tarpu „Digest“ kombinuoti su „SSL“ ar „TSL“ nėra prasminga ar reikalinga, todėl ši autentifikacija veikia greičiau.

Pagrindinis „Digest“ autentifikacijos trūkumas – serveris naudotojui suteikia vienkartinį raktą (angl. „nonce“), kurį panaudojus reikia autentifikuotis iš naujo. Tai reiškia, kad klientas yra priverstas kiekvieną kartą siųsti dvi užklausas vietoje vienos. Tai apkrauna serverius ir lėtina naudojimosi produktu procesą. Nors ši ypatybė apsaugo nuo pakartojimo atakų (angl. „replay attacks“), ji pažeidžiama įsiterpusio atakuotojo (angl. „man in the middle attack, MITM“).

Atviroji autentifikacija – „Oauth“ (angl. „Open Authentication) yra autentifikacijos protokolas, leidžiantis aplikacijoms atlikti specifines operacijas kieno nors vardu. Tai reiškia, kad nėra dalinamasi naudotojo slapyvardžiu ir slaptažodžiu, o protokolas veikia kaip autorizacija. „OAuth“ buvo sukurtas bazinės autentifikacijos keliamoms rizikoms išvengti. Protokolo veikimas yra pagrįstas žyme (angl. „token-based). Tai reiškia, kad kiekvienam naudotojui yra sugeneruojama unikali, saugi žymė (dar vadinama raktu), kuri yra perduodama aplikacijai autorizavimo metu. Tokiu būdu išvengiama dalinimosi asmenine naudotojų informacija [May15].

Nors „OAuth“ autentifikacijos mechanizmas yra sudėtingesnis nei bazinės, autorizacinės žymės įgyvendinimas yra to vertas. Tiek „OAuth“, tiek bazinė (kombinuojant su „SSL“ protokolu) autentifikacijos yra panašiai saugios. Tačiau naudotojo duomenų persiuntimas yra rizikingesnis – įsibrovimo atveju kenkėjiškos programos gali nutekinti naudotojų duomenis ir jų vardu pasinaudoti API produktu, t.y. gauti pilną prieigą prie PĮ. Tuo tarpu „OAuth“ autorizacinės žymės generavimas suteikia daugiau kontrolės PĮ savininkams, kadangi leidžia sekti kiekvieną prisijungusų prie API įrenginį bei nustatyti kokią sritį (angl. „scope“) duomenų autorizuota aplikacija gali pasiekti. Be to, žymei galima suteikti gyvavimo trukmę (angl. „time to live, TTL“), kas padeda apsisaugoti nuo pakartojimo atakų, kadangi žymės po nustatyto laiko tarpo automatiškai atsinaujina. Jei pastebima, kad specifinė žymė generuoja didelę serverių apkrovą ar yra naudojama neteisėtai, ją galima padaryti nebeveiksnia [Har13].

API raktus (autorizacines žymes) patariama generuoti naudojantis saugiais maišymo algoritmais (angl. „secure hashing algorythms, SHA“). Pvz.: naujo API naudotojo kūrimo metu sugeneruoti ir jam

priskirti kokią nors „SALT“ reikšmę. „SALT“ reikšmė gali būti pasirinkta konstanta arba kiekvienam naudotojui atsitiktiniu būdu sugeneruota nauja reikšmė, saugoma duomenų bazėje [Jos16] (6 pav.).

```
const SALT_VALUE = "abcd123";
var auth_token = SHA(User.ID + SALT_VALUE);
```

6 pav. Unikalios saugumo žymės „auth_token“ generavimas naudojant konstantinę „SALT“ reikšmę „SALT_VALUE“

API kūrėjai saugumo užtikrinimui siūlo papildomai naudoti baltojo sąrašo funkcionalumą (angl. „whitelisting functionality“), kuriuo leidžiama apriboti operacijų prieinamumą, pvz.: ištrinti objektus su „DELETE“ metodu leisti tik autorizuotiems naudotojams. Įgyvendinant iš pradžių yra taikomi aukščiausio lygio apribojimai, vėliau mažinami pagal poreikį [Amb14].

4.5. Panaudojamumas

API kūrėjai rekomenduoja vykdyti panaudojamumo testavimą (angl. „usability testing“) ir retkarčiais atlikti realizacinę patikrą, kuomet nauji naudotojai iš realaus naudotojų segmento mėgina kurtis eksperimentines API integracijas.

Šių procesų metu yra tikrinama kiek API produktas atitinka gero API bruožus. Panaudojamumo testavimo metu yra stebima ar produkto dokumentacija sukurta pakankamai intuityviai, kur trūksta aiškumo ir galėtų būti tobulinama. Taip pat atkreipiamas dėmesys ar produktas leidžia pasiekti naudotojui reikiamą rezultatą atliekant kuo mažiau žingsnių ir įtraukiant kuo mažiau skirtingų detalių. Tokiu būdu realizacinės patikros metu yra nustatoma ar produktas yra pakankamai stabilus, lankstus ir pritaikomas praktiškai.

Kita vertus, šių procesų metu yra tikrinami tik pavieniai naudotojai, tad yra sunku identifikuoti visas tobulintinas produkto vietas. Alternatyvus arba lygiagrečiai vykdytinas variantas yra rinkti grįžtamąjį ryšį iš klientų. Jo rinkimui yra buriamos specialios bendruomenės, kuriami forumai, naudojami klaidas fiksuojantys agentai (angl. „error trackers“), bei pagalbos tarnybų (angl. „help desk“) platformos, pvz.: „kayako“, „JIRA“. Svarbu tinkamai įvertinti gaunamos informacijos racionalumą, o fiksuojamas klientų problemas spręsti proaktyviai ir reaktyviai.

5. API PUBLIKACIJA

5.1. Apžvalga

API nėra nauja sąvoka programuotojams. Tačiau pastaruoju metu pagreitėjusį API ekosistemos augimą galima paaiškinti vis labiau populiarėjančia tendencija publikuoti API. Publikacija realizuojama „REST“ arba „SOAP“ žiniatinklio tarnybomis (angl. „web services“), kurios suteikia prieigą prie reikiamo PĮ funkcionalumo per adresą saityne (angl. „World Wide Web“).

Kad būtų pilnai suprasti šių technologijų veikimo principai, paaiškinti „REST“ ir „SOAP“ tarnybų sąvokas neužtenka. Reikia papildomai įsigilinti į jų panaudojimo kontekstą – technologijų kombinaciją, dalimi technologijos esančius arba papildomai prijungiamus protokolus ir standartus.

5.2. „REST“ ir „SOAP“

5.2.1. Apžvalga

„SOAP“ (angl. „Simple Object Access Protocol“) – tai protokolas, skirtas mainytis struktūrizuota informacija kompiuterių tinklais. Jis naudojamas kartu su „WSDL“ (angl. „Web Service Description Language“) dokumentu, kuris yra formatuojamas naudojant „XML“ (angl. „eXtensible Markup Language“) duomenų aprašymo kalbą. „SOAP“ veikia virš tinklo protokolų, todėl papildomai naudojamas „HTTP“ (angl. Hyper Text Transfer Protocol“), kurio pagalba aprašymas tampa pasiekiamas per adresą saityne. Siunčiant „SOAP“ užklausą informacija yra perduodama servisais, iškviečiant procesus su reikiamais objektais, pvz.: *gautiKnygas*, *gautiMokėjimus*.

„WSDL“ dokumentas yra suformuotas taip, kad būtų lengvai apdorojamas kompiuteriu. Jis apibrėžia žiniatinklio tarnyba teikiamą funkcionalumą: nusako, kaip servisas gali būti iškviečiamas, kokių parametrų tikėtis ir kokia duomenų struktūra yra grąžinama [W3C19].

Tuo tarpu „REST“ (angl. „Representational State Transfer“) yra ne protokolas, o architektūrinių taisyklių rinkinys. Priešingai nei „SOAP“ tarnyba, „REST“ leidžia didelį operacijų rinkinį suskirstyti CRUD (angl. „Create-Read-Update-Delete) operacijomis tokiomis kaip „GET“, „POST“, „PUT“ ir „DELETE“. Tokiu būdu galima lengvai manipuliuoti turimais ištekliais – parametrais bei kintamaisiais. Pavyzdžiui (7 pav.), kaip pasiekti užklausai reikalingą šaltinį (angl. „resource“)

„REST“ servisui gali būti pranešama tiek pačiame URL kelyje (angl. „path“, 1), tiek papildomais nuorodos parametrais (2).

```
http://maps.googleapis.com/maps/api/staticmap?center=Sydney,NSW&zoom=14&size=400x400&sensor=false
```

7 pav. Pavyzdinė „REST“ API URL nuoroda

Klientas siunčia užklausą serveriui, serveris nustato užklausai reikiamą šaltinį ir grąžina rezultatą bei jo atributus.

5.2.2. Palyginimas

Tiek „SOAP“, tiek „REST“ yra geros ir naudojamos technologijos. „SOAP“ naudojama senose sistemose, skirtose mokėjimams bei finansinėms tranzakcijoms, pvz.: „Paypal SOAP API“, „Salesforce SOAP API“. Tuo tarpu „REST“ – naujesnėse sistemose, socialiniuose tinkluose bei raiškiojoje medijoje, pvz: „Slack REST API“, „Twitter REST API“ [WTS16].

Analizuojant technologijas dokumentavimo aspektu, „REST“ išsiveržia į priekį, kadangi technologija yra suderinama su dabar plačiai naudojamais dokumentavimo servisais „Apiary“ ir „Swagger UI“. Tuo tarpu „SOAP“ servisus tenka dokumentuoti turinio valdymo sistemomis, kurių galimybės visokeriopai pademonstruoti produkto funkcionalumą yra ribotesnės.

Nuoseklumo atžvilgiu „REST“ technologija nusileidžia „SOAP“, kuri palaiko įvairias specifikacijas (pvz.: WS-Adressing, WS-Security) skirtas servisų formavimui bei turi šabloną duomenų atvaizdavimui – WSDL. Tuo tarpu „REST“ technologija reikalauja papildomo įdirbio formuojant ir apibrėžiant šiuos standartus ir specifikacijas produkto kūrimo metu. Tačiau tai smulkmena palyginus su technologijos privalumais stabilumo klausimu – „REST“ leidžia kaupti podėlį [Fra13] ir manipuliuoti ištekliais per URL nuorodą, o ne kreipiantis į patį API, kaip yra „SOAP“ atveju. Tai padeda išvengti PĮ monolitiškumo ir skaidyti servisus į statybos blokus. Be to, galimybė siųsti individualias operacijas, o ne jų komplektus, leidžia išvengti papildomos CPU apkrovos. Tuo tarpu naudojant „SOAP“ vienu metu yra siunčiamas didelis operacijų rinkinys bei sudėtinga, daugiau užimanti ir sunkiau apdorojama „XML“ sintaksė.

„REST“ naudotojams suteikia daugiau pasirinkimo laisvės. Pavyzdžiui, detales apie užklausą galima perduoti URL keliu, parametrais, antraštėmis, turinio laukais. Tuo tarpu „SOAP“ metodų

nuorodos yra pastovios ir nekintančios, todėl informacija apie užklausas gali būti perduota tik antraštėmis ir turinio laukais. Lankstumo „REST“ technologijai prideda ir tai, kad siunčiamų užklausų turinys nėra tiesiogiai pririštas prie šaltinių. Tai įgalina duomenų formatų įvairovę. Yra palaikomi tokie duomenų formatai kaip „JSON“, „YAML“, „XML“ ir kiti. „SOAP“ šiuo klausimu yra ribotesnis, kadangi palaikomas tik vienas – „XML“ – formatas, kurio karkasas yra sudėtingas tiek skaityti, tiek serveriams apdoroti. Dėl to akivaizdu, kad „SOAP“ technologija yra mažiau lanksti nei „REST“.

Saugumo aspektu laimi „SOAP“, kadangi saugumas ir autorizacija yra protokolo dalis (WS-Security). Yra palaikomi duomenų vientisumo ir privatumo standartai. Tuo tarpu „REST“ saugumą reikia įgyvendinti papildomai. Be to, „REST“ technologija gali naudoti tik perdavimo lygio saugumą [Sta13], kurį teikia „HTTPS“ (t.y. „HTTP“ ir „TSL“ bei „SSL“ kombinacija). Tokio lygio saugumas veikia tik tarp kliento ir interneto paslaugos. Interneto paslaugai sąveikaujant su kitomis programomis nėra garantijos, jog duomenys perduodami taipogi „HTTPS“. Dėl to „REST“ laikomas mažiau saugiu ir patikimu. Kita vertus, gera papildoma saugumo implementacija, pavyzdžiui „OAuth“ naudojimas, atperka šią spragą.

Apibendrinant, galima daryti išvadą, kad produkto publikavimas „REST“ technologija yra paprastesnis. Technologija padeda lengviau kontroliuoti produktą ir jį pritaikyti praktiškai. Be to, „REST“, priešingai nei „SOAP“, turi aukštą suderinamumą su naršyklėmis ir mobiliosiomis aplikacijomis. Šia technologija publikuotas produktas yra draugiškesnis fiziniam naudotojui, tuo tarpu „SOAP“ – mechaniniam (aukštesnio lygio automatizacija).

Bendras technologijų vertinimas pateikiamas lentelėje (4 lentelė).

4 lentelė. „REST“ ir „SOAP“ palyginimas API savybių perteikimo aspektu. Vertinimo skalė: 1 – prastas, 2 – vidutiniškas, 3 – puikus

Kriterijus	**„REST“**	**„SOAP“**
Dokumentacija	3	1
Stabilumas	3	2
Lankstumas	3	1
Saugumas	1	3
Panaudojamumas	3	1
Rezultatas:	13	8

REZULTATAI IR IŠVADOS

Bakalaurinio darbo metu buvo gauti tokie rezultatai:

1. Identifikuoti gero API bruožai:
   - Aiški ir pilna dokumentacija.
   - Stabilumas. Vertinamas pagal produkto nuoseklumą, pastovumą, monolitiškumą ir atsparumą – trikių vengimą ir jų šalinimą.
   - Lankstumas. Vertinamas pagal programuotojų pasirinkimo laisvę skirtingoms technologijoms, produkto moduliarumą, plečiamumą, struktūros formavimą ir keičiamumą bei suderinamumą su skirtingais įrenginiais, technologijomis.
   - Saugumas. Vertinamas pagal saugumo implementacijos sudėtingumą ir patikimumą, t.y. atsparumą atakoms ir naudotojų duomenų saugumą.
   - Panaudojamumas. Vertinamas pagal produkto realistiškumą, implementacijos paprastumą ir atitikimą naudotojo poreikiams.
2. Identifikuotos API kūrimo strategijos:
   - „Pirmiausia API“ – API kūrimas pradedamas nuo produkto projektavimo. Vėliau, realizavus API produktą, kuriama į API servisus orientuota aplikacija
   - „Pirmiausia UI“ – API produktas kuriamas pagal jau esamą aplikaciją. Yra galimybė įterpti projektavimo etapą, tačiau dažniausiai prisirišama prie jau esamos PĮ struktūros.
3. Atlikta strategijų „Pirmiausia API“ ir „Pirmiausia UI“ palyginamoji analizė. Jos metu įvertintos galimybės formuoti gero API bruožus taikant vieną arba kitą strategiją. Nustatyta, kad taikant strategiją „Pirmiausia API“ sukuriamas geresnis API produktas, nei taikant „Pirmiausia UI“, todėl ir pati strategija vertinama kaip pranašesnė. Pagrindinė tokio rezultato priežastis – nėra prisirišama prie vienos specifinės aplikacijos realizacijos ir tokiu būdu sukuriamas lankstesnis ir realistiškesnis produktas, pritaikomas įvairesnėse integracijose. Nustatyta, kad projektavimo metu yra nuo pagrindų apsprendžiami realizacijos klausimai ir išvengiama resursų švaistymo, todėl patartina projektavimo etapą įterpti ir „Pirmiausia UI“ strategijos taikymo metu.
4. Identifikuoti metodai, kuriais yra formuojami gero API bruožai:
   - Aiški ir pilna dokumentacija formuojama naudojant dokumentacijos servisus, tokius kaip „Apiary“ arba „Swagger UI“. Turinys formuojamas atitinkamai „API

Blueprint“ arba „Swagger“ specifikacijomis. Dokumentacijos formuluotė lemia kaip gerai informacija bus pateikta ir suprasta, todėl ją siūloma kurti taikant strategiją „Pirmiausia dokumentacija“. Tokiu būdu gaunamas rezultatas veikia ne tik kaip pristatomoji, tačiau kartu ir kaip tarpininkavimo priemonė – kūrėjai demonstruoja produktą jo projektavimo fazėje ir modifikuoja pagal gaunamus atsiliepimus.

- Stabilumas užtikrinamas įgalinant užklausų versijavimą metodo nuoroda arba antraštėmis. Serverių apkrova kontroliuojama įgalinant podėlio kaupimą, grupines operacijas arba srautinį siuntimą.
- API lankstumas ryškiausiai formuojamas projektavimo etape – kuriama moduliari struktūra, produktas dalinamas į funkcinius objektus. Vėliau lankstumo pridėti galima įgalinus API grandinės kūrimą. Tačiau analizės metu paaiškėjo, kad grandinės kūrimas yra vengtina praktika – jos implementacija sudėtinga, todėl kelia papildomų rizikų.
- Saugumas užtikrinamas įgyvendinus autentifikaciją ir autorizacija. Autentifikacijos metu nustatoma ar naudotojui gali būti suteikta prieiga prie šaltinių, o autorizacijos metu nusakoma, kokio lygio prieiga turėtų būti suteikta naudotojams. Papildomai prieigą galima kontroliuoti įgyvendinus baltojo sąrašo funkcionalumą. Autentifikacijos raktams siūloma naudoti maišymo algoritmus, priskirti „SALT“ reikšmes, kad atakos metu būtų sunkiau dekoduoti informaciją apie naudotojus.
- Panaudojamumas priklauso nuo API produkto realistiškumo ir to kaip harmoningai tarpusavyje sąveikauja praktikos, naudojamos formuojant gero API bruožus.

5. Atlikta priemonių, naudojamų gero API formavimui, palyginamoji analizė:
    - Atlikta „Apiary“ ir „Swagger UI“ dokumentavimo servisų bei „API Blueprint“ ir „Swagger“ specifikacijų palyginamoji analizė. Įvertintos priemonių galimybės formuoti gero API bruožus. Nustatyta, kad „Apiary“ yra pranašesnis už „Swagger“, kadangi leidžia ne tik dokumentuoti, bet ir projektuoti API produktą. Taip pat „Apiary“ yra tinkamesnė priemonė dokumentacijai formuoti, kuomet taikoma strategija „Pirmiausia API“, kuri buvo identifikuota kaip geresnė (žr. 3 rezultatų punkte).
    - Atlikta versijavimo metodų palyginamoji analizė. Nustatyta, kad geriausia yra versiją identifikuoti antraštėje. Tokio tipo versijavimą yra lengva įgyvendinti ir palaikyti, o naudotojų integracijos naujinant versijas yra paveikiamos mažiausiai.

Versijavimas antrašte yra suderinamesnis su „REST“ technologija, kuri buvo identifikuota kaip geresnė (žr. 7 rezultatų punkte), kadangi nėra laužomas pagrindinis technologijos principas – kiekvienas API šaltinis yra identifikuojamas vienu URL keliu.

- Atlikta autentifikacijos įgyvendinimo būdų – bazinės, „Digest“ bei „OAuth“ – palyginamoji analizė. Nustatyta, kad „OAuth“ protokolo implementacija yra sudėtingiausia, tačiau užtikrina aukštesnio lygio naudotojo duomenų apsaugą. Be to, leidžia stebėti ir kontroliuoti API naudojimą. Nuspręsta, kad „OAuth“ yra vertas pradinės investicijos.

6. Identifikuotos API publikacijai naudojamos technologijos. Nustatyta, kad yra naudojamos žiniatinklio tarnybos. Implementacija pagrindžiama „REST“ architektūrinių taisyklių rinkiniu arba „SOAP“ protokolu.
7. Atlikta „REST“ ir „SOAP“ technologijų palyginamoji analizė. Nustatyta, kad „REST“ technologija yra pranašesnė už „SOAP“ 4 iš 5 gero API bruožų atžvilgiu. „SOAP“ yra pranašesnis tik saugumo klausimu, kadangi saugumo standartai yra protokolo dalis. Tačiau pasirinkus tinkama priemonių kombinaciją, galima lengvai užpildyti „REST“ saugumo spragą, pavyzdžiui – naudojant „SSL“ protokolą bei „OAuth“ saugumo standartą. Taip pat nustatyta, kad „REST“ technologija yra aukštesnio suderinamumo su API projektavimui, kūrimui ir palaikymui naudojamomis priemonėmis, pavyzdžiui – dokumentavimo servisais, naršyklėmis, platformomis.

Gauti rezultatai rodo aukštą strategijų: „Pirmiausia API“, „Pirmiausia projektavimas“, „Pirmiausia dokumentacija“ ir „REST“ technologijos tarpusavio suderinamumą, todėl galima daryti išvadą, kad tai yra potencialiausia kombinacija siekiant sukurti gerą API produktą.

ŠALTINIAI

[Adf17] Adform, *„Adform API Support“.* 2017. URL adresas: https://api.adform.com/.

[Amb14] Jordan Ambra, *„5 Golder Rules for Great Web API Design“,* 2014. URL adresas: https://www.toptal.com/api-developers/5-golden-rules-for-designing-a-great-web-api#Rule1_Documentation, žiūrėta: 2017-04-28.

[Bau11] Nikko Bautista, *„Creating an API-Centric Web Application“,* 2011. URL adresas: https://code.tutsplus.com/tutorials/creating-an-api-centric-web-application--net-23417, žiūrėta: 2017-05-13.

[Bea17] Vangie Beal, *„Application Programming Interface“*, 2017. URL adresas: http://www.webopedia.com/TERM/A/API.html, žiūrėta: 2017-04-23.

[Ber15] David Berlind, *„APIs Are Like User Interfaces – Just With Different Users in Mind“*, 2015. URL adresas: https://www.programmableweb.com/news/apis-are-user-interfaces-just-different-users-mind/analysis/2015/12/03, žiūrėta: 2017-04-23.

[Dad17] Dadi, *„Platform Concepts“,* 2017. URL adresas: https://dadi.tech/platform/concepts/api-first-and-cope/, 2017-05-13.

[DuV17] Adam DuVander, *„3 Ways to Make Your API Responses Flexible“,* 2017. URL adresas: https://zapier.com/engineering/flexible-api-responses/, žiūrėta: 2017-04-28.

[EFT16] Entity Framework Tutorial, *„What is Code First“*, 2016. URL adresas: http://www.entityframeworktutorial.net/code-first/what-is-code-first.aspx, žiūrėta: 2017-05-14.

[Flo15] Zachary Flower, *„API Design – A Documentation-First Aproach“,* 2015. URL adresas: https://www.sumologic.com/blog/code/api-design/, žiūrėta: 2017-05-13.

[Fra13] Steve Francia, *„REST Vs SOAP, The Difference Between Soap And Rest“,* 2013. URL adresas: http://spf13.com/post/soap-vs-rest, žiūrėta: žiūrėta: 2017-05-27.

[Har13] Jason Harmon, *„How OAuth 2 trumps Basic authentication“,* 2013. URL adresas: http://apiux.com/2013/07/10/oauth-2-trumps-basic-authentication/, žiūrėta: 2017-05-25.

[IBM14] IBM, *„API 101“*, 2014. URL adresas: https://developer.ibm.com/apiconnect/documentation/api-101/, žiūrėta: 2017-05-09.

[Yea16] Jennifer Yeadon, *„3 Benefits of API-First Design“*, 2016. URL adresas: https://www.smartfile.com/blog/3-benefits-of-api-first-design/, žiūrėta: 2017-05-12.

[Jos16] Carl Joseph, *„The difference between encryption, hashing and salting“*, 2016. URL adresas: https://gooroo.io/GoorooTHINK/Article/13023/The-difference-between-encryption-hashing-and-salting/2085#.WQuYQ4iGNhE, žiūrėta: 2017-05-05.

[Kas16] Diksha Kashyap, *„Designing a New Product: Design, Strategy and Policies“,* 2016. URL adresas: http://www.yourarticlelibrary.com/production-management/designing-a-new-product-product-design-strategy-and-policies/57456/, žiūrėta: 2017-05-08.

[Kuo16] Marc Kuo, *„Build or Buy? The Pros and Cons of using APIs“,* 2016. URL adresas: https://blog.routific.com/why-businesses-choose-apis-8ef622b6c1c8, žiūrėta: 2017-04-28.

[Lak14] Dariuš Laketčenko, *„Kompiuterių tinklų saugumo aiškinamasis žodynas“,* 2014. URL adresas: http://www.tinklusaugumas.lt/Application%20Programing%20Interface, žiūrėta: 2017-04-24.

[Lan14] Kin Lane*, „What is an API First Strategy? Adding Some Dimensions to This New Question“*, 2014. URL adresas: http://apievangelist.com/2014/08/11/what-is-an-api-first-strategy-adding-some-dimensions-to-this-new-question/, žiūrėta: 2017-05-10.

[Lev16a] Guy Levin, *„RESTFul API Versioning Insights“,* 2016. URL adresas: http://blog.restcase.com/restful-api-versioning-insights/, žiūrėta: 2017-05-15.

[Lev16b] Guy Levin, *„RESTful API Authentication Basics“*, 2016. URL adresas: http://blog.restcase.com/restful-api-authentication-basics/, žiūrėta: 2017-05-16.

[May15] Lexy Mayko, *„Choosing an OAuth Type for Your API“,* 2015. URL adresas: https://www.api2cart.com/blog/choosing-oauth-type-api/, žiūrėta: 2017-05-25.

[MM13] Mokslo Medis, *„Tyrimų metodai ir metodikos“,* 2013. URL adresas: http://www.mokslomedis.lt/tyrimu-metodai/, žiūrėta 2017-04-23.

[Nei12] Mark O Neill, “*Becoming ‘API First’: Beyond the traditional application server architecture“,* 2012. URL adresas: http://sdtimes.com/becoming-api-first-beyond-the-traditional-application-server-architecture/, žiūrėta: 2017-05-14.

[RCF17] Rackspace, *„Rackspace Cloud Files API 1.0“,* 2017. URL adresas: https://developer.rackspace.com/docs/cloud-files/v1/, žiūrėta: 2017-04-18.

[Reh12] Junaid Rehman, *„What are advantages and disadvantages of batch processing systems“,* 2012. URL adresas: http://www.itrelease.com/2012/12/what-are-advantages-and-disadvantages-of-batch-processing-systems/, žiūrėta: 2017-05-24.

[Ric10] Mark Richardson, *„Good Documentation Practices (GdocP) are Critical to Success!“*, URL adresas: http://learnaboutgmp.com/good-documentation-practices-gdp-are-critical-to-success/, žiūrėta: 2017-05-14.

[Rub15] Owen Rubel, *„Why The API Pattern Is Broken And How We Can Fix It“,* 2015. URL adresas: http://apievangelist.com/2015/05/05/guest-post-why-the-api-pattern-is-broken-and-how-we-can-fix-it/, žiūrėta: 2017-05-25.

[San17] Kristopher Sandoval, *„Weighing the Pros and Cons of API Chaining“,* 2017. URL adresas: http://nordicapis.com/weighing-the-pros-and-cons-of-api-chaining/, žiūrėta: 2017-05-25.

[Seg17] Nathan Segal, *„What Makes a Great API“,* 2017. URL adresas: http://www.htmlgoodies.com/beyond/reference/what-makes-a-great-api.html, žiūrėta: 2017-04-24.

[Sim16] Luke Simmons, *„Bad API documentation: Why and what you can do about it“,* 2016. URL adresas: https://cloudrail.com/bad-api-documentation-why-and-what-you-can-do-about-it/, žiūrėta: 2017-04-28.

[Sta13] Kęstutis Stankevičius, „REST architektūrinio stiliaus palyginimas su SOAP, įgyvendinant šiuolaikines interneto paslaugas“, 2013. URL adresas: www.mla.vgtu.lt/index.php/mla/article/download/mla.2013.16/pdf, žiūrėta: 2017-05-27.

[Sta17] StackShare, *„Apiary vs. ReadMe.io vs. Swagger UI“,* 2017. URL adresas: https://stackshare.io/stackups/apiary-vs-readme-io-vs-swagger-ui, žiūrėta: 2017-05-23.

[Tai15] Andrew Tait, *„Benefits of API First Development“,* 2015. URL adresas: http://blog.learningtree.com/benefits-of-api-first-development/, žiūrėta: 2017-05-13.

[TW17] ThoughtWorks, *„APIs as a product“,* 2017. URL adresas: https://www.thoughtworks.com/radar/techniques/apis-as-a-product, žiūrėta 2017-04-17.

[WTS16] Web Tech Sharing, *„SOAP vs. REST, Basic and difference“,* 2016. URL adresas: http://webtechsharing.com/soap-vs-rest/, žiūrėta: 2017-10-05.

[W3C19] W3Schools, *„XML WSDL“,* 1999-2017. URL adresas: https://www.w3schools.com/xml/xml_wsdl.asp, žiūrėta: 2017-05-09.

SĄVOKŲ APIBRĖŽIMAI IR SANTRUMPOS

API	– aplikaciju kūrimo sąsaja.
CPU	– (angl. „Central Processing Unit) centrinis procesorinis įrenginys, atsakingas už procesų analizavimą ir vykdymą.
CRUD	– (angl. „Create-Read-Update-Delete“) 4 pagrindinės operacijos skirtos duomenims kurti, modifikuoti, ištrinti ir gauti.
Geras API	– išsamiai dokumentuotas, aiškus, stabilus, lankstus, saugus, lengvai pritaikomas ir panaudojamas API produktas.
„Markdown“	– sintaksė, leidžianti tekstą rašyti tarytum paprastu redaktoriumi, tačiau papildomų žymių pagalba jį suformatuoti (padaryti kursyvų, pabrauktą, keisti dydį), įterpti nuorodas ir t.t.
Metodas	– žingsnių seka reikiamų resursų kūrimui (pvz.: gero API bruožų formavimui).
Nuoseklumas	– savybė, kuomet produktas kuriamas laikantis standartų, pvz.: vardinimo, duomenų saugumo užtikrinimo.
PĮ	– programinė įranga.
SDK	– PĮ kūrimo įrankis.
Serializacija	– duomenų struktūrų ar objektų būsenų konvertacija į duomenų formatą, kuris gali būti saugomas, pvz.: faile arba atminties buferyje.
SHA	– saugaus maišymo algoritmas.
Strategijai	– perspektyva, kuri apima ilgalaikius tikslus (pvz.: kaip bus formuojamas, palaikomas ir vystomas API produktas). Strategijos formavimo metu yra atsižvelgiama į organizacijos veiklos kryptis ir veiklos būdus; atliekama situacijos analizė, nustatomi turimų resursų panaudojimo scenarijai arba kūrimo būdai.
Technologija	– išteklių (pvz.: įrankių) praktinis taikymas PĮ kurti.
UI	– vartotojo sąsaja.