ŠIAULIŲ UNIVERSITETAS

Informacinių technologijų katedra

Tomas Šeirys

REST architektūra paremta MySQL duomenų bazių valdymo sistema

Bakalauro baigiamasis darbas

Vadovė dr. S. Ramanauskaitė

Šiauliai, 2014

ŠIAULIŲ UNIVERSITETAS

Informacinių technologijų katedra

TVIRTINU

IT katedros vedėja dr. A. Slotkienė
2014-06-02

REST architektūra paremta MySQL duomenų bazių valdymo sistema

Informatikos inžinerijos bakalauro baigiamasis darbas

Vadovė
IT katedros docentė dr. S. Ramanauskaitė
2014 m. gegužės 30 d.

Recenzentė
IT katedros docentė dr. A. Slotkienė
2014 m. birželio d.

Atliko
IT-10 gr. studentas T. Šeirys
2014 m. gegužės 30 d.

Šiauliai, 2010

UŽDUOTIS

Šio lapo realiai pildyti nereikia, tiesiog vietoj jo įsegsite darbo užduotį

DARBO SANTRAUKA

Tikslas	Sukurti MySQL duomenų bazių valdymo sistemos duomenų bazių duomenų valdymo sistemą, kuri duomenų perdavimui naudotų REST architektūros principus
Užduoties pagrindimas	REST architektūra gali būti naudojama padidinti įvairių sistemų integravimo galimybes, nes suteikia patogiai prieinamą sąsają prie sistemos funkcionalumo ir duomenų. Tad sukūrus sistemą MySQL DBVS valdymui REST architektūros pagalba, būtų galima kurti sistemas, kurios naudojasi kitų tiekėjų duomenimis.
Kuriamo produkto prototipai	restSQL – sistema MySQL arba MariaDB DBVS valdymui REST architektūra. mongoDB – DBVS, turintis REST palaikymą. elasticSearch – DBVS turintis REST palaikymą.
Technologinis sprendimas	Python, JSON
Naudotos diagramos	Veiklos, sekų, esybių ryšių, klasių, komponentų diagramos.
Testavimo apimtis	Funkcionalumo ir našumo testavimas.
Vartotojo dokumentacijos sudėtinės dalys	Diegimo ir konfigūravimo vadovai.

SUMMARY

MySQL Database Management System Based on REST Architecture

REST architecture allows easier collaboration between two systems, as provided an easy way to call necessary functions in remote system and to get desired results. Such systems as Amazon, eBay, Twitter have their own API with support of REST architecture. However the REST architecture is not provided by default in MySQL, therefore additional tools have to be used.

The purpose of this work is to design and implement a system, which could be used as a database management tool in REST architecture. Therefore in this thesis a requirement and architecture specifications are provided for such a system. The implemented system is tested and install and configuration manuals are provided.

TURINYS

ĮVADAS

Nuolat didėjantys IT projektų ir programų kiekiai labai apsunkinta duomenų sinchronizavimą tarp kelių nutolusių sistemų. Kadangi kiekviena sistema arba neturi, arba propaguoja savitą duomenų perdavimo protokolą, vis sunkiau yra nustatyti standartinį ir universalų sinchronizacijos protokolą. Šiuo metu labiausiai paplitę yra SOAP arba REST architektūra paremti protokolai, tačiau duomenų perdavimo ir saugojimo architektūros kiekvienoje sistemoje yra labai skirtingos. Kas dažai pareikalauja tarp kiekvienos sistemos, duomenų sinchronizacijos programinę įrangą kurti individualiai.

Šio darbo tikslas – sukurti MySQL duomenų bazių valdymo sistemos duomenų bazių duomenų valdymo sistemą, kuri duomenų perdavimui naudotų REST architektūros principus.

Darbe keliami šie uždaviniai:

1. Susipažinti su REST architektūra ir jos palaikymu skirtingose duomenų bazių valdymo sistemose.
2. Parengti sistemos reikalavimų ir architektūros specifikacijas.
3. Realizuoti suprojektuotą sistemą.
4. Atlikti sukurtos sistemos funkcionalumo testavimą.

1. REST ARCHITEKTŪROS IR JOS REALIZAVIMO GALIMYBIŲ APŽVAGA

1. 1 REST architektūros apžvaga

Sutrumpinimas REST reiškia Reprezentacinis būsenos persiuntimas (*angl.* Representaional State Transfer).Jis pagrįstas būsenomis, duomenų perdavimais, ir praktiškai visais atvejais yra naudojamas HTTP protokolas. Tai yra architektūrinis stilius, kuriant tinklo programas. Pagrindinė idėja yra pakeisti tokius sudėtingus protokolus kaip CORBA, SOAP ar RPC (*angl.* Remote procedure call. Nuotolinis funkcijų iškvietimas, pasinaudojant url adresu.) sujungiant dvi ar daugiau sistemų, ir vietoj jų naudoti paprasčiausią HTTP protokolą. REST architektūra pagrįstos sistemos naudoja HTTP protokolo užklausas duomenų nuskaitymui, duomenų atnaujinimui, ištrynimui ar įrašymui, paprasčiau tariant -- REST architektūra galima įgyvendinti visas keturias CRUD (Create/Read/Update/Delete) operacijas. Kadangi tai yra nesudėtinga architektūra, tai tampa puiki alternatyva SOAP ar RPC. Be to kad yra paprasta, REST architektūra yra labai lanksti, todėl kas liečia Web servisus, nėra nieko to, ko nebūtų galima įgyvendinti pasinaudojant REST architektūra. Svarbu suprasti, kad REST nėra standartizuota, todėl nėra vieno universalaus ir teisingo būdo, kaip įgyvendinti REST architektūra pagrįstą sistemą.

Kas liečia programavimą, REST yra puiki alternatyva RPC, dėl daugelio priežasčių. Pagrindinės priežastys:

- REST architektūra yra nepriklausoma nuo platformos. Sistemai neturės įtakos kokią operacinę sistemą naudoja klientas, ir kokią serveris. Dėl to mobiliosiose programėlėse duomenų transakcijoms dažnai yra pasirenkama REST architektūra.
- Nepriklausoma nuo programinės kalbos. Sistemos sukurtos pasinaudojant JAVA programavimo kalbą, gali pilnavertiškai atlikti duomenų transakcijas su sistemomis kurios yra parašytos Perl programavimo kalba, ar panašiai.
- REST architektūra yra pagrįsta plačiai paplitusiu ir nusistovėjusiu HTTP protokolu. Tai leidžia išnaudoti visus HTTP protokolo privalumus, nuo failų persiuntimo, iki SSL šifravimo.

Kad geriau pastebėtumėme REST architektūra pagrįstos sistemos pranašumą, paanalizuokime paprastą pavyzdį. Tarkime, klientas pasinaudodamas Web servisu, nori gauti informaciją apie vartotoją. Jei naudotumėme SOAP, užklausa būtų panaši į:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
     <soap:body pb="http://www.acme.com/phonebook">
                <pb:GetUserDetails>
                          <pb:UserID>12345</pb:UserID>
                </pb:GetUserDetails>
     </soap:Body>
</soap:Envelope>
```

1 pav. *SOAP užklausos pavyzdys*

Vidinės detalės dabar nėra svarbios. Visas blokas turėtų būti nusiųstas į serverį pasinaudojant HTTP protokolo POST užklausa. Kaip atsaką servisas turėtų pateikti XML tipo atsaką, tačiau jis bus

struktūrizuotas konkrečiai SOAP protokolui. Analogiška užklausa, pasinaudojant REST atrodytu panašiai į:

```
http://www.acme.com/phonebook/UserDetails/12345
```

2 pav. *REST užklausos pavyzdys*

Reikia atkreipti dėmesį, jog užklausoje nėra nurodomi jokie papildomi parametrai, kurie yra reikalingi REST architektūrai. Tai yra tik paprasta HTTP protokolu atliekama GET užklausa. Ir kadangi REST architektūra nėra standartizuota, atsakymas bus nestruktūrizuotas ir nepritaikytas konkrečiam protokolui. Galime pastebėti kad Web servisams, kurie bus pagrįsti SOAP protokolu, darbui palengvinti bus reikalingos įvairios bibliotekos, tam kad būtų galima greičiau ir programiškai paprasčiau išskaidyti SOAP užklausas ir atsakymus, tuo tarpu, kai REST architektūra pagrįstoms sistemoms reikės tik tinklo prieigos, ir jas testuoti galima tiesiog pasinaudojant naršykle. Tačiau REST supaprastinimui vis tiek yra kuriamos bibliotekos, tam kad būtų galima dar labiau supaprastinti programuotojo darbą.

Jei užklausoje reikia perduoti daugiau duomenų, ar kitokio formato duomenis, informacija galima nurodyti POST užklausos kintamųjų masyve, tačiau REST architektūroje yra priimta, kad GET užklausa dažniausiai atstoja duomenų nuskaitymą (CRUD-Read operaciją), ir neatlieka jokių duomenų pakitimų. Duomenų įrašymui, atnaujinimui ar ištrynimui yra naudojamos PUT ir DELETE užklausos. Tačiau jei duomenų užklausimas yra sudėtingesnis, POST užklausa taip pat gali būti naudojama.

REST architektūra pagrįstose sistemose, atsakymas gali būti formuojamas kaip XML failas. To pranašumas yra griežta grąžinamų duomenų struktūra. Tačiau dažnais atvejais, yra pasirenkami kiti duomenų grąžinimo formatai, tokie kaip CSV (*angl.* Comma Separated Values) arba JSON (*angl.* JavaScript Object Notation). Kiekvienas formatas turi savo privalumų ir trūkumų. XML turi griežtą ir lengvai plečiamą duomenų struktūrą, CSV yra labai kompaktiškas, o JSON yra lengva skaidyti pasinaudojant JavaScript ir PHP programavimo kalbas. Kai kuriose sistemose REST atsakymas būna tiesiog HTML kodas.

1. 2 Egzistuojantys REST architektūros naudojimo pavyzdžiai

Daug gerai žinomų sistemų naudoja REST architektūra pagrįstą API. Atkreipkime dėmesį į tai, kad kai kurios iš jų palaiko ir REST ir SOAP protokolą, tam kad klientas galėtų pasirinkti sau patogesnį duomenų apsikeitimo būdą. Tačiau dauguma atvejų, kai abu protokolai yra palaikomi, REST metodu pagrįstas užklausas būna lengviau sukurti, rezultatus - paprasčiau išgauti, o užklausų kūrimas ir rezultatų skaidymas reikalauja mažiau sisteminių resursų (atsakymai ir užklausos turi mažiau simbolių nei SOAP protokole). Keletas populiarių sistemų, kurios naudoja REST API:

- Google Glass API, dar žinomas kaip "Mirror API", naudoja tik REST API.
- Twitter sistemoje REST API yra naudojamas kaip pagrindinis API Twitter aplikacijose[9].
- Flickr.
- Amazon.com tiekia kelis REST API. Viena iš jų Amazon.com S3 saugyklos sistemoje[10].

- Atom yra RSS alternatyva pasinaudojant REST architektūros principais[11].
- Tesla Model S naudoja REST architektūra pagrįstas duomenų transakcijas tarp automobilio borto kompiuterio ir Android/iOS sistemų[8].
- Ebay, PayPal vartotojų panelėje.

1. 3 AJAX ir REST

AJAX (angl. Asynchronous Javascript And Xml) yra populiari tinklapių kūrimo technologija, kuri padeda tinklapiams bendrauti tarpusavyje pasinaudojat JavaScript programavimo kalbą. AJAX užklausos į serverį yra siunčiamos pasinaudojant XMLHttpRequest objektais. Atsakymai būna apdorojami JavaScript parašytu skriptu, ir atitinkamai nuo parašyto skripto, informacija yra dinamiškai naudojama tinklapyje. Dažnai AJAX aplikacijos naudojasi REST architektūros principais.

Kiekviena XMLHttpRequest užklausa gali būti interpretuojama kaip REST serviso GET užklausa, ir atsakymas dažnai būna JSON formatu, kuris dažniausiai pasitaiko REST architektūroje. Norint tinklapius padaryti dar dinamiškesnius, kartais REST atsakymas būna JSONP (*angl.* JavaScript Object Notation with Padding) formato. Tada atsakyme būna pateikti ne tik užklausti duomenys, bet ir JavaScript funkcijos pavadinimas, kurį reikia iškviesti gavus atsakymą.

1. 4 DBVS valdymo sprendimai, palaikantys REST architektūrą

Šiuo metu rinkoje yra daugybė analogiškų paslaugų ir programų, kurios padeda programuotojams aplikaciją prijungti prie nuotolinės duomenų bazės pasinaudojant REST architektūra. Kadangi tai yra puiki alternatyva standartiniams ODBC (*angl.* Open DataBase Connectivity) kolektoriams, norint išvengti sujungimo apribojimų, kurie atsiranda pasirenkant programavimo kalbą. Analizėje bus analizuojami tik kelios populiariausios REST architektūra pagrįstos duomenų bazės – restSQL, mongoDB ir elasticSearch. Apžvelgdami egzistuojančius sprendimus, atkreipsime dėmesį į tai, kokias duomenų bazes programa naudoja, arba gali naudoti ir tai, ar programa turi kokią nors grafinę sąsają, kuri palengvintu programos administravimą.

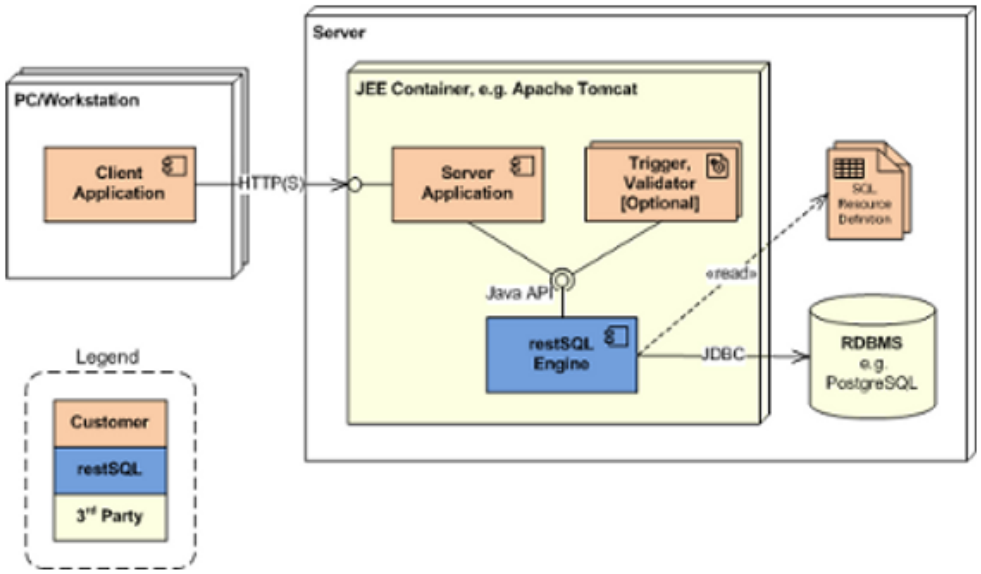
2 lentelė. REST architektūra pagrįstų sistemų, skirtų valdyti DBVS palyginimas

Sistemos pavadinimas	DBVS	Grafinės sąsajos palaikymas
restSQL	MySQL arba MariaDB	Ne
mongoDB	Objektiška	Ne
elasticSearch	Objektiška	Ne

1. 4. 1 restSQL

Tai ypač mažai resursų reikalaujanti duomenų bazė. Programa naudoja jau egzistuojančią MySQL, MariaDB arba PostgreSQL duomenų bazę. Šios programos pranašumas, palyginus su kitais sprendimais yra tai jog programa yra prašyta JAVA programavimo kalba, todėl gali būti naudojama ne tik kaip atskiras Web servisas, duomenim išgauti, bet gali būti naudojama ir kaip biblioteka kitoms JAVA kalba parašytoms programoms. Programa gali dirbti dviem rėžimais – paprastu ir WAR rėžimu. WAR rėžimas programą praplečia šifravimo ir autentifikavimo galimybėmis tarp kliento ir serviso. Taip pat programa išsiskiria tuo, kad turi savo trigerius, kurie leidžia praplėsti sistemos darbą, prijungiant dar kitas sistemas.

Kadangi restSQL yra sukurta pasinaudojant JAVA programavimo kalbą, programa yra nepriklausoma nuo platformos, t.y programa veikia ir Windows ir Unix šeimos operacinėse sistemose. Programa naudoja MIT licenziją[1].



3 pav. *restSQL komponentų diagramą [2]*

Iš komponentų diagramos matyti, kad restSQL programos varikliukas "restSQL Engine" yra kontroliuojamas per Java API[2]. Tai leidžia praplėsti programos darbą, pasitelkiant kitas JAVA kalba parašytas programas. RestSQL programa neturi grafinės sąsajos ir yra administruojama tik per konfigūracinius failus. To pasekmė yra ta, kad kaskart pakeitus programos konfigūracijas, programą reikia paleisti iš naujo.

1. 4. 2 MongoDB

MongoDB yra ne reliacinė duomenų bazė. MongoDB neturi tiesioginės REST sąsajos, tačiau yra lengvai praplečiama, pasinaudojant Node.js, DrowsyDromedary, Kule arba MongoBD Java programomis[1]. MongoDB patogi kuriant REST architektūra pagrįstas duomenų bazes, tuo, kad mongoDB yra objektinė duomenų bazė. Lengvas integravimas į Node.js yra pagrindinis faktorius, kodėl ši duomenų bazė yra dažnai renkamasi kuriant REST pagrindu pagrįstas aplikacijas. Šios duomenų bazės minusas yra tas, kad duomenų bazė naudoja savo sukurtą duomenų saugojimo variklį. Ji duomenis saugo BSON (*angl.* Binary javascript Object Notation) formatu kietajame diske, o atmintyje saugo tik nuorodas į duomenų failuose esančius objektus. Tai reiškia kad duomenų bazės spartai tiesioginės įtakos turi kietojo disko momentinė apkrova, bei nuskaitymo ir įrašymo greitis[3]. MongoDB neturi grafinės sąsajos ir administruojama yra per konfigūracinius failus. To pasekmė yra analogiška restSQL – kaskart pakeitus programos konfigūracijas, programą reikia paleisti iš naujo.

1. 4. 3 ElasticSearch

Tai JAVA kalba parašyta programa, todėl yra nepriklausoma nuo platformos, kas leidžia programai pilnavertiškai veikti tiek Windows tiek Unix šeimos operacinėse sistemose. ElasticSearch turi plačias praplėtimo galimybes, nes yra orientuota į klasteriavimą. Tai reiškia, kad esant būtinybei galima sujungti keletą serverių, ir taip praplėsti ElasticSearch programos naudojamus resursus. ElasticSearch turi jau integruotą ir plačiai dokumentuotą REST API. Sistema naudoja savo vietinę

objektinę duomenų bazę. ElasticSearch neturi jokios grafinės sąsajos, kuri palengvintų sistemos administravimą ir yra konfigūruojama tik per konfigūracinius failus.

2. REIKALAVIMŲ SPECIFIKACIJA

2. 1 Sistemos paskirtis

2. 1. 1 Projekto kūrimo pagrindas

Šiuo metu realizuoti duomenų sinchronizavimą tarp kelių nutolusių sistemų yra ganėtinai sudėtinga. Norint tai atlikti galima pasirinkti vieną iš scenarijų:

- Viena iš nutolusių sistemų būtų pirminė “master”, ir klonuoti visus duomenis į kitas nutolusias sistemas. Tačiau tokiu atveju duomenų pakitimas antrinėse sistemose “slave” nepasiekia “master” sistemos. Taip pat, realizuodami šį scenarijų, galima sinchronizuoti tik visą duomenų bazę, o ne tam tikras pasirinktas lenteles.
- Realizuoti sinchronizavimo API. Šiuo atveju reikėtų atlikti modifikacijas pačioje sistemoje. Modifikacijos sudėtingumas priklauso nuo sistemos pobūdžio ir norimo duomenų sinchronizavimo kiekio. Tačiau API realizavimas ir kūrimas gali reikalauti daug laiko ir testavimo. Taip pat, realizavus API reikalinga nuolatinė priežiūra, jei sinchronizuojamų duomenų struktūra pakistų.
- Vienoje iš sistemų sukurti pagrindinį duomenų serverį arba pagrindinę duomenų bazę, iš kurios informaciją ims kitos nutolusios sistemos. Tam reikia atlikti daug konfigūracijų pasirinktame duomenų serveryje, jei duomenų bazė veikia kaip servisas, reikia sukurti atitinkamus vartotojus, su atitinkamais leidimais. Taip pat gali reikėti atlikti modifikacijas pačioje nutolusioje sistemoje, turi būti tinkamai sukonfigūruotos ugniasienės.

Galimi ir kiti duomenų sinchronizavimo sprendimo atvejai, tačiau beveik visi jie reikalauja didelių modifikacijų pačioje sistemoje, papildomos programinės įrangos arba sudėtingų konfigūracijų. Nė vienas išvardintų sprendimo būdų nepasižymi universalumu, ar paprastu pritaikymu kitoms nutolusioms sistemoms.

2. 1. 2 Sistemos paskirtis

Sistemos tikslas – išspręsti duomenų sinchronizavimo sudėtingumo problemą, supaprastinti duomenų įterpimo ir išgavimo procesą pasinaudojant REST architektūros privalumais. Padaryti sistemą kuo paprastesnę sistemos vartotojui ir duomenų klientui. Palengvinti darbą programuotojams supaprastinant sistemą taip, kad duomenų išgavimas ar įterpimas nereikalautu daug programinių įgūdžių. Padaryti sistemą lengvai integruojamą į kitas sistemas.

Sistemos savybės:

- Paprastumas – sistema bus paprasta naudotis vartotojams.
- Universalumas – universalus duomenų priėjimas pasinaudojant REST architektūra.
- Saugumas – sistemoje naudojama autentifikavimas ir autorizacija.
- Integracija – lengva integracija į kitas sistemas pasinaudojant REST architektūra.

2. 1. 3 Sistemos pritaikymo sritys

Kuriamą sistemą yra galima pritaikyti bet kur, kur yra reikalinga pasiekti duomenis iš vienos arba keliu nuotolinių duomenų bazių.

- Norima atlikti apklausą keliose skirtingose sistemose. Sistemose kuriuose norima realizuoti apklausą, reikia įterpti atitinkamą Javascript kalba parašytą kodą, kurs apklausos rezultatus talpintu nuotolinėje duomenų bazėje. Tokiu būdu apklausos rezultatai gali būti surenkami iš kelių sistemų į vieną nutolusią duomenų bazę.
- Turime kelias elektronines parduotuves, patalpintas skirtinguose serveriuose. Įvykus pirkimui norima, kad produkto sumažėjimo faktas būtų įgyvendintas visose parduotuvėse. Po pirkimo atnaujiname produkto kiekio likutį kuriamoje sistemoje, ir kitos nutolusios sistemos matys produkto kiekio sumažėjimą.
- Turime mobiliąją aplikaciją, kuri ima duomenis iš nutolusio serverio. Mobiliajai aplikacijai duomenų išgavimo procesą bus lengva įgyvendinti naudojantis kuriama sistema.
- Vieno forumo integravimas keliose nutolusiose sistemose. Scenarijus panašus į pirmąjį. Tik šiuo atveju reikės išgauti ir komentarų įrašus, ir komentarų įterpimui paruoštą formą. Parašius komentarą, jis matysis visose nutolusiose sistemose.

2. 1. 4 Sistemos nauda užsakovui

Užsakovas siekia supaprastintą duomenų sinchronizavimo, įterpimo ir išgavimo proceso sprendimą tiekti kaip paslaugą, arba tiekti kaip programinę įrangą.

2. 1. 5 Sistemos nauda klientui

Sistema kuriama visoms klientų grupėms užsiimančioms programavimo paslaugomis, duomenų surinkimo paslaugomis, statistikos paslaugomis ar bet kokioms kitomis paslaugomis, kurioms reikalinga bent minimali duomenų saugojimo operacija, arba duomenų pateikimas bei išgavimas iš nutolusių sistemų.

2. 1. 6 Sistemos nauda kitoms suinteresuotoms šalims

Duomenų centrai (bus talpinama sistema). Statistika užsiimantys asmenys (kai kurie klientų projektai ir jų duomenys gali būti viešai prieinami).

2. 2 Projekto dalyviai

2. 2. 1 Užsakovas

Šiaulių universitetas. Su užsakovu derinama pagrindinė sistemos idėja ir baigta reikalavimų specifikacija jo patvirtinimui.

Kontaktinė informacija:

e.paštas: all@it.su.lt

adresas: Vilniaus g. 88, 76285, Šiauliai

tel.nr: (8 41) 595 800

2. 2. 2 Vykdytojas

Tomas Šeirys. Vykdytojas yra atsakingas už sistemos realizavimą, kuri tenkins reikalavimų specifikacijoje aprašytą sistemą.

Kontaktinė informacija:

e.paštas: sheirys2@gmail.com

tel.nr: +370 629 31914

2. 3 Vartotojai

Sistema neturi griežtai apibrėžtų vartotojo grupių, nes sistemoje nėra įdiegtai nei autorizacijos, nei autentifikavimo mechanizmai. Sistemos paleidimui bei konfigūravimo darbams atlikti bus reikalingas sistemos administratorius.

3 lentelė. Vartotojo „Sistemos administratorius“ detalizavimas

Vartotojo kategorija	**Sistemos administratorius**
Vartotojo sprendžiami uždaviniai	Sistemos administratorius prižiūri sistemos darbo tęstinumą, atlieka konfigūracinius darbus, paruošia operacinę sistemą programos darbui, diegia sistemos funkcionalumui reikalingus modulius, registruoja naujas nuotolines duomenų bazes
Patirtis dalykinėje srityje	Darbas su tinklais bei duomenų bazėmis. Gilus REST architektūros išmanymas
Patirtis informacinėse technologijose	Srities specialistas. Privalo gerai išmanyti duomenų bazes, REST architektūrą, mokėti SQL kalbą.
Papildomos vartotojo charakteristikos	Privalo mokėti SQL kalbą, dirbti su Unix šeimos operacinėmis sistemomis. Privalo puikiai išmanyti anglų kalbą, sistemos architektūrą, REST architektūrą. Jei kurio nors dalyko neišmano -- privalomas apmokymas.
Apsimokymo poreikis	Sistemos architektūros apmokymas, SQL kalbos apmokymas, REST architektūros pagrindų apmokymas, darbo iš komandinės eilutės Unix tipo operacinėse šeimose apmokymas.

4 lentelė. Vartotojo „Duomenų klientas“ detalizavimas

Vartotojo kategorija	**Duomenų klientas**
Vartotojo sprendžiami uždaviniai	Sistemos klientas naudoja visas sistemos funkcijas, kurios susijusios su duomenų atranka, kūrimu, redagavimu ir šalinimu, bei pačios duomenų bazės struktūros kaita.
Patirtis dalykinėje srityje	Darbas su tinklais bei duomenų bazėmis. Gilus REST architektūros išmanymas.
Patirtis informacinėse technologijose	Srities specialistas. Privalo gerai išmanyti duomenų bazes, REST architektūrą, mokėti SQL kalbą.
Papildomos vartotojo charakteristikos	Privalo mokėti SQL kalbą, dirbti su Unix šeimos operacinėmis sistemomis. Privalo puikiai išmanyti anglų kalbą, sistemos architektūrą, REST architektūrą. Jei kurio nors dalyko neišmano – privalomas apmokymas.
Apsimokymo poreikis	Sistemos architektūros apmokymas, SQL kalbos apmokymas, REST architektūros pagrindų apmokymas, darbo iš komandinės eilutės Unix tipo operacinėse šeimose apmokymas.

5 lentelė. Vartotojų prioritetai.

Vartotojo kategorija	**Prioritetas**
Duomenų klientas	Svarbiausias vartotojas.
Sistemos administratorius	Pirmaeilis vartotojas.

2. 4 Įpareigojantys apribojimai

2. 4. 1 Apribojimai sprendimui

- Sistema galės funkcionuoti dviem rėžimais – *solo* ir *multi*. *Multi* rėžime dirbdama sistema galės pasiekti kelias nuotolines duomenų bazes vienu metu. *Solo* rėžime dirbdama sistema galės pasiekti tik vieną duomenų bazę vienu metu.
- Sistema REST užklausos duomenis grąžins tik JSON duomenų formatu.
- Sistema turės būti paleidžiama kaip atskiras HTTP servisas, todėl sistema visada turės būti paleidžiama iš vartotojo turinčio prieigą prie tinklo.
- Jei sistema paleidžiama *solo* rėžimu, konfigūraciniame faile, arba komandinėje eilutėje paleidimo metu, būtinai turi būti nurodomas pagrindinis serveris, priešingu atveju sistema neveiks.
- Paleidžiant sistemą *multi* rėžimu, duomenų bazių konfigūraciniame faile, privalo egzistuoti bent viena veikianti nuotolinė duomenų bazė.
- Sistema gali neveikti sklandžiai jei yra paleidžiama Windows operacinėje sistemoje.

2. 4. 2 Diegimo aplinka

Pilnaverčiam sistemos funkcionavimui bus reikalingas nuolatos veikiantis, dedikuotas arba virtualus serveris ar bet koks kitoks serveris, kuriame diegiant sistemą bus suteikiamos administratoriaus teisės. Serveris turi tenkinti šiuos reikalavimus:

- Sistemos funkcionavimui reikalinga Unix šeimos operacinė sistema.
- Sistemos funkcionavimui reikalinga duomenų bazė MySQL-5.5.32 arba naujesnė. Arba MySQL alternatyvi MariaDB duomenų bazės versija, kuri turėtų ne mažesnį funkcionalumą nei MySQL-5.5.32.
- Sistemoje turi būti įdiegtas Python2.7 interpretatorius. Dėl Python kalboje pakitusių duomenų struktūrų Python versija negali būti naujesnė.
- Sistemoje turi būti įdiegtas MySQLdb-1.2.2 modulis, kuris reikalingas MySQL duomenų bazės valdymui pasinaudojant Python programavimo kalba.
- Sistemoje turi būti įdiegta libc6 arba naujesnė biblioteka, kuri reikalinga MySQLdb moduliui funkcionuoti.
- Norint gauti programinės įrangos atnaujinimus, sistemoje turi būti įdiegta Git versijavimo sistema suderinta su Github.com repozitorija.
- Sistemos nuolatiniam funkcionalumui užtikrinti reikalingas bent Tier-1 standartus tenkinantis serveris.

Paleidžiant, programa (jei nebuvo nurodyti kiti failai ir keliai) bandys nuskaityti *pymyrestrc.conf* ir *hosts.conf* failus, todėl rekomenduojama, kad vartotojas, kuris paleidžia programą turėtų teises kurti, rašyti ir skaityti failus esamame aplanke. Jei paleidžiant programą bus nurodomi kiti

konfigūraciniai failai ar kituose kataloguose, rekomenduojama programą leisti suteikiant root (Unix šeimų operacinių sistemų vartotojas turintis administratoriaus privilegijas) privilegijas.

2. 4. 3 Bendradarbiaujančios sistemos

Programa priklausomai nuo aplinkos kurioje ji veiks, bei paleidimo konfigūracijų naudos MySQL arba MariaDB duomenų bazes. Jei programa nebus susieta nė su viena duomenų baze, programa neveiks.

2. 4. 4 Sistemos kūrimo biudžetas

Sistemos kūrimo procesai nereikalauja biudžeto, nes yra pasirinkti atviro kodo sprendimai. Programos funkcionalumui reikalingas biudžetas yra neapibrėžtas ir priklauso nuo sistemos, kurioje veiks programa aplinkos. Programa finansavimo reikalaus tiek – kiek reikalaus sistema kurioje veiks programa.

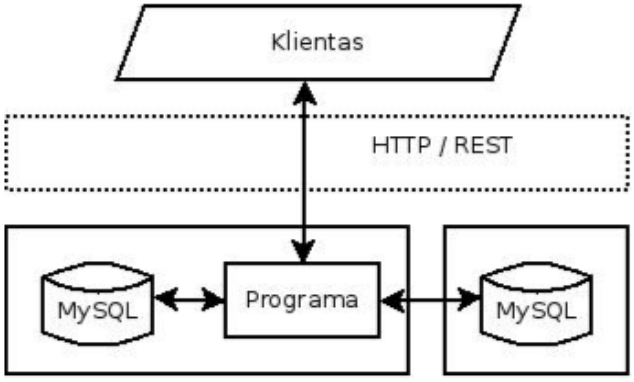
2. 4. 5 Sistemos kūrimo žingsniai

- Reikalavimų specifikacijos derinimas su užsakovu.
- Sistemos realizavimo darbai pagal sistemos terminuose 6 lentelėje.
- Sistemos diegimas, testavimas ir pateikimas užsakovui.
- Sistemos ilgalaikė priežiūra.

6 lentelė. Sistemos kūrimo terminai

Užduotis	**Terminas**
Suprojektuoti REST API.	2014-01-15
Suprojektuoti REST serverį.	2014-01-20
Realizuoti REST serverį.	2014-02-10
Suprojektuoti konfigūracijas.	2014-02-15
Realizuoti konfigūracijų nuskaitymą iš komandinės eilutės.	2014-02-20
Realizuoti konfigūracijų nuskaitymą iš konfigūracinių failų.	2014-02-25
Suprojektuoti nuotolinių duomenų bazių konfigūracijas.	2014-03-01
Realizuoti nuotolinių duomenų bazių konfigūracijų nuskaitymą.	2014-03-05
Realizuoti prisijungimą prie nuotolinės duomenų bazės.	2014-03-10
Realizuoti CRUD veiksmus per REST API.	2014-03-15
Realizuoti HEAD užklausos informaciją apie projektus.	2014-03-20
Realizuoti GET užklausoje atliekamus veiksmus.	2014-03-25
Realizuoti POST užklausoje atliekamus veiksmus.	2014-03-28
Realizuoti PUT užklausoje atliekamus veiksmus.	2014-04-01

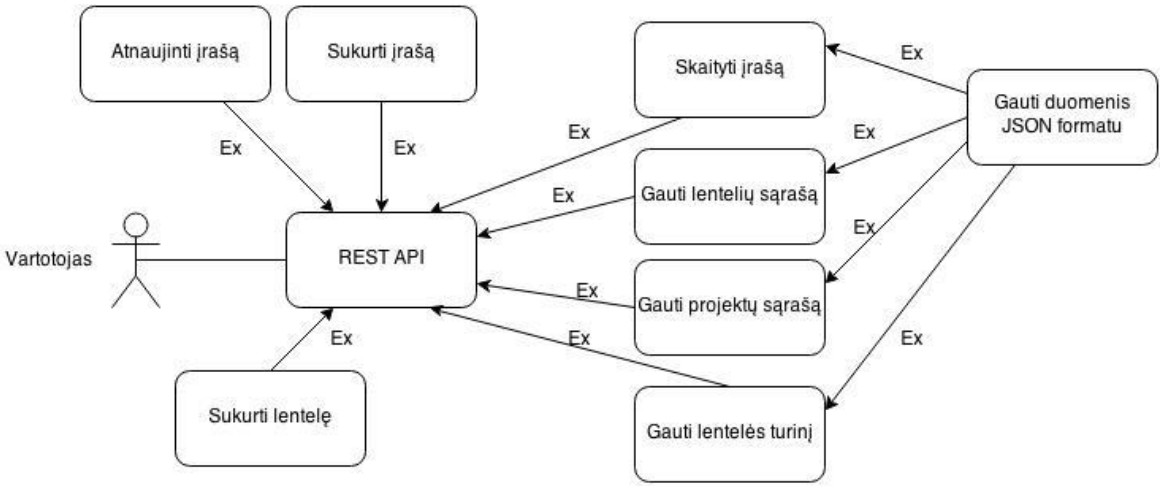
2. 5 Veiklos kontekstas



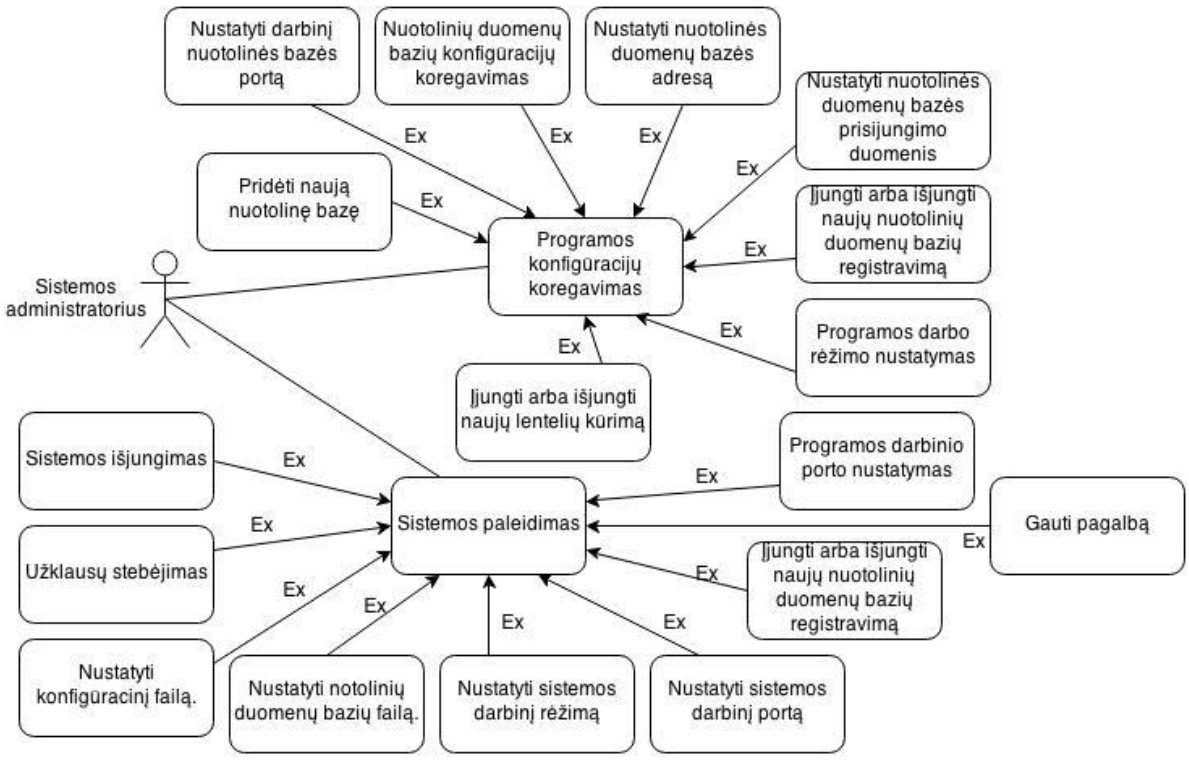
4 pav. *Veiklos konteksto diagrama*

2. 6 Produkto veiklos sfera

2. 6. 1 Sistemos ribos



5 pav. *Sistemos funkcijos iš vartotojo pusės*



6 pav. *Sistemos funkcijos iš administratoriaus pusės*

2. 6. 2 Panaudos atvejų sąrašas

7 lentelė. Panaudojimo atvejo „Sukurti įrašą“ detalizavimas

1.	PANAUDOJIMO ATVEJIS:	Sukurti įrašą
	Vartotojas/Aktorius:	Vartotojas
	Aprašas:	Vartotojas naudodamasis REST užklausa, gali sukurti naują įrašą pasirinktoje lentelėje.

8 lentelė. Panaudojimo atvejo „Skaityti įrašą“ detalizavimas

2.	PANAUDOJIMO ATVEJIS:	Skaityti įrašą
	Vartotojas/Aktorius:	Vartotojas
	Aprašas:	Vartotojas naudodamasis REST užklausa, gali skaityti įrašą pasirinktoje lentelėje.

9 lentelė. Panaudojimo atvejo „Gauti informaciją JSON formatu“ detalizavimas

3.	PANAUDOJIMO ATVEJIS:	Gauti informaciją JSON formatu
	Vartotojas/Aktorius:	Vartotojas
	Aprašas:	Vartotojas naudodamasis REST užklausa, duomenis apie lentelėje esančius įrašus gauna JSON duomenų formatu.

10 lentelė. Panaudojimo atvejo „Gauti lentelių sąrašą“ detalizavimas

4.	PANAUDOJIMO ATVEJIS:	Gauti lentelių sąrašą
	Vartotojas/Aktorius:	Vartotojas
	Aprašas:	Vartotojas naudodamasis REST užklausa, projekte esančių lentelių sąrašą.

11 lentelė. Panaudojimo atvejo „Gauti projektų sąrašą“ detalizavimas

5.	PANAUDOJIMO ATVEJIS:	Gauti projektų sąrašą
	Vartotojas/Aktorius:	Vartotojas
	Aprašas:	Aprašymas & Vartotojas naudodamasis REST užklausa, sistemoje esančių projektų sąrašą.

12 lentelė. Panaudojimo atvejo „Gauti lentelės turinį“ detalizavimas

6.	PANAUDOJIMO ATVEJIS:	Gauti lentelės turinį
	Vartotojas/Aktorius:	Vartotojas
	Aprašas:	Vartotojas naudodamasis REST užklausa, gali lentelėje esantį turinį.

13 lentelė. Panaudojimo atvejo „Gauti įrašą“ detalizavimas

7.	PANAUDOJIMO ATVEJIS:	Gauti įrašą
	Vartotojas/Aktorius:	Vartotojas
	Aprašas:	Vartotojas naudodamasis REST užklausa, gali gauti lentelėje esančio įrašo informaciją.

14 lentelė. Panaudojimo atvejo „Atnaujinti įrašą“ detalizavimas

8.	PANAUDOJIMO ATVEJIS:	Atnaujinti įrašą
	Vartotojas/Aktorius:	Vartotojas
	Aprašas:	Vartotojas naudodamasis REST užklausa, gali atnaujinti lentelėje esančio įrašo informaciją.

15 lentelė. Panaudojimo atvejo „REST užklausimas“ detalizavimas

9.	PANAUDOJIMO ATVEJIS:	REST užklausimas
	Vartotojas/Aktorius:	Vartotojas
	Aprašas:	Vartotojas naudodamasis REST užklausomis, sistemoje gali atlikti visus CRUD veiksmus.

16 lentelė. Panaudojimo atvejo „Sukurti lentelę“ detalizavimas

10.	PANAUDOJIMO ATVEJIS:	Sukurti lentelę
	Vartotojas/Aktorius:	Vartotojas
	Aprašas:	Vartotojas naudodamasis REST užklausa, gali lentelę projekte

17 lentelė. Panaudojimo atvejo „Projekto registravimas“ detalizavimas

11.	PANAUDOJIMO ATVEJIS:	Projekto registravimas
	Vartotojas/Aktorius:	Vartotojas
	Aprašas:	Vartotojas naudodamasis REST užklausa, gali sukurti naują projektą

18 lentelė. Panaudojimo atvejo „Konfigūracijų nuskaitymas“ detalizavimas

12. PANAUDOJIMO ATVEJIS:	Konfigūracijų nuskaitymas
Vartotojas/Aktorius:	Administratorius
Aprašas:	Vartotojas naudodamasis komandine eilute, paleidimo metu gali komandinėje eilutėje gali nustatyti sistemos konfigūracijas.

19 lentelė. Panaudojimo atvejo „Šalinti įrašą“ detalizavimas

13. PANAUDOJIMO ATVEJIS:	Šalinti įrašą
Vartotojas/Aktorius:	Vartotojas
Aprašas:	Vartotojas naudodamasis REST užklausa, gali šalinti įrašą pasirinktoje lentelėje.

20 lentelė. Panaudojimo atvejo „Šalinti lentelę“ detalizavimas

14. PANAUDOJIMO ATVEJIS:	Šalinti lentelę
Vartotojas/Aktorius:	Vartotojas
Aprašas:	Vartotojas naudodamasis REST užklausa, gali šalinti projekte egzistuojančią lentelę.

21 lentelė. Panaudojimo atvejo „Nustatyti rėžimą“ detalizavimas

15. PANAUDOJIMO ATVEJIS:	Nustatyti rėžimą
Vartotojas/Aktorius:	Administratorius
Aprašas:	Vartotojas sistemos paleidimo metu gali nustatyti sistemos darbo rėžimą

22 lentelė. Panaudojimo atvejo „Gauti pagalbą“ detalizavimas

16. PANAUDOJIMO ATVEJIS:	Gauti pagalbą
Vartotojas/Aktorius:	Vartotojas
Aprašas:	Vartotojas paleisdamas sistemą iš komandinės eilutės, gali gauti pagalbą, apie sistemą.

23 lentelė. Panaudojimo atvejo „Nustatyti darbinį portą“ detalizavimas

17. PANAUDOJIMO ATVEJIS:	Nustatyti darbinį portą
Vartotojas/Aktorius:	Administratorius
Aprašas:	Vartotojas paleisdamas sistemą iš komandinės eilutės, paleidimo metu gali nustatyti sistemos darbinį portą.

24 lentelė. Panaudojimo atvejo „Nustatyti pagrindinį serverį“ detalizavimas

18. PANAUDOJIMO ATVEJIS:	Nustatyti pagrindinį serverį
Vartotojas/Aktorius:	Administratorius
Aprašas:	Vartotojas paleisdamas sistemą iš komandinės eilutės, paleidimo metu gali nustatyti sistemos pagrindinį serverį.

25 lentelė. Panaudojimo atvejo „Stebėti užklausą“ detalizavimas

19. PANAUDOJIMO ATVEJIS:	Stebėti užklausą
Vartotojas/Aktorius:	Administratorius
Aprašas:	Administratorius programos darbo metu, gali stebėti programos apdorojamas užklausas.

26 lentelė. Panaudojimo atvejo „Įjungti nauju duomenų bazių registravimą“ detalizavimas

20. PANAUDOJIMO ATVEJIS:	Įjungti nauju duomenų bazių registravimą
Vartotojas/Aktorius:	Administratorius
Aprašas:	Administratorius prieš paleisdamas programą gali nustatyti ar ji užregistruos naujas duomenų bazes.

27 lentelė. Panaudojimo atvejo „Sistemos išjungimas“ detalizavimas

21. PANAUDOJIMO ATVEJIS:	Sistemos išjungimas
Vartotojas/Aktorius:	Administratorius
Aprašas:	Administratorius programos darbo metu, gali išjungti programą.

28 lentelė. Panaudojimo atvejo „Naujos nuotolinės duomenų bazės priėjimas“ detalizavimas

22. PANAUDOJIMO ATVEJIS:	Naujos nuotolinės duomenų bazės priėjimas
Vartotojas/Aktorius:	Administratorius
Aprašas:	Administratorius konfigūraciniuose failuose gali pridėti naują duomenų bazę.

29 lentelė. Panaudojimo atvejo „Nustatyti duomenų bazės adresą“ detalizavimas

23. PANAUDOJIMO ATVEJIS:	Nustatyti duomenų bazės adresą
Vartotojas/Aktorius:	Administratorius
Aprašas:	Administratorius prieš paleisdamas programą gali nustatyti duomenų bazės adresą.

30 lentelė. Panaudojimo atvejo „Nustatyti prisijungimo duomenis“ detalizavimas

24. PANAUDOJIMO ATVEJIS:	Nustatyti prisijungimo duomenis
Vartotojas/Aktorius:	Administratorius
Aprašas:	Administratorius, prieš paleisdamas programą gali nustatyti duomenų bazės prisijungimo duomenis.

2. 7 Funkciniai reikalavimai

31 lentelė. Funkcinio reikalavimo „Automatinis naujo projekto kūrimas“ detalizavimas

Reikalavimas nr.:	1.	**Reikalavimo tipas:**	0	**Įvykis/panaudojimo atvejo nr.:**	11
Aprašymas:	Automatinis naujo projekto kūrimas				
Pagrindimas:	Vartotojas norėdamas atlikti REST užklausas privalo būti sukūręs projektą.				
Šaltinis:	Užsakovas				
Tinkamumo kriterijus:	Kai vartotojas atlieka atitinkamą REST užklausą, sistema automatiškai užregistruoja naują projektą.				
Užsakovo tenkinimas:	4	**Užsakovo netenkinimas:**	3		
Priklausomybės:	-	**Konfliktai:**	Nėra		

32 lentelė. Funkcinio reikalavimo „Rankinis naujo projekto kūrimas“ detalizavimas

Reikalavimas nr.:	2.	**Reikalavimo tipas:**	0	**Įvykis/panaudojimo atvejo nr.:**	11
Aprašymas:	Rankinis naujo projekto kūrimas				
Pagrindimas:	Vartotojas norėdamas atlikti REST užklausas privalo būti sukūręs projektą.				
Šaltinis:	Užsakovas				
Tinkamumo kriterijus:	Kai vartotojas padaro atitinkamus pakitimus duomenų bazių konfigūraciniame faile, sistema pasileisdama automatiškai užregistruoja tą projektą.				
Užsakovo tenkinimas:	4	**Užsakovo netenkinimas:**	3		
Priklausomybės:	-	**Konfliktai:**	Nėra		

33 lentelė. Funkcinio reikalavimo „Naujos lentelės kūrimas“ detalizavimas

Reikalavimas nr.:	3.	**Reikalavimo tipas:**	0	**Įvykis/panaudojimo atvejo nr.:**	10
Aprašymas:	Naujos lentelės kūrimas				
Pagrindimas:	Vartotojas gali nuskaityti arba įrašyti duomenis tik į egzistuojančią lentelę.				
Šaltinis:	Užsakovas				
Tinkamumo kriterijus:	Kai vartotojas atlieka atitinkamą REST užklausą, sistema sukuria naują lentelę.				
Užsakovo tenkinimas:	4	**Užsakovo netenkinimas:**	3		
Priklausomybės:	1, 2	**Konfliktai:**	Nėra		

34 lentelė. Funkcinio reikalavimo „Lentelės įrašo nuskaitymas“ detalizavimas

Reikalavimas nr.:	4.	**Reikalavimo tipas:**	0	**Įvykis/panaudojimo atvejo nr.:**	2
Aprašymas:		Lentelės įrašo nuskaitymas			
Pagrindimas:		Vartotojas atlikdamas atitinkamą REST užklausą nori matyti lentelės įrašą.			
Šaltinis:		Užsakovas			
Tinkamumo kriterijus:		Kai vartotojas atlieka atitinkamą REST užklausą, sistema suranda ir grąžina vartotojui informaciją apie norimą įrašą lentelėje.			
Užsakovo tenkinimas:		4	**Užsakovo netenkinimas:**		3
Priklausomybės:		3	**Konfliktai:**		Nėra

35 lentelė. Funkcinio reikalavimo „Lentelės įrašų nuskaitymas“ detalizavimas

Reikalavimas nr.:	5.	**Reikalavimo tipas:**	0	**Įvykis/panaudojimo atvejo nr.:**	4
Aprašymas:		Lentelės įrašų nuskaitymas			
Pagrindimas:		Vartotojas atlikdamas atitinkamą REST užklausą, nori matyti visus lentelėje esančius įrašus.			
Šaltinis:		Užsakovas			
Tinkamumo kriterijus:		Kai vartotojas atlieka atitinkamą REST užklausą, grąžina visus įrašus esančius užklaustoje lentelėje			
Užsakovo tenkinimas:		4	**Užsakovo netenkinimas:**		3
Priklausomybės:		3	**Konfliktai:**		Nėra

36 lentelė. Funkcinio reikalavimo „Naujo įrašo įterpimas“ detalizavimas

Reikalavimas nr.:	6.	**Reikalavimo tipas:**	0	**Įvykis/panaudojimo atvejo nr.:**	1
Aprašymas:		Naujo įrašo įterpimas			
Pagrindimas:		Vartotojas atlikdamas atitinkamą REST užklausa, nori įterpti naują įrašą į egzistuojančią lentelę			
Šaltinis:		Užsakovas			
Tinkamumo kriterijus:		Kai vartotojas atlieka atitinkamą REST užklausą, sistema sukuria naują įrašą nurodytoje lentelėje.			
Užsakovo tenkinimas:		4	**Užsakovo netenkinimas:**		3
Priklausomybės:		4	**Konfliktai:**		Nėra

37 lentelė. Funkcinio reikalavimo „Įrašo šalinimas“ detalizavimas

Reikalavimas nr.:	7.	**Reikalavimo tipas:**	0	**Įvykis/panaudojimo atvejo nr.:**	13
Aprašymas:		Įrašo šalinimas			
Pagrindimas:		Vartotojas atlikdamas atitinkamą REST užklausą, nori pašalinti įrašą iš nurodytos lentelės.			
Šaltinis:		Užsakovas			
Tinkamumo kriterijus:		Kai vartotojas atlieka atitinkamą REST užklausą, sistema automatiškai pašalina įrašą iš nurodytos lentelės			
Užsakovo tenkinimas:		4	**Užsakovo netenkinimas:**		3
Priklausomybės:		6	**Konfliktai:**		Nėra

38 lentelė. Funkcinio reikalavimo „Įrašo atnaujinimas“ detalizavimas

Reikalavimas nr.:	8.	**Reikalavimo tipas:**	0	**Įvykis/panaudojimo atvejo nr.:**	8
Aprašymas:		Įrašo atnaujinimas			
Pagrindimas:		Vartotojas atlikdamas atitinkamą REST užklausą, nori atnaujinti lentelėje esantį įrašą.			
Šaltinis:		Užsakovas			
Tinkamumo kriterijus:		Kai vartotojas atlieka atitinkamą REST užklausą, sistema atnaujiną nurodytą įrašą, varotojo paduotais duomenimis.			
Užsakovo tenkinimas:		4	**Užsakovo netenkinimas:**		3
Priklausomybės:		6	**Konfliktai:**		Nėra

39 lentelė. Funkcinio reikalavimo „Nuotolinės duomenų bazės versijos išgavimas“ detalizavimas

Reikalavimas nr.:	9.	**Reikalavimo tipas:**	0	**Įvykis/panaudojimo atvejo nr.:**	17, 18
Aprašymas:	Nuotolinės duomenų bazės versijos išgavimas				
Pagrindimas:	Vartotojas atlikdamas tam tikrą REST užklausą, nori sužinoti nuotolinės duomenų bazės versiją.				
Šaltinis:	Užsakovas				
Tinkamumo kriterijus:	Kai vartotojas atlieka atitinkamą REST užklausą, sistema nuotolinėje duomenų bazėje atlieka *SELECT VERSION()* užklausą ir rezultatus pateikia vartotojui.				
Užsakovo tenkinimas:	4	**Užsakovo netenkinimas:**	3		
Priklausomybės:	-	**Konfliktai:**	Nėra		

40 lentelė. Funkcinio reikalavimo „Komandinės eilutės argumentų nuskaitymas“ detalizavimas

Reikalavimas nr.:	10.	**Reikalavimo tipas:**	0	**Įvykis/panaudojimo atvejo nr.:**	-
Aprašymas:	Komandinės eilutės argumentų nuskaitymas				
Pagrindimas:	Vartotojas paleisdamas sistemą per komandinę eilutę, nori nustatyti norimus nustatymus.				
Šaltinis:	Užsakovas				
Tinkamumo kriterijus:	Kai vartotojas paleidžia sistemą per komandinę eilutę, sistema nuskaito komandinėje eilutėje nurodytus nustatymus.				
Užsakovo tenkinimas:	4	**Užsakovo netenkinimas:**	3		
Priklausomybės:	-	**Konfliktai:**	Nėra		

41 lentelė. Funkcinio reikalavimo „Sistemos darbinio porto nuskaitymas iš konfigūracinio failo“ detalizavimas

Reikalavimas nr.:	11.	**Reikalavimo tipas:**	0	**Įvykis/panaudojimo atvejo nr.:**	17
Aprašymas:	Sistemos darbinio porto nuskaitymas iš konfigūracinio failo				
Pagrindimas:	Vartotojas nenorėdamas kaskart paleidžiant sistemą nurodyti darbinį sistemos portą, jį gali įrašyti į sistemos konfigūracinį failą.				
Šaltinis:	Užsakovas				
Tinkamumo kriterijus:	Kai vartotojas paleidžia sistemą komandinėje eilutėje nenurodydamas sistemos darbinio porto, sistema darbinį portą nuskaito iš konfigūracinio failo.				
Užsakovo tenkinimas:	4	**Užsakovo netenkinimas:**	3		
Priklausomybės:	-	**Konfliktai:**	Nėra		

42 lentelė. Funkcinio reikalavimo „Sistemos darbinio rėžimo nuskaitymas iš konfigūracinio failo“ detalizavimas

Reikalavimas nr.:	12.	**Reikalavimo tipas:**	0	**Įvykis/panaudojimo atvejo nr.:**	18
Aprašymas:	Sistemos darbinio rėžimo nuskaitymas iš konfigūracinio failo				
Pagrindimas:	Vartotojas nenorėdamas kaskart paleidžiant sistemą nurodyti sistemos darbinį rėžimą, jį gali išsaugoti sistemos konfigūraciniame faile.				
Šaltinis:	Užsakovas				
Tinkamumo kriterijus:	Kai vartotojas paleidžia sistemą iš komandinės eilutės nenurodydamas sistemos darbinio rėžimo, sistema iš konfigūracinio failo nuskaito kokiu darbiniu rėžimu ji turės dirbti.				
Užsakovo tenkinimas:	4	**Užsakovo netenkinimas:**	3		
Priklausomybės:	11	**Konfliktai:**	Nėra		

43 lentelė. Funkcinio reikalavimo „Duomenų bazės prisijungimo vartotojo nuskaitymas iš failo“ detalizavimas

Reikalavimas nr.:	13.	**Reikalavimo tipas:**	0	**Įvykis/panaudojimo atvejo nr.:**	24
Aprašymas:	Duomenų bazės prisijungimo vartotojo nuskaitymas iš failo				
Pagrindimas:	Vartotojas nori nurodyti duomenų bazės prisijungimo vardą				
Šaltinis:	Užsakovas				
Tinkamumo kriterijus:	Norint prisijungti prie duomenų bazės reikalingas vartotojo vardas.				
Užsakovo tenkinimas:	4	**Užsakovo netenkinimas:**	3		
Priklausomybės:	-	**Konfliktai:**	Nėra		

44 lentelė. Funkcinio reikalavimo „Duomenų bazės prisijungimo slaptažodžio nuskaitymas iš failo“ detalizavimas

Reikalavimas nr.:	14.	**Reikalavimo tipas:**	0	**Įvykis/panaudojimo atvejo nr.:**	24
Aprašymas:	Duomenų bazės prisijungimo slaptažodžio nuskaitymas iš failo				
Pagrindimas:	Vartotojas nori nurodyti duomenų bazės prisijungimo slaptažodį				
Šaltinis:	Užsakovas				
Tinkamumo kriterijus:	Norint prisijungti prie duomenų bazės reikalingas vartotojo slaptažodis				
Užsakovo tenkinimas:	4	**Užsakovo netenkinimas:**	3		
Priklausomybės:	-	**Konfliktai:**	Nėra		

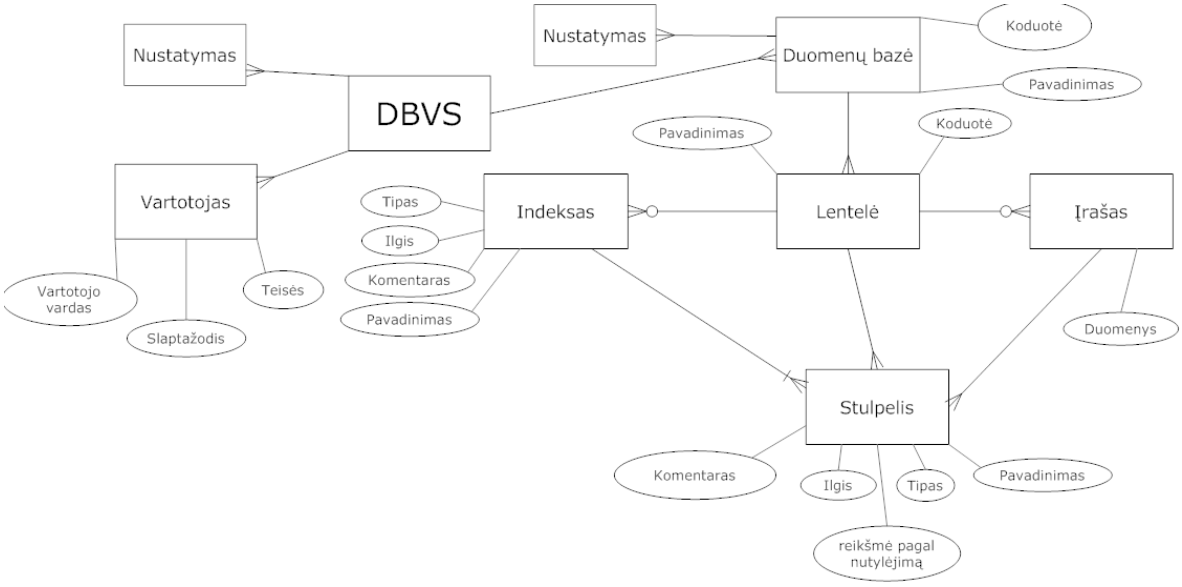
45 lentelė. Funkcinio reikalavimo „Duomenų bazės prisijungimo porto nuskaitymas iš failo“ detalizavimas

Reikalavimas nr.:	15.	**Reikalavimo tipas:**	0	**Įvykis/panaudojimo atvejo nr.:**	17
Aprašymas:	Duomenų bazės prisijungimo porto nuskaitymas iš failo				
Pagrindimas:	Vartotojas nori nurodyti duomenų bazės prisijungimo slaptažodį				
Šaltinis:	Užsakovas				
Tinkamumo kriterijus:	Norint prisijungti prie duomenų bazės reikia nurodyti portą				
Užsakovo tenkinimas:	4	**Užsakovo netenkinimas:**	3		
Priklausomybės:	-	**Konfliktai:**	Nėra		

46 lentelė. Funkcinio reikalavimo „Duomenų bazės adreso nuskaitymas iš failo“ detalizavimas

Reikalavimas nr.:	16.	**Reikalavimo tipas:**	0	**Įvykis/panaudojimo atvejo nr.:**	-
Aprašymas:	Duomenų bazės adreso nuskaitymas iš failo				
Pagrindimas:	Vartotojas nori nurodyti duomenų bazės adresą				
Šaltinis:	Užsakovas				
Tinkamumo kriterijus:	Norint prisijungti prie duomenų bazės reikia nurodyti adresą, kur jungtis.				
Užsakovo tenkinimas:	4	**Užsakovo netenkinimas:**	3		
Priklausomybės:	-	**Konfliktai:**	Nėra		

2. 7. 1 Reikalavimai duomenims



7 pav. *Sistemos duomenų loginė struktūra*

2. 8 Nefunkciniai reikalavimai

2. 8. 1 Reikalavimai panaudojamumui

- Galimybė manipuliuoti duomenimis REST architektūros pagrindu.
- Programa dirbs kaip atskiras HTTP serveris.
- Programos darbui bus nereikalinga duomenų bazė.

- Programa prisijungs prie nutolusių duomenų bazių kaip SQL klientas.
- Programa dirbs tik su MySQL arba MariaDB duomenų bazėmis.

2. 8. 2 Reikalavimai vykdymo charakteristikoms

- Duomenų užklausėjo užklausa turi būti apdorota greičiau nei per 50ms.
- Duomenų užklausėjo užklausų dažnumas negali visiškai sutrikdyti visos sistemos darbo.
- Visa duomenų bazė turi būti prieinama per REST užklausas.

2. 8. 3 Reikalavimai sistemos priežiūrai

- Sistemos priežiūrai reikalingas žmogus.
- Nuolatinis klaidų žurnalo tikrinimas ir klaidų šaltinio identifikavimas.
- Duomenų bazės kopijos yra daromos kasdien.

2. 8. 4 Reikalavimai saugumui

- Jei norima, kad pašaliniai klientai negalėtų prieiti prie sistemos, sistemą reikia paleisti vietiniame tinkle, ir išorinį prisijungimą apriboti ugniasiene. Sistema turi galimybę nustatyti, kuris IP adresas, kokius veiksmus gali atlikti sistemoje.
- Visos REST užklausos yra saugomos užklausų žurnale.
- Sistema niekaip nėra susijusi su nuotolinių duomenų bazių saugumu.
- Konfigūraciniai failai neturi būti prieinami kitiems operacinės sistemos vartotojams, kadangi konfigūraciniuose failuose yra nurodyti nuotolinės duomenų bazės slaptažodžiai.

2. 9 Egzistuojantys sprendimai

2. 9. 1 Galimos naudoti sistemos

- Sistemos kūrimui pasinaudota python2.7 programavimo kalba.
- Sistemos kūrimui paspartinti, realizuojant HTTP serverį panaudotas python modulis BaseHTTPServer.
- Sistemos kūrimo procese, prisijungimui prie nuotolinės duomenų bazės palengvinti gali būti panaudotas MySQLdb python modulis.
- MySqlAdmin-4.0.7 nuotolinių duomenų bazių priežiūrai, realizavimui ir administravimui.

2. 9. 2 Galimi panaudoti komponentai

- *Iceweazel 24.4.0* naršyklė, programos testavimui.
- *Curl 7.26.0* naršyklė, programos testavimui.
- *MySQL* arba *MariaDB* duomenų bazė.

2. 10 Pritaikymas

Vartotojas jau i egzistuojančias duomenų bazes gali įkelti **.sql* atsarginės kopijos failą duomenų atstatymui. Failo įkėlimas vyksta ne per programą.

2. 11 Galimos sistemos kūrimo rizikos

47 lentelė. Sistemos kūrimo rizikos

Rizikos faktorius	**Tikimybinis įvertinimas, %**
Esant daugiau kaip 1000 projektų nepadidinus serverio resursų gali sulėtėti kuriamos sistemos darbas.	10
Reikalavimų specifikacijos pasikeitimai realizavimo fazėje.	6

2. 12 Vartotojo dokumentacija ir apmokymas

- Administratoriai, prižiūrinės šią sistemą bus supažindinti su visa sistemos architektūra, supažindinti su programos konfigūravimo galimybėmis.
- Klientams, kurie naudosis šia programa, bus paruošta REST serverio API dokumentacija.

2. 13 Perspektyviniai reikalavimai

- Programos diegimui gali būti paruoštas *deb* paketas, kuris palengvins programos diegimą Debian pagrindu grįstose operacinėse sistemose.
- Programos praplečiamumui palengvinti, programa bus rašoma Python programavimo kalba.

3. SISTEMOS ARCHITEKTŪROS SPECIFIKACIJA

3. 1 Architektūros specifikacijos paskirtis

IS architektūros specifikacijos paskirtis – pateikti ir išanalizuoti sistemos komponentus. Aprašyti kas vyksta programos paleidimo metu, aprašyti būsenų diagramas, veiklos diagramas.

3. 2 Architektūros pateikimas

Nuotolinės duomenų bazės valdymo sistemos architektūros specifikacijoje pateikėme sistemos architektūrą iš skirtingų požiūrio taškų. Statinį sistemos vaizdą nusako klasių diagrama, dinaminį vaizdą perteikia būsenų ir sekų diagramos.

3. 3 Architektūros apribojimai

Kuriant šią sistemą, buvo atsižvelgta ir numatyta, jog ji veiks tik Unix šeimos operacinėse sistemose. Konkrečiau -- sistema buvo kuriama *Debian Wheezy* operacinėje sistemoje, ir tai gali turėti įtakos sistemos darbui, jei jin veiks kitoje operacinėje sistemoje. Ypatingai daug dėmesio programos paleidimo, ir veikimo metu gali reikalauti Windows tipo operacinės sistemos paruošimas.

Numatomi ir esantys sistemos architektūriniai apribojimai:

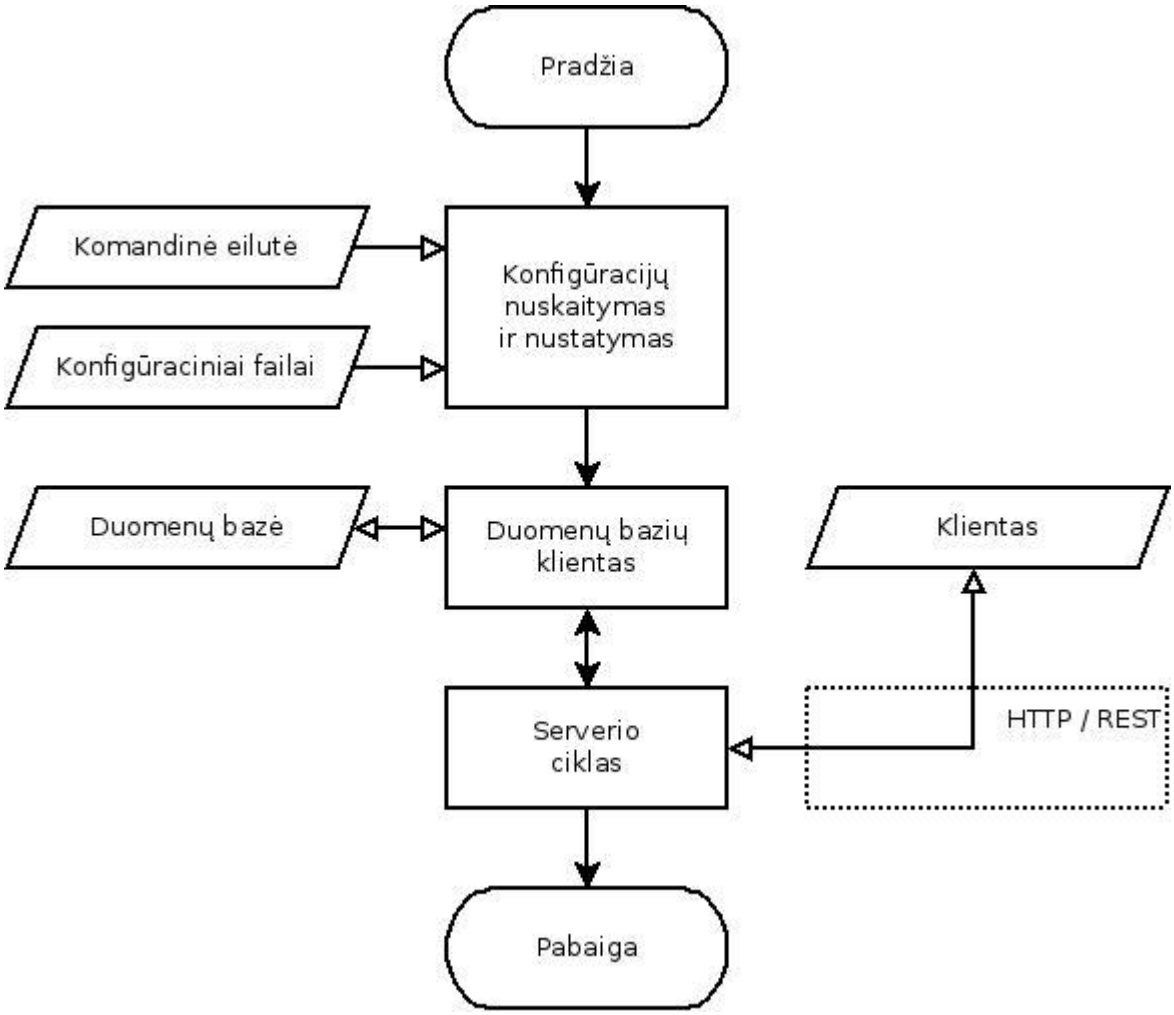
- *Python2.7* interpretatorius. Kadangi naujesnėse python interpretatoriaus versijose, atsirado skirtumų tarp kai kurių duomenų struktūrų, programa neveiks, jei bus naudojamas kitoks python iterpretatorius.
- *MySQLdb* python paketas. Tam kad sistema galėtų bendradarbiauti su nutolusiomis duomenų bazėmis, būtina kad sistemoje būtų įdiegtas šis paketas. Šio paketo versijos įtakos neturi, nes jis yra palaikomas tiesiogiai MySQL kūrėjų komandos.
- Sistema turi būti paleista vartotojaus, turinčio teises kurti ir skaityti failus. Jei programos aplanke nebus sukurti *hosts.conf* ir *pymyrestrc.conf* failai, programa juos bandys sukurti pati. Tačiau, jei šie failai jau yra sukurti, reikia įsitikinti, kad vartotojas, kuris paleidžia programą, turi teises skaityti šiuos failus.
- Jei programa bus diegiama Debian pagrindu grįstose operacinėse sistemose, būtina, kad sistemoje būtų įdiegta *apt* paketų tvarkyklė, ir įjungtos *main* ir *non-free* repozitorijos. Tik tada *apt* paketų tvarkyklė galės įdiegti *MySQLdb* paketą.

3. 4 Sistemos dinaminis vaizdas

3. 4. 1 Sistemos pagrindinės būsenos

Iš pateiktos diagramos, matome, kad programos gyvavimo ciklas susideda iš penkių žingsnių. Pirmoji būsena - "pradžia". Joje vartotojas paleidžia programą. Po paleidimo programa pereina į antrąją būseną - "Konfigūracijų nuskaitymas ir nustatymas". Šioje būsenoje programa gauna informaciją apie konfigūracijas, apibūdinančias programos būsimą darbą iš komandinės eilutės ir konfigūracinių failų. Po nurodytų konfigūracijų nustatymo programa pereina į "duomenų bazių klientas" būseną. Šioje būsenoje, programa prisijunginėja prie visų konfigūraciniuose failuose nurodytų nuotolinių duomenų bazių. Į šią būseną programa galės grįžti iš "Serverio ciklo" būsenos, kai reikės išgauti informaciją iš nutolusios duomenų bazės. Po prisijungimo prie nutolusių duomenų bazių

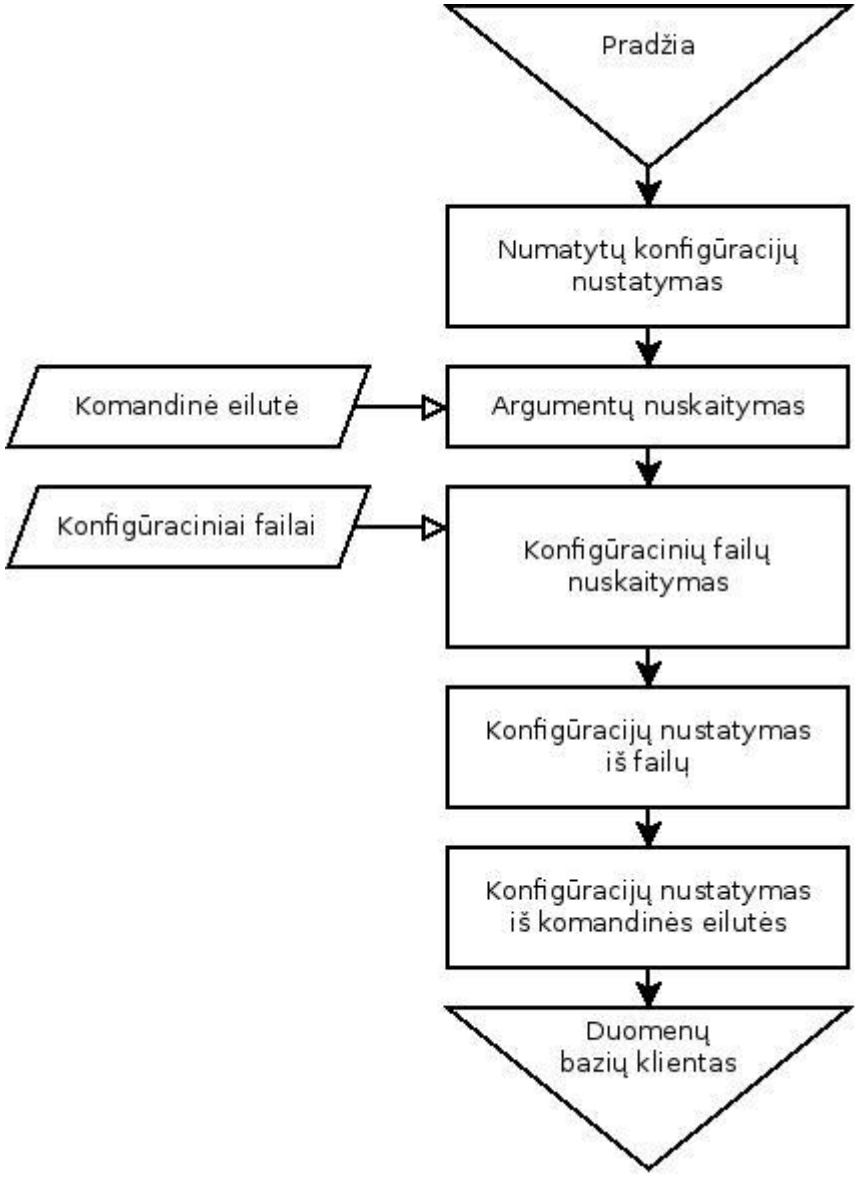
sistema pereina į "serverio ciklas" būseną, kurioje veikia amžinas ciklas, tol kol jį nutraukia vartotojas, arba kita programa. Šioje būsenoje, programa sukuria HTTP serverį ir laukia klientų užklausų. Tik šioje būsenoje programa supranta ir gali apdoroti REST užklausas. Kai programa gauna užklausą iš kliento, pirmiausiai yra patikrinama ar ji yra supranta programai. Jei ji suprantama ir reikalauja duomenų išgavimo iš kažkurios nutolusios duomenų bazės, sistema pereina į būseną "duomenų bazių klientas" ir atlieka nuotolinėje duomenų bazėje nurodytą užklausą. Gavusi užklausos rezultatus, programa grįžta į "serverio ciklas" būseną, ir išsiunčia rezultatus JSON formatu klientui. Programa kartoja šiuos veiksmus tol, kol nenutraukiamas jos darbas ir ji nepereina į būseną "Pabaiga".



8 pav. *Sistemos būsenos ir jas įtakojantys faktoriai*

Šioje diagramoje yra detalizuojama būsenų diagramoje aprašyta būsena "Konfigūracijų nuskaitymas ir nustatymas" ir aprašoma konfigūracijų gyvavimo eiga. Pirmoje būsenoje "Numatytų konfigūracijų nustatymas" programa, nustato programos kode nurodytas konfigūracijas. Tai yra atliekama todėl, kad programos eigoje, jei nebus nurodyti kiti konfigūracijų šaltiniai (komandinė eilutėje arba konfigūracijos failai), arba bus nurodytos ne visos konfigūracijos reikalingos sistemos darbui, sistema jau turės nustatytus numatytuosius nustatymus ir galėtų sėkmingai dirbti. Po to, sistema pereina į kitą būseną, kurioje nuskaito programos paleidimo metu argumentuose nurodytus nustatymus, tačiau, jų dar nenaudoja. Sekantis programos veiksmas – "Konfigūracinių failų nuskaitymas", programa bando nuskaityti *hosts.conf* ir *pymyrestrc.conf* failus, ir juose esančius nustatymus.

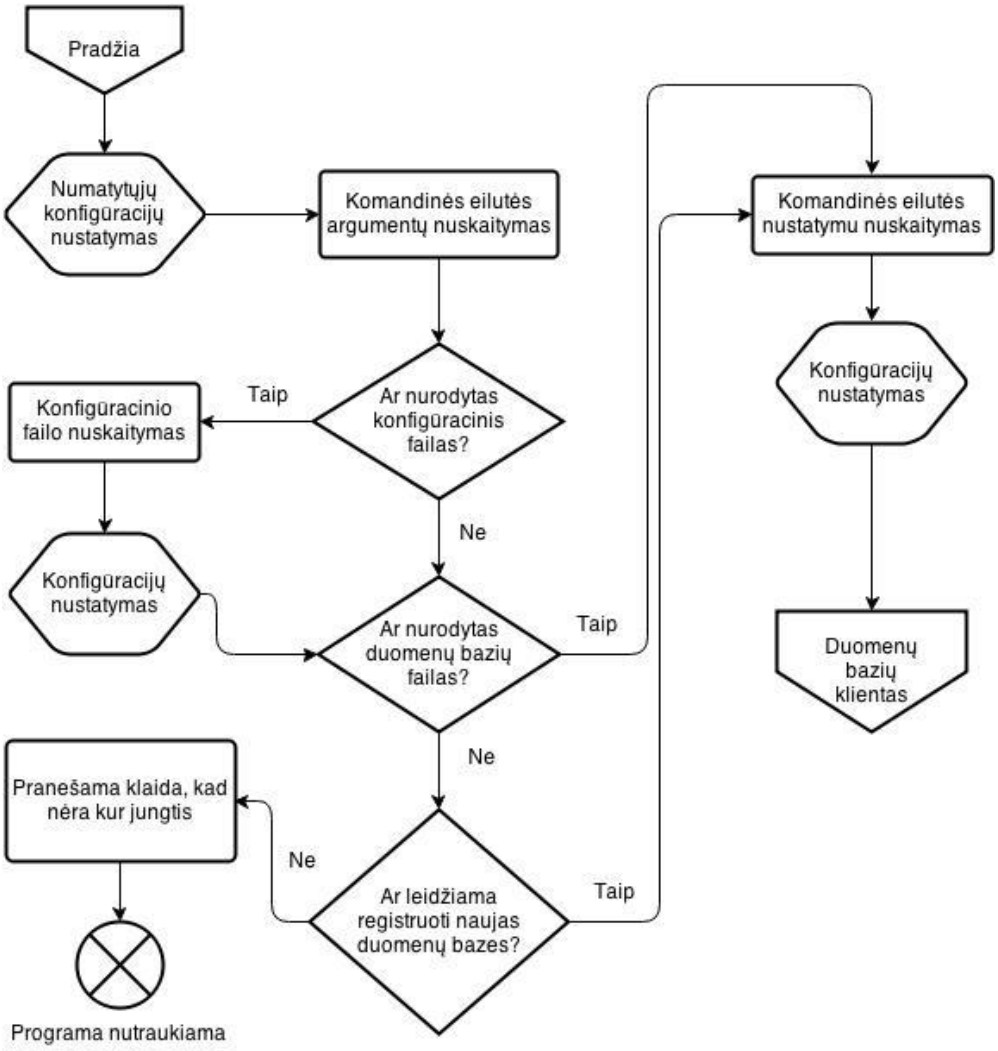
Sekančioje būsenoje programa nustato konfigūracijas, kurias rado failuose ir pereina į kitą būseną, kurioje konfigūracijas nustato iš programos paleidimo metu nurodytų argumentų. Toks konfigūracijų gyvavimo ciklas yra gyvybiškai svarbus programai, nes leidžia konfigūracijom suteikti prioritetus. Mažiausiai įtakos turės numatytieji nustatymai, kurie yra programoje, juos perrašys failuose nurodyti nustatymai, o šiuos -- argumentuose nurodyti nustatymai. Aukščiausią prioritetą turi argumentuose nurodyti nustatymai. Tai leidžia testuoti programą paleidimo metu, nekoreguojant konfigūracinių failų. Toliau programa tęsia darbą pereidama į "Duomenų bazių klientas" būseną.



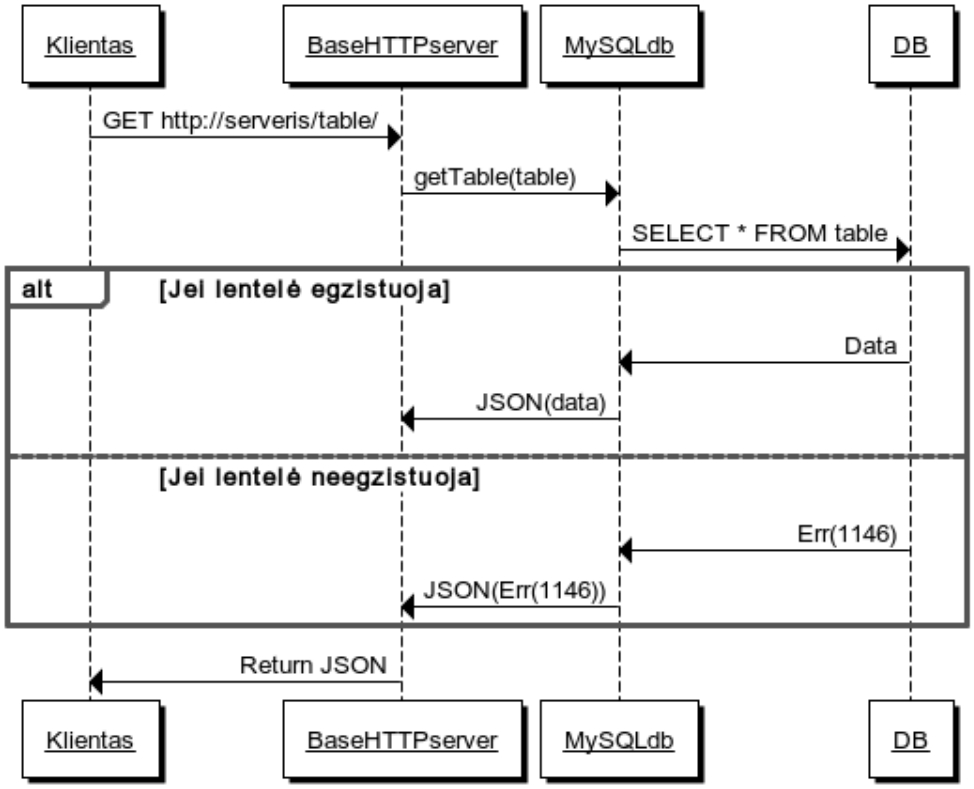
9 pav. *Sistemos konfigūracijų vykdymo eiga ir tai įtakojantys faktoriai*

3. 4. 2 Programos veikla

Diagramoje detalizuojama "Konfigūracijų nuskaitymas ir nustatymas" būsena ir parodomas konfigūracijų nuskaitymo ir nustatymo algoritmas, ir pavaizduojama, po kokių veiksmų programos būsena pereina į "Duomenų bazių klientas".

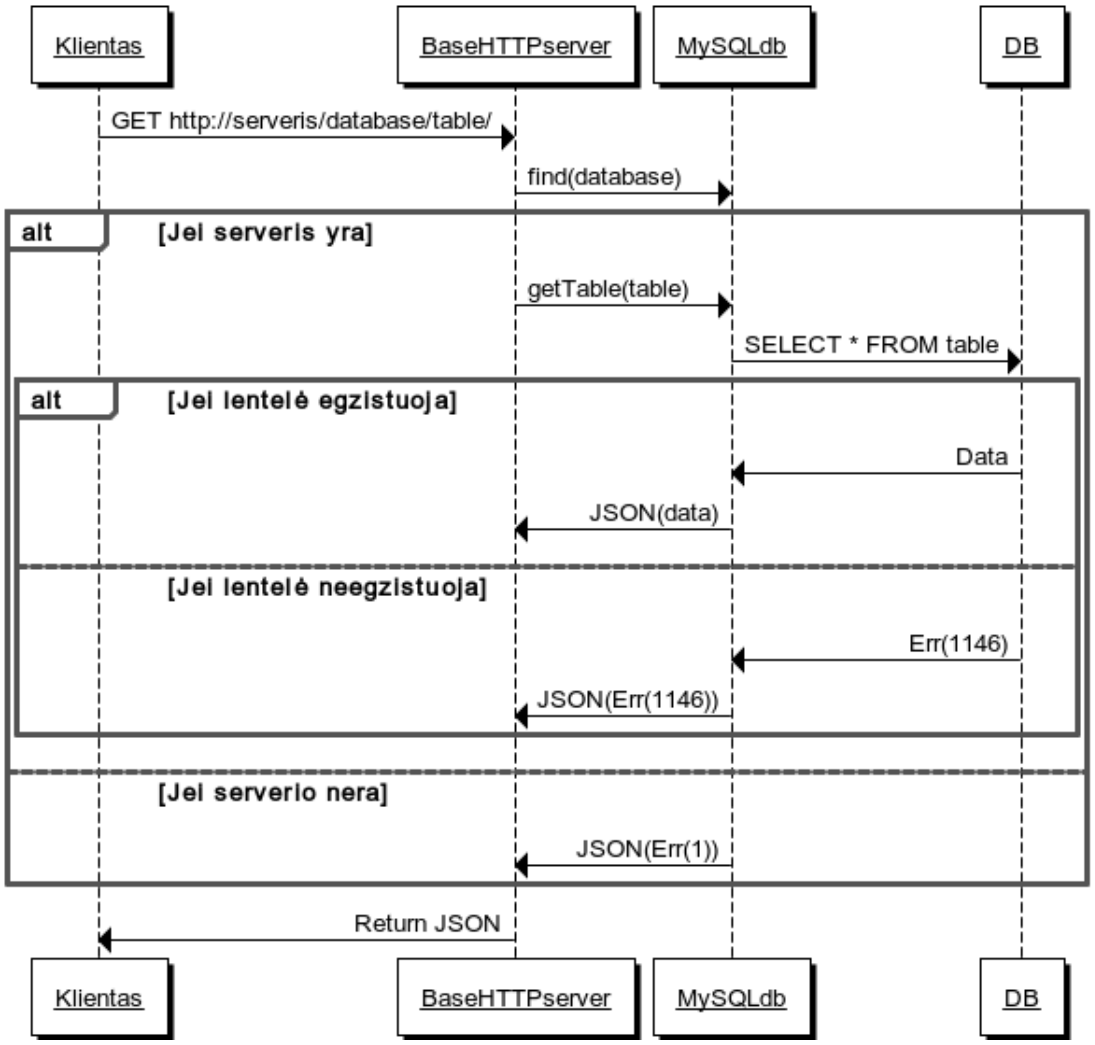


10 pav. *Sistemos konfigūracijų nuskaitymo algoritmas*



11 pav. *Lentelės turinio gavimas solo rėžime*

Sekų diagramoje "Lentelės turinio gavimas solo rėžime" atvaizduojamas sistemos darbas, kai programa apdoroja užklausą kuri reikalauja išgauti lentelės turinį *solo* rėžime. Pirmiausia klientas kreipiasi į serverį atlikdamas *GET http://serveris/table/* užklausą. Tada serveris iškviečia funkciją *getTable(table),* kuris sužadina *MySQLdb* modulį, kuris yra atsakingas už bendravimą su nutolusiomis duomenų bazėmis. *MySQLdb* modulis atlieka *SELECT * FROM table* užklausą nutolusioje duomenų bazėje, gautus rezultatus suformuluoja JSON formatu, ir perduoda *BaseHTTPserver* moduliui. *BaseHTTPserver* modulis atsakymą įrašo į *wfile* virtualų failą, kurio turinys yra perduodamas klientui kaip HTTP atsakymas. Šitaip klientas užklausos rezultatus gauna JSON formatu.



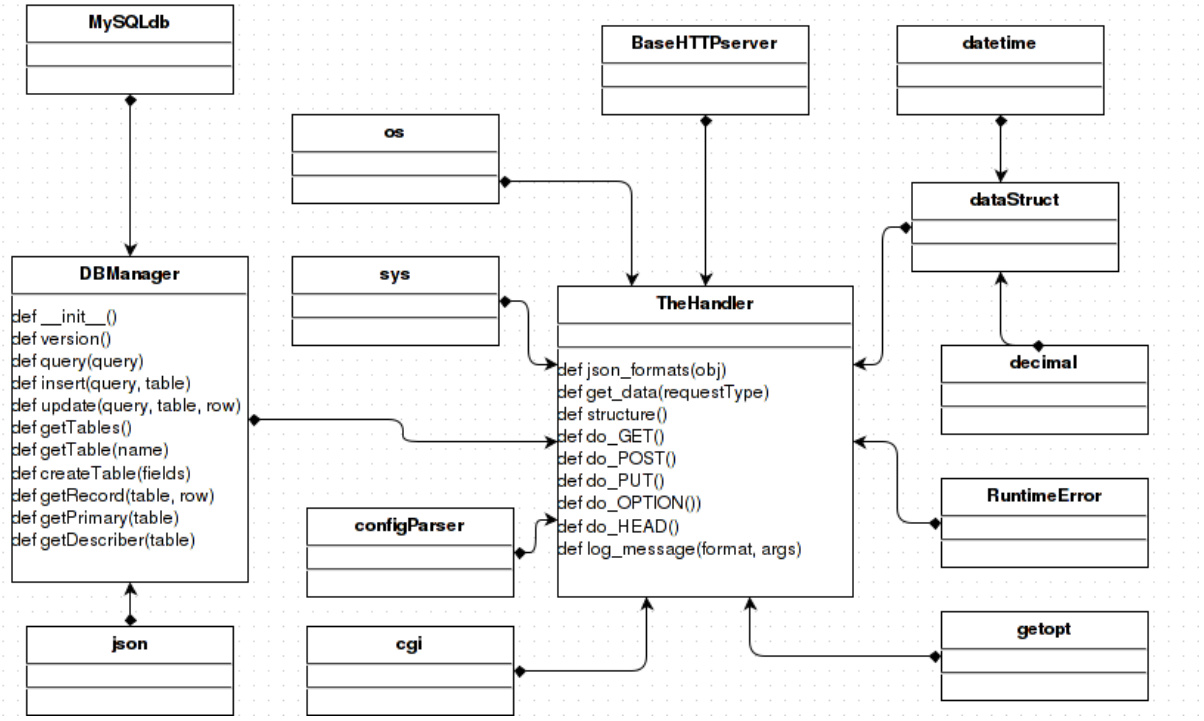
12 pav. *Lentelės turinio gavimas multi rėžime*

Adresas	http://	serveris	/	duomenųbazė	/	lentelė	/	raktas
Metodai	GET	Duomenų bazių sąrašo gavimas		Lentelių sąrašo gavimas		Lentelės įrašų gavimas		Įrašo gavimas
	POST	Duomenų bazės registravimas		Lentelės kūrimas		Įrašo kūrimas		Įrašo atnaujinimas
	PUT							Įrašo atnaujinimas

13 pav. *Metodų vykdymo logika*

Kurį metodą kada vykdyti yra nesprendžiama atsižvelgiant į url adresą ir užklausos perdavimo metodą. Struktūra, leidžianti įsivaizduoti kuris metodas kada yra kviečiamas yra pateikta 13 pav.

3. 5 Sistemos statinis vaizdas



14 pav. *Sistemos klasių diagrama*

48 lentelė. Klasės „DBManager(RuntimeError)“ detalizavimas

Klasė	DBManager(RuntimeError)
Apibrėžimas	Klasė naudojama sukurti duomenų bazės valdikliui, kuris yra nuolatos prisijungęs prie nuotolinės duomenų bazės. Ši klasė taip pat atlieka visus su duomenų baze susijusius veiksmus, tokius kaip naujos lentelės kūrimas, ar konkrečių duomenų nuskaitymas.
Struktūra	• init(username, password, hostname, database, name) -- klasės konstruktorius, kuris prijungia objektą prie nuotolinės duomenų bazės. Funkcijos argumentai: o username -- prisijungimo vartotojas, kuriuo reikia jungtis prie nuotolinės duomenų bazės. o password -- prijungiamo vartotojo slaptažodis. o database -- nuotolinės duomenų bazės vardas. Kai objektas prisijungs prie duomenų bazės jis automatiškai atlieka komandą *USE database*, kurioje nurodo šiame argumente nurodytos duomenų bazės vardą. o name -- nuotolinės duomenų bazės vardas sistemoje. Šis argumentas nurodo, kokiu vardu sistemoje bus saugomas prijungimas prie duomenų bazės. Argumente dažniausiai nurodoma reikšmė, kuri yra nurodyta *hosts.conf* ar kitame duomenų bazių konfigūraciniame faile. Reikšmė pagal nutylėjimą yra paimama kaip skilties pavadinimas, kuris apibūdina duomenų bazę. • def version(self) -- funkcija grąžina nuotolinės duomenų bazės versiją. Prilygsta duomenų bazėje *SELECT VERSION()* atliekamą komandą. • query(query) -- funkcija atlieka užklausa nuotolinėje duomenų bazėje. Rezultatus grąžina kaip *MySQLdb* duomenų masyvą, kuris yra sudarytas iš *list* tipo masyvo, kurio elementuose yra saugoma python kalboje naudojamas *dictionary* tipo masyvas. Funkcijos argumentai: o query -- tekstinio tipo kintamasis, kuriame nurodoma kokią SQL užklausą norima atlikti nuotolinėje duomenų bazėje. • insert(query, table) -- funkcija įterpia naujus duomenis į argumente nurodytą lentelę. Duomenų bazėje atitinka *INSERT INTO ...* tipo komandą. Funkcijos

	argumentai: ○ query -- tekstinio tipo kintamasis, kuriame nurodoma kokią SQL užklausą norima atlikti nuotolinėje duomenų bazėje reikės atlikti. ○ table -- nuotolinėje duomenų bazėje egzistuojančios lentelės vardas. • update(query, table, row) -- funkcija kuri atnaujina duomenis nuotolinėje duomenų bazėje. Funkcijos argumentai: ○ query -- tekstinio tipo kintamasis, kuriame nurodoma kokią SQL užklausą norima atlikti nuotolinėje duomenų bazėje. ○ table -- nuotolinėje duomenų bazėje egzistuojančios lentelės vardas. ○ row -- tekstinio tipo kintamasis, nurodantis lentelėje egzistuojantį pirminį raktą, pagal kurį bus identifikuojama eilutė, kurią reikia atnaujinti. • getTables() -- funkcija grąžina duomenų bazės lentelių sąrašą *list* tipo masyve. • getTable(name) -- funkcija grąžina visus lentelėje esančius įrašus kaip MySQLdb duomenų masyvą, kuris yra sudarytas iš *list* tipo masyvo, kurio elementuose yra *dictionary* tipo masyvai. Funkcijos argumentai: ○ name -- nuotolinėje duomenų bazėje egzistuojančios lentelės vardas. • createTable(name) -- funkcija sukuria nuotolinėje duomenų bazėje naują lentelę. Informaciją apie tai, kokią lentelę reikia kurti, funkcija pasiima iš kliento paduoto *POST* masyvo elementų. Kiekvieno *POST* elemento vardas bus traktuojamas kaip naujo stulpelio vardas, o elemento reikšmė apibūdins stulpelio informaciją. Pavyzdžiui *POST* masyve egzistuojantis elementas *user_id=int,pri,ai* bus traktuojamas kaip *user_id* stulpelio apibūdinimas, jog šis turi būti *int* tipo stulpelis, kuris yra pirminis raktas lentelėje (tai identifikuoja *pri* vėliavėlė), ir jog šis stulpelis yra savaime pasididinantis (tai parodo *ai* vėliavėlė)-- t.y kaskart įrašius naują reikšmę šioje lentelėje, stulpelio reikšmė padidės per vienetą. Funkcijos argumentai: ○ name -- naujai kuriamos lentelės vardas. • def getRecord(table, row) -- funkcija grąžina nuotolinėje duomenų bazėje įrašą, kaip *MySQLdb* duomenų masyvą, kuris yra sudarytas iš *list* tipo masyvo, kurio elementuose yra saugoma python kalboje naudojamas *dictionary* tipo masyvas. Funkcijos argumentai: ○ table -- nuotolinėje duomenų bazėje egzistuojančios lentelės vardas. ○ row -- nuotolinėje duomenų bazėje, lentelėje *table* egzistuojantis pirminis raktas, pagal kurį galima identifikuoti įrašą. • def getPrimary(table) -- funkcija grąžina lentelės kuri yra nurodyta funkcijos argumente stulpelio vardą, kuris yra naudojamas kaip pirminis raktas. Funkcijos argumentai: ○ table -- nurodomas lentelės pavadinimas, kurios lentelės pirminį raktą norime gauti. • def getDescriber(table) -- funkcija grąžina nuotolinėje duomenų bazėje lentelės aprašą, kaip *MySQLdb* duomenų masyvą, kuris yra sudarytas iš *list* tipo masyvo, kurio elementuose yra saugoma python kalboje naudojamas *dictionary* tipo masyvas. Funkcijos grąžinami rezultatai sutampa su nuotolinėje duomenų bazėje komandos *DESCRIBE table* grąžinamais rezultatais. Funkcijos argumentai: ○ table -- egzistuojančios lentelės kurios aprašą norime gauti pavadinimas.

49 lentelė. Klasės „BaseHTTPServer“ detalizavimas

Klasė	BaseHTTPServer
Apibrėžimas	Klasė atsakinga už HTTP serverio veikimą, ir užklausų apdorojimą.
Struktūra	Klasės kintamieji: • client_address -- saugomas python *tuple* tipo masyvas formatu (*host*, *port*), kurie nurodo kliento adresą. • server -- saugo SocketServer.TCPServer klasės objektą, kuris yra atsakingas už darbą tinklu. • command -- tekstinio tipo kintamasis, kuris saugo užklausos pavadinimą.

	Pavyzdžiui *GET*. • path -- tekstinio tipo kintamasis, kuris kliento užklaustos užklausos adresą. • request_version -- tekstinio tipo kintamasis kuris saugo užklausos versiją. Pavyzdžiui *HTTP/1.0* • headers -- masyvas saugoantis informaciją, kurią klientas atsiuntė *HEAD* dalyje. • rfile -- virtualus failas, kuriame yra saugoma visas kliento *HTTP* užklausos tekstas. • wfile -- virtualus failas, kuriame yra saugomas visas serverio *HTTP* užklausos atsakymas. • error_message_format -- tekstinio tipo kintamasis, kuris saugo klaidų pranešimo formatą. Pavydžiui *yyyy-mm-dd H:i:s \%e /n*, kur *yyyy* atitinka metus, *mm* mėnesį, *dd* dieną, *H* - 24 valandų formato valandą, *i* - minutes, *s* - sekundes, kada buvo užfiksuota klaida. Kintamasis *\%e* nurodo kurioje vietoje žinutėje turi būti rašoma klaida, vėliavėlė *\\n* nurodo eilutės pabaigą, ir naujos eilutės pradžią. • error_content_type -- tekstinio tipo kintamasis, kuris nurodo kokiu formatu serveris grąžins klaidą klientui. Šis kintamasis perrašo *HTTP Content-Type* kintamąjį. Pavyzdžiui *text/html*. • protocol_version -- tekstinio tipo kintamasis, kuris nurodo kokiu tipu serveris atsakinės klientui. perrašo *HTTP Content-Type* kintamąjį. Formatas nurodomas kaip mime tipo, pavyzdžiui *text/html*. Klasės funkcijos: • def handle() -- iškviečia *handle_one_request()* funkciją. O jei vienu metu yra keletą prisijungimų, funkciją iškviečia keletą kartų. • def handle_one_request() -- iškviečia funkciją, kurios pavadinimas sutampa su kliento *HTTP* užklausimo tipu. Pavyzdžiui jei klientas Kreipsis į serverį atlikdamas GET tipo užklausą, funkcija iškvies *do_GET* funkciją. • def send_error(code, message) -- klientui išsiunčia klaidos žinutę. Atsakymas bus tokio tipo, koks kintamasis buvo nurodytas *error_content_type* kintamajame. • def send_response(code, message) -- funkcija išsiunčia klientui viską, kas buvo įrašyta *wfile* virtualiame faile. • def send_header(keyword, value) -- funkcija įrašo į *wfile* virtualų failą, nurodyta kintamąjį. Funkcija automatiškai kintamuosius rašo *HEAD* atsakymo dalyje. Pvz. *Content-Type: 'text/html'*. • def end_headers() -- funkcija virtualiame *wfile* faile įrašo tuščią eilutę iškart po HEAD kintamųjų, kas leidžia naršyklei suprasti, kad dabar baigėsi HEAD dalis. • def log_request(code, message) -- išsaugo be klaidų apdorotų užklausų užklausimus. Unix šeimų operacinėse sistemose, atitinka *sys.stdout* buferį. • def log_error() -- išsaugo užklausų užklausimus, kuriuos apdorojant kilo klaidų. Unix šeimų operacinėse sistemose atitinka *sys.stderr* buferį. • def log_message() -- nustato išsaugo užklausų žinutes, kurias gavome apdorodami kliento užklausoje nurodytą funkciją. Unix šeimų operacinėse sistemose atitinka *stdout* buferį. • def address_string() -- grąžina kliento IP adresą tekstiniu formatu, kuris sužadino objektą.

50 lentelė. Klasės „ThisHandler(BaseHTTPServer.BaseHTTPRequestHandler)“ detalizavimas

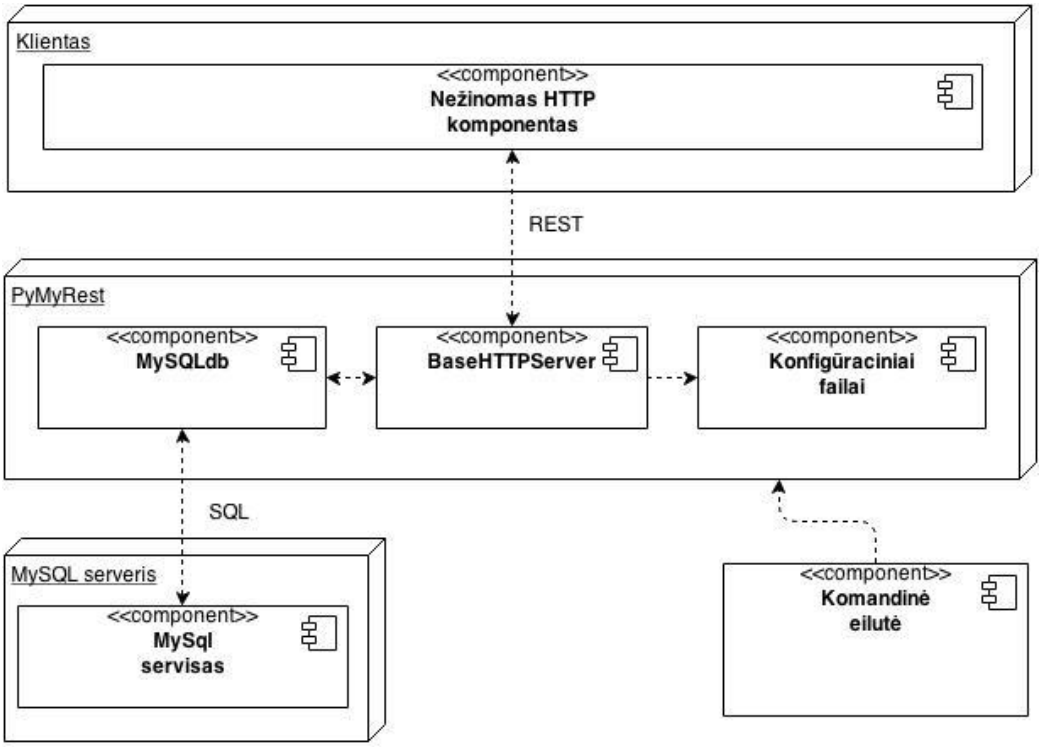
Klasė	ThisHandler(BaseHTTPServer.BaseHTTPRequestHandler)
Apibrėžimas	Tai pagrindinė sistemos klasė, apibūdinanti HTTP serverio darba.
Struktūra	• def json_formats(obj) -- funkcija konvertuoja *MySQLdb* kintamųjų masyvą į JSON tekstą. Paprastai, python kalboje slankaus kablelio formato skaitmenys turi žymę f, pavyzdžiui 23.4f todėl *JSON* tekstas juos gali interpretuoti ne kaip skaičių, o kaip tekstą. Ši funkcija pašalina specialias žymas nuo kintamųjų, ir grąžina JSON tekstą. Funkcijos argumentai: o obj -- *MySQLdb* rezultatai, kuriuos reikia konvertuoti į JSON tekstą.

	• def get_data(requestType) -- funkcija nuskaito virtualų *rwrite* failą, kuriame yra pateiktas visas kliento užklausos tekstas. Pagal tame tekste *HEAD* dalyje esančius kintamuosius, funkcija sukuria *dictionary* tipo kintamąjį, ir jį grąžina. Funkcijos argumentai: o requestTyle -- tekstinio tipo kintamasis, kuriame reikia nurodyti, kokio tipo užklausa buvo įvykdyta. Pavyzdžiui *GET* arba *POST*. Pagal nutylėjimą, funkcija ieškos *POST* tipo kintamųjų. • def structure() -- funkcija nustato kokio tipo užklausa klientas vykdė, ir nustato visus užklausoje esančius kintamuosius. • def do_GET() -- šią funkciją sužadina klientas atlikdamas *GET* užklausą. Pagal esama užklausos *url* funkcija atlieka loginius veiksmus, ir grąžina rezultatus į virtualų *wfile* failą, kurio tekstas būna nusiunčiamas klientui. • def do_POST() -- šią funkciją sužadina klientas atlikdamas *POST* užklausą. Pagal esama užklausos *url* funkcija atlieka loginius veiksmus, atlieka pakitimus duomenų bazėje ir grąžina rezultatus į virtualų *wfile* failą, kurio tekstas būna nusiunčiamas klientui. • def do_PUT() -- šią funkciją sužadina klientas atlikdamas *PUT* užklausą. Pagal esama užklausos *url* funkcija atlieka loginius veiksmus, atlieka pakitimus duomenų bazėje ir grąžina rezultatus į virtualų *wfile* failą, kurio tekstas būna nusiunčiamas lientui. • def do_OPTION() -- šią funkciją sužadina klientas atlikdamas OPTION užklausą. Pagal esama užklausos *url* funkcija atlieka loginius veiksmus, atlieka pakitimus duomenų bazėje ir grąžina rezultatus į virtualų *wfile* failą, kurio tekstas būna nusiunčiamas klientui. • def do_HEAD() -- šią funkciją sužadina klientas atlikdamas HEAD užklausą. Pagal esama užklausos *url* funkcija atlieka loginius veiksmus, atlieka pakitimus duomenų bazėje ir grąžina rezultatus į virtualų *wfile* failą, kurio tekstas būna nusiunčiamas klientui. • def log_message(format, *args) -- ši funkcija saugo užklausų istoriją, ir spausdina atliekamas užklausas ekrane. Unix šeimų operacinėse sistemose ši funkcija rezultatus grąžina į *stdout* failą.

51 lentelė. Klasės „Globalus kintamieji ir funkcijos“ detalizavimas

Klasė	Globalus kintamieji ir funkcijos
Apibrėžimas	Globalus kintamieji ir funkcijos, kurios yra pasiekiamos visoje sistemoje
Struktūra	• PORT -- darbinis programos portas. • HOSTNAME -- kompiuterio pavadinimas, kuriame dirbs programa. • CONFIG_FILE -- konfigūracinio failo adresas. • MODE -- programos darbinis rėžimas. • HOSTS_FILE -- duomenų bazių konfigūracinis failas. • MAIN_HOST -- pagrindinė duomenų bazė, jei programa dirba *solo* rėžime. • HOSTS -- visų duomenų bazių masyvas. • CONNECTIONS -- visų prijungtų duomenų bazių masyvas. • CONNECTIONS_INFO -- masyvas, kuriame saugoma informacija apie prisijungimą. • REGISTER_ALLOW -- *boolean* tipo kintamasis, kuris nurodo, ar programa priima naujas nuotolines duomenų bazes. • __main__ -- pagrindinė funkcija, kuri yra iškviečiama programos paleidimo metu. • main -- funkcija, kurioje vyksta pagrindinis ciklas, kai serveris klausosi kliento užklausų.

3. 6 Sistemos komponentai



15 pav. *Sistemos komponentų tarpusavio sąveika*

4. TESTAVIMAS

4. 1 Testavimo paskirtis

4. 1. 1 Testavimo tikslai

Atlikdami sistemos testavimą, siekiame išsiaiškinti likusias klaidas, saugumo spragas, bei įvertinti funkcinį sistemos išpildymą.

Atliekant testavimą siekiama:

- Testuoti duomenų įrašymo validacijos tikrinimą.
- Testuoti rezultatų grąžinimą.
- Testuoti rezultatų įrašymą.

4. 1. 2 Testavimo resursai

Serverio programinė įranga:

- MySQL 5.5.31
- Python2.7
- Operacinė sistema: Debian Wheezy 64bit.
- Operacinės sistemos branduolys: 2.6.32-042stab081.8.

Serverio aparatinė įranga:

- Serveris veikia VPS pagrindu, todėl serverio resursus testavimo metu nustatyti sudėtiga.

Kliento programinė įranga:

- Operacinė sistema: Linux UBUNTU 13.04 64bit.
- Testavimo programa: curl 7.26.0

Kliento aparatinė įranga:

- Procesorius: AMD Vision A6.
- Operatyvioji atmintis: 4 GB.

4. 2 Sistemos funkcijų testavimas

4. 2. 1 Konfigūracijų nuskaitymas iš failo

52 lentelė. Konfigūracinio failo nuskaitymo testavimo rezultatai

Testo ID	Reikalavimai/ tikslai	Įvykis/įvestis	Laukiamas rezultatas	Gautas rezultatas	T/N
1.	Numatytojo konfigūracinio failo nuskaitymas	Programa paleidžiama kai konfigūracinis failas nenurodomas	Programa turėtų nuskaityti *pymyrestrc.conf* failą.	Programa nuskaitė *pymyrestrc.conf* failą	T
2.	Nuskaityti nurodytą darbinį portą konfigūraciniame faile.	Paleidžiant programą nurodomas konfigūracinis failas	Programa iš nurodyto konfigūracinio failo, programa nuskaito jame nurodytą darbinį portą.	Programa iš nurodyto konfigūracinio failo, programa nuskaitė jame nurodytą darbinį portą	T
3.	Nuskaityti nurodytą darbinį rėžimą	Paleidžiant programą	Programa iš nurodyto konfigūracinio failo,	Programa iš nurodyto konfigūracinio failo,	T

	konfigūraciniame faile.	nurodomas konfigūracinis failas.	programa nuskaito jame nurodytą darbinį rėžimą	programa nuskaito jame nurodytą darbinį rėžimą	
4.	Nuskaityti registracijos leidimą konfigūraciniame faile	Paleidžiant programą nurodomas konfigūracinis failas	Programa iš nurodyto konfigūracinio failo, nuskaito jame nurodytą registracijos leidimą	Programa iš nurodyto konfigūracinio failo, nuskaito jame nurodytą registracijos leidimą	T

4. 2. 2 Konfigūracijų nuskaitymo iš argumentų testavimas

53 lentelė. Vartotojų registracijos testavimas

Testo ID	**Reikalavimai/tikslai**	**Įvykis/įvestis**	**Laukiamas rezultatas**	**Gautas rezultatas**	**T/N**
5.	Darbinio porto nustatymas iš komandinės eilutės pasinaudojant *-p* nustatymu.	Programa paleidžiama kai konfigūracinis failas nenurodomas, tačiau nurodomas darbinis programos portas argumentuose, pasinaudojant *-p* nustatymu.	Programa turėtų nuskaityti ir naudoti po *-p* nustatymo nurodytą darbinį portą.	Programa nuskaitė po *-p* nustatymo nurodytą darbinį portą.	T
6.	Darbinio porto nustatymas iš komandinės eilutės pasinaudojant *--port* nustatymu.	Programa paleidžiama kai konfigūracinis failas nenurodomas, tačiau nurodomas darbinis programos portas argumentuose, pasinaudojant *--port* nustatymu.	Programa turėtų nuskaityti ir naudoti po *--port* nustatymo nurodytą darbinį portą.	Programa nuskaitė po *--port* nustatymo nurodytą darbinį portą.	T
7.	Konfigūracinio failo nustatymas iš komandinės eilutės pasinaudojant *-c* nustatymu.	Paleidžiant programą argumentuose nurodomas konfigūracinis failas esantis po *-c* nustatymu.	Programa turėtų nuskaityti konfigūracinį failą nurodytą po *-c* nustatymo.	Programa nuskaitė konfigūracinį failą nurodytą po *-c* nustatymo.	T
8.	Konfigūracinio failo nustatymas iš komandinės eilutės pasinaudojant *--config* nustatymu.	Paleidžiant programą argumentuose nurodomas konfigūracinis failas esantis po *--config* nustatymu.	Programa turėtų nuskaityti konfigūracinį failą nurodytą po *--config* nustatymo.	Programa nuskaitė konfigūracinį failą nurodytą po *--config* nustatymo.	T
9.	Neegzistuojančio konfigūracinio failo nustatymas iš komandinės eilutės pasinaudojant *-c*	Paleidžiant programą argumentuose nurodomas neegzistuojantis	Programa turėtų naudoti numatytuosius nustatymus	Programa naudoja numatytuosius nustatymus.	T

	nustatymu.	konfigūracinis failas esantis po *-c* nustatymu.			
10.	Neegzistuojančio konfigūracinio failo nustatymas iš komandinės eilutės pasinaudojant *--config* nustatymu.	Paleidžiant programą argumentuose nurodomas neegzistuojantis konfigūracinis failas esantis po *--config* nustatymu.	Programa turėtų naudoti numatytuosius nustatymus	Programa naudoja numatytuosius nustatymus.	T
11.	Nuotolinių duomenų bazių konfigūracinio failo nustatymas iš komandinės eilutės pasinaudojant *-H* nustatymu.	Paleidžiant programą argumentuose nurodomas nutolusių duomenų bazių konfigūracinis failas esantis po *-H* nustatymu.	Programa turėtų nuskaityti nutolusių duomenų bazių konfigūracinį failą nurodytą po *-H* nustatymo.	Programa nuskaitė nutolusių duomenų bazių konfigūracinį failą nurodytą po *-H* nustatymo.	T
12.	Neegzistuojančio nuotolinių duomenų bazių konfigūracinio failo nustatymas iš komandinės eilutės pasinaudojant *-H* nustatymu.	Paleidžiant programą argumentuose nurodomas neegzistuojantis nutolusių duomenų bazių konfigūracinis failas esantis po *-H* nustatymu.	Programa turėtų nuskaityti numatytąjį *hosts.conf* failą.	Programa nuskaito numatytąjį *hosts.conf* failą.	T
13.	Nuotolinių duomenų bazių konfigūracinio failo nustatymas iš komandinės eilutės pasinaudojant *--host* nustatymu.	Paleidžiant programą argumentuose nurodomas nutolusių duomenų bazių konfigūracinis failas esantis po *--host* nustatymu.	Programa turėtų nuskaityti nutolusių duomenų bazių konfigūracinį failą nurodytą po *--host* nustatymo.	Programa nuskaitė nutolusių duomenų bazių konfigūracinį failą nurodytą po *--host* nustatymo.	T
14.	Neegzistuojančio nuotolinių duomenų bazių konfigūracinio failo nustatymas iš komandinės eilutės pasinaudojant *--host* nustatymu.	Paleidžiant argumentuose nurodomas neegzistuojantis nutolusių duomenų bazių konfigūracinis failas esantis po *--host* nustatymu.	Programa turėtų nuskaityti numatytąjį *hosts.conf* failą.	Programa nuskaito numatytąjį *hosts.conf* failą.	T
15.	Pagrindinės nutolusios duomenų bazės nustatymas iš komandinės eilutės pasinaudojant *-M* nustatymu.	Paleidžiant programą nustatoma pagrindinė nutolusi duomenų bazė nurodyta po *-M* nustatymu.	Programa turėtų nuskaityti pagrindinę nutolusią duomenų bazę nurodytą po *-M* nustatymu.	Programa nuskaitė pagrindinę nutolusią duomenų bazę nurodytą po *-M* nustatymu.	T

16.	Neegzistuojančios pagrindinės nutolusios duomenų bazės nustatymas iš komandinės eilutės pasinaudojant *-M* nustatymu.	Paleidžiant programą bandoma nustatyti neegzistuojanti pagrindinė nutolusi duomenų bazė nurodyta po *-M* nustatymu.	Programa turėtų neveikti.	Programa neveikė.	T
17.	Pagrindinės nutolusios duomenų bazės nustatymas iš komandinės eilutės pasinaudojant *--main* nustatymu.	Paleidžiant programą nustatoma pagrindinė nutolusi duomenų bazė nurodyta po *--main* nustatymu.	Programa turėtų nuskaityti pagrindinę nutolusią duomenų bazę nurodytą po *--main* nustatymu.	Programa nuskaitė pagrindinę nutolusią duomenų bazę nurodytą po *--main* nustatymu.	T
18.	Neegzistuojančios pagrindinės nutolusios duomenų bazės nustatymas iš komandinės eilutės pasinaudojant *--main* nustatymu.	Paleidžiant bandoma nustatyti neegzistuojanti pagrindinė nutolusi duomenų bazė nurodyta po *--main* nustatymu.	Programa turėtų neveikti.	Programa neveikė.	T
19.	*Multi* rėžimo nustatymas iš komandinės eilutės pasinaudojant *-m* nustatymu.	Norima programą paleisti *multi* darbo rėžime, nurodant nustatymą *-m*.	Programa tūrėtų pradėti darbą *multi* rėžime.	Programa pradėjo darbą *multi* rėžime.	T
20.	*Multi* rėžimo nustatymas iš komandinės eilutės pasinaudojant *--multi* nustatymu.	Norima programą paleisti *multi* darbo rėžime, nurodant nustatymą *--multi*.	Programa tūrėtų pradėti darbą *multi* rėžime.	Programa pradėjo darbą *multi* rėžime.	T
21.	*Solo* rėžimo nustatymas iš komandinės eilutės pasinaudojant *-s* nustatymu.	Norima programą paleisti *solo* darbo rėžime, nurodant nustatymą *-s*.	Programa tūrėtų pradėti darbą *solo* rėžime.	Programa pradėjo darbą *solo* rėžime.	T
22.	*Solo* rėžimo nustatymas iš komandinės eilutės pasinaudojant *-solo* nustatymu.	Norima programą paleisti *solo* darbo rėžime, nurodant nustatymą *-solo*.	Programa tūrėtų pradėti darbą *solo* rėžime.	Programa pradėjo darbą *solo* rėžime.	T
23.	Naujų nutolusių duomenų bazių registravimo įjungimas pasinaudojant *-R* nustatymu.	Programos paleidimo metu norima nustatyti jog, programa leistų registruoti naujas nutolusias duomenų bazes.	Programos paleidimo metu nurodžius *-R* nustatymą programa turi leisti registruoti naujas nutolusias duomenų bazes.	Programa leidžia registruoti naujas nutolusias duomenų bazes.	T
24.	Naujų nutolusių duomenų bazių registravimo įjungimas pasinaudojant *--register* nustatymu.	Programos paleidimo metu norima nustatyti jog, programa leistų registruoti naujas nutolusias	Programos paleidimo metu nurodžius *--register* nustatymą programa turi leisti registruoti naujas nutolusias duomenų	Programa leidžia registruoti naujas nutolusias duomenų bazes.	T

		duomenų bazes.	bazes.		
25.	Nurodymas jog naujas, programos darbo metu užregistruotas nutolusias duomenų bazes įrašyti į konfigūracinį failą pasinaudojant *-a* nustatymu.	Norima, kad programos darbo metu, užregistruotą naują nutolusią duomenų bazę, programa ją įrašytų į konfigūracinį failą.	Programos paleidimo metu nurodžius *-a* nustatymą, programos darbo metu, naujai užregistruotos nuotolinės duomenų bazės yra įrašomos į nutolusių duomenų bazių konfigūracinį failą.	Programos paleidimo metu nurodžius *-a* nustatymą, programos darbo metu, naujai užregistruotos nuotolinės duomenų bazės yra įrašomos į nutolusių duomenų bazių konfigūracinį failą.	T
26.	Nurodymas jog naujas, programos darbo metu užregistruotas nutolusias duomenų bazes įrašyti į konfigūracinį failą pasinaudojant *--append* nustatymu.	ją įrašytų į konfigūracinį failą.	Programos paleidimo metu nurodžius *--append* nustatymą, programos darbo metu, naujai užregistruotos nuotolinės duomenų bazės yra įrašomos į nutolusių duomenų bazių konfigūracinį failą.	Programos paleidimo metu nurodžius *--append* nustatymą, programos darbo metu, naujai užregistruotos nuotolinės duomenų bazės yra įrašomos į nutolusių duomenų bazių konfigūracinį failą.	T

4. 2. 3 REST užklausų apdorojimo testavimas

54 lentelė. Vartotojų registracijos testavimas

Testo ID	**Reikalavimai/tikslai**	**Įvykis/įvestis**	**Laukiamas rezultatas**	**Gautas rezultatas**	**T/N**
27.	Nuotolinių duomenų bazių sąrašo išgavimas.	Atliekama HTTP *GET* užklausa adresu *http://serveris/*	Programa tūrėtų grąžinti visą nutolusių duomenų bazių sąrašą.	Programa grąžina visas šiuo metu prijungtas nutolusias duomenų bazes.	T
28.	Nuotolinės duomenų bazės lentelių sąrašo išgavimas.	Atliekama HTTP *GET* užklausa adresu *http://serveris/ bazeA/*	Programa tūrėtų grąžinti visas nutolinės duomenų bazės lenteles.	Programa grąžino visas nutolinės duomenų bazės lenteles.	T
29.	Neegzistuojančios nutolusios duomenų bazės lentelių sąrašo išgavimas	Atliekama HTTP *GET* užklausa adresu *http://serveris/ bazeB/*	Programa tūrėtų pranešti apie klaidą.	Programa pranešė apie klaidą	T
30.	Nuotolinės duomenų bazės lentelės turinio išgavimas.	Atliekama HTTP *GET* užklausa adresu *http://serveris/ bazeA/ lenteleA*	Programa tūrėtų grąžinti visą nutolusios duomenų bazės lentelės turinį.	Programa grąžino visą nutolusios duomenų bazės lentelės turinį.	T
31.	Nuotolinės duomenų bazės, neegzistuojančios lentelės turinio išgavimas	Atliekama HTTP *GET* užklausa adresu *http://serveris/ bazeA/ lenteleB*	Programa tūrėtų pranešti apie klaidą.	Programa pranešė apie klaidą.&	T
32.	Nuotolinės duomenų bazės įrašo išgavimas iš	Atliekama HTTP *GET* užklausa adresu	Programa tūrėtų grąžinti visą informaciją apie įrašą.	Programa grąžino visą informaciją apie įrašą.	T

	egzistuojančios lentelės.	*http://serveris/ bazeA/ lenteleA/ 1*			
33.	Nuotolinės duomenų bazės įrašo išgavimas iš neegzistuojančios lentelės.	Atliekama HTTP *GET* užklausa adresu *http://serveris/ bazeA/ lenteleB/ 1*	Programa tūrėtų nieko negrąžinti.	Programa nieko negrąžino.	T
34.	Nuotolinių duomenų bazių sąrašo išgavimas.	Atliekama HTTP *GET* užklausa adresu *http://serveris/*	Programa tūrėtų grąžinti visą nutolusių duomenų bazių sąrašą.	Programa grąžina visas šiuo metu prijungtas nutolusias duomenų bazes	T
35.	Nuotolinės duomenų bazės lentelių sąrašo išgavimas.	Atliekama HTTP *GET* užklausa adresu *http://serveris/ bazeA/*	Programa tūrėtų grąžinti visas nutolinės duomenų bazės lenteles.	Programa grąžino visas nutolinės duomenų bazės lenteles	T
36.	Neegzistuojančios nutolusios duomenų bazės lentelių sąrašo išgavimas.	Atliekama HTTP *GET* užklausa adresu *http://serveris/ bazeB/*	Programa tūrėtų pranešti apie klaidą.	Programa pranešė apie klaidą	T
37.	Nuotolinės duomenų bazės lentelės turinio išgavimas.	Atliekama HTTP *GET* užklausa adresu *http://serveris/ bazeA/ lenteleA*	Programa tūrėtų grąžinti visą nutolusios duomenų bazės lentelės turinį.	Programa grąžino visą nutolusios duomenų bazės lentelės turinį.	T
38.	Nuotolinės duomenų bazės, neegzistuojančios lentelės turinio išgavimas	Atliekama HTTP *GET* užklausa adresu *http://serveris/ bazeA/ lenteleB*	Programa tūrėtų pranešti apie klaidą.	Programa pranešė apie klaidą	T
39.	Nuotolinės duomenų bazės įrašo išgavimas iš egzistuojančios lentelės.	Atliekama HTTP *GET* užklausa adresu *http://serveris/ bazeA/ lenteleA/ 1*	Programa tūrėtų grąžinti visą informaciją apie įrašą.	Programa grąžino visą informaciją apie įrašą.	T
40.	Nuotolinės duomenų bazės įrašo išgavimas iš neegzistuojančios lentelės.	Atliekama HTTP *GET* užklausa adresu *http://serveris/ bazeA/ lenteleB/ 1*	Programa tūrėtų nieko negrąžinti.	Programa nieko negrąžino.	T

4. 3 Našumo testavimas

Našumo testavimui buvo įvertinta kiek laiko trunka sistemos kodo vykdymas, reikalingas SQL užklausos suformavimui ir gautų rezultatų pateikimui JSON formatu. Laikas, kuris sugaištamas gauti atsakymui iš DBVS nėra skaičiuojamas, nes jis būtų tapatus ir tiesioginio kreipimosi į DBVS metu.

55 lentelė. Sistemos našumo rezultatai

	REST užklausos apdorojimas	**Duomenų grąžinimas JSON formatu**	**Bendras pymyrest sistemos vėlinimas**
Duomenų bazių sąrašo gavimas	1ms	1ms	2ms
Lentelių sąrašo	4ms	1ms	5ms

gavimas			
Įrašo išgavimas	3ms	1ms	4ms
Lentelės kūrimas	3ms	1ms	4ms
Įrašo kūrimas	6ms	1ms	7ms
Įrašo atnaujinimas	6ms	1ms	7ms
Lentelės šalinimas	1ms	1ms	2ms
Įrašo šalinimas	1ms	1ms	2ms

4. 4 Testavimo išvados

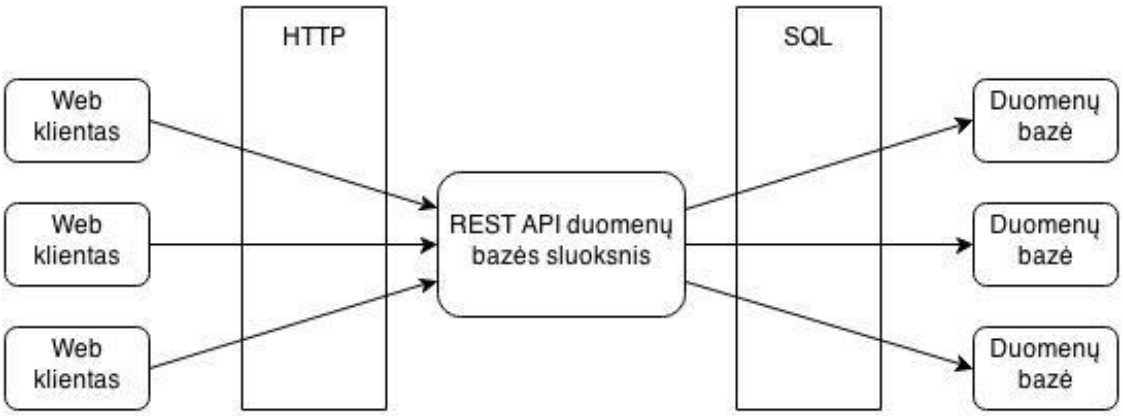
Atlikus sistemos funkcionalumo testavimą sistemos veikimo netikslumų nepastebėta.

Našumo testavimo metu pastebėta, kad daugiausiai laiko yra sugaištama formuojant INSERT tipo užklausas. Tačiau net ir šių užklausų vykdymo laikas nėra kritiškas, nes pati sistema užklausos sugeneravimui ir rezultatų grąžinimui sugaišta tik *7ms*.

5. VARTOTOJO DOKUMENTACIJA

5. 1 Vartotojo vadovo paskirtis

Sistema skirta palengvinti duomenų sinchronizavimą tarp kelių nutolusių sistemų. Sistema priims REST užklausas, atitinkamai nuo užklausos sistema atliks SQL užklausą nutolusioje duomenų bazėje, ir rezultatus klientui grąžins JSON formatu. Tad programa veiks kaip savotiškas REST / SQL vertėjas.



16 pav. *Sistemos veikimo teorinis modelis*

Pateikiama programos pritaikymo schema. Programa vienu metu gali palaikyti keletą klientų ir keletą nutolusių duomenų bazių. Programa gali būti pritaikyta bet kur, kur reikia susieti kelias nutolusias duomenų bazes iš vieno taško, arba kur duomenims iš duomenų bazės išgauti, negali būti panaudota SQL kalba.

5. 2 Sistemos diegimas

Programą galima įdiegti Debian pagrindu grįstose sistemose pasinaudojant kompakte pradėtu paketu. Tai galima atlikti pasinaudojant komanda:

```
dpkg -i pymyrest.deb
```

17 pav.

Būtina pastebėti, kad programa teisingai įsidiegs tik *Debian Wheezy* operacinėje sistemoje. Kitose Debian pagrindu grįstose operacinėse sistemose tokiose kaip *Ubuntu* ar *Knoppix,* programa gali neįsidiegti dėl kitaip besivadinančių programos funkcionalumui reikalingų programinių paketų nei *Debian Wheezy* operacinėje sistemoje. Taip pat pat (Debian pagrindu grįstų operacinių sistemų programa, atsakinga už paketų diegimą) programoje turi būti įjungtos *main* ir *non-free* repozitorijos.

Jei programos diegimas pasinaudojant *dpkg* komanda nepavyksta, programą galima įdiegti rankiniu būdu. Tačiau reikai įsitikinti kad sistemoje įra įdiegti *python2.7* ir *python-mysqldb* paketai. Programa turi vieną pagrindinį failą *--pymyrest.py*, kurį reikia nukopijuoti į pasirinktą katalogą. Katalogui turi būti suteikiamos skaitymo ir rašymo teisės vartotojui, kuriuo bus paleidžiama sistema. Taip pat rekomenduojama tame pačiame aplanke sukurti tuščius *pymyrestrc.conf* ir *hosts.conf* failus, kuriuose bus nurodytos programos konfigūracijos.

5. 3 Sistemos paleidimas ir konfigūravimas

Programą Unix šeimų operacinėse sistemose galima paleisti pasinaudojus viena iš keletą komandų:

- *python pymyrest* -- programa paleidžiama tiesiogiai besinaudojant python interpretatoriumi.
- *./pymyrest* -- programą galima paleisti, kaip vykdomąjį failą, tašiau prieš tai jam turi būti suteiktos *execution* teisės, pasinaudojus komanda *chown +x pymyrest*.
- *pymyrest* -- jei programa buvo įdiegta Debian pagrindu grįstoje operacinėje sistemoje.

Paleidimo metu, programa bandys nuskaityti *hosts.conf* ir *pymyrestrc.conf* failus, todėl reikia įsitikinti kad šie failai yra sukurti ir jiems yra suteiktos skaitymo ir rašymo teisės. Failus galima sukurti panaudojus komanda *touch pymyrestrc.conf hosts.conf*, o rašymo ir skaitymo teises galima suteikti pasinaudojus *chmod +rw hosts.conf pymyrestrc.conf* komanda. Sistema po paleidimo turės nusistačiusius numatytuosius nustatymus. Tai yra, programa naudos 8222 darbinį portą, programa dirbs *Multi* darbiniame rėžime, programos konfigūracinis failas bus *pymyrestrc.conf*, o nuotolinių duomenų bazių konfigūracinis failas bus *hosts.conf*.Šiuos parametrus galima pakeisti programos paleidimo metu, pasinaudojus komandinės eilutės argumentais, arba koreguojant konfigūracinius failus.

Galimi komandinės eilutės argumentai:

- *-p <port> / --port=<port>* -- nurodyti kurį darbinį portą naudoti programai. pagal nutylėjimą, jei portas bus nenurodytas, programa naudos 8222 darbinį portą.
- *-c <file> / --config=<file>* -- nurodyti kurį konfigūracinį failą naudoti programai. jei konfigūracinis failas nenurodytas, programa bandys nuskaityti *pymyrestrc.conf* failą. Jei jo neras, programa naudos pagal nutylėjimą programoje nurodytus parametrus.
- *-H <file> / --hosts=<file>* -- nurodyti kurį nutolusių duomenų bazių konfigūracinį failą naudoti, kuriame yra prisijungimų duomenys i domenų bazes. Jei failas bus nenurodytas, programa
- pagal nutylėjimą bandys nuskaityti *hosts.conf* failą. Jei šio failo nebus, arba jis bus tuščias, programa neveiks ir išsijunks. Tačiau yra išimtis, jei tokio failo nėra, arba jis yra tuščias, o programos rėžimas bus nustatytas kaip *multi,* ir bus leidžiama registruoti naujas nutolusias duomenų bazes, programa veiks.
- *-m / --multi arba -s / --solo* -- čia nurodomas programos darbinis rėžimas. Programa gali dirbti vienu iš dviejų rėžimų -- *solo* arba *multi*. *Solo* rėžime dirbdama programa, vienu metu gali būti prisijungusi tik prie vienos duomenų bazės, o *Multi* rėžime, programa gali prisijungi prie kelių duomenų bazių iškarto. Tai turi įtakos formuojant *url* REST užklausas. Pavyzdžiui *solo* rėžime, po *GET http://serveris/table* galima iškarto nurodyti lentelės pavadinimą, o *multi* rėžime, užklausoje reikės nurodyti ir duomenų bazės prisijungimo vardą -- *GET http://serveris/database/table*.
- *-R / --register* -- šį parametrą galima nurodyti tik tada, jei programa dirba *multi* rėžime. Jis įgalina, programos darbo metu užregistruoti naujas nutolusias duomenų bazes.
- *-A / --append* -- šis parametras įtakos turės tik tada, jei programos darbinis rėžimas bus *multi* ir programai bus leidžiama registruoti naujas nutolusias duomenų bazes. Šio

nustatymo reikšmė yra tokia, kad programa užregistravus naują duomenų bazės prisijungimą, jį įrašytų į *hosts.conf*, ar kitą nurodyta duomenų bazių nustatymų failą. Tai turės įtakos kitam programos paleidimo momentui, nes darbinio metu naujai užregistruotos duomenų bazės niekur yra niekur neišsaugomos, tad programą paleidus per naujo, programa pakartotinai prie jų nebeprisijungs.

- *-M <host> / --main <host>* -- šis nustatymas nurodo pagrindinę nutolusią duomenų bazę, ir įtakos turi tik *solo* darbiniame rėžime. Paprastai programą paleidus *solo* rėžime, programa dirbs tik su viena duomenų baze, tačiau nežinos, kurią duomenų bazę naudoti. Todėl programą paleidžiant *solo* rėžimu, šį parametrą būtina nurodyti.

Programos darbui reikia dviejų konfigūracinių failų -- *pymyrestrc.conf* ir *hosts.conf*. *pymyrestrc.conf* faile yra nurodomos programos konfigūracijos, o *hosts.conf* faile nurodomos nuotolinės duomenų bazės, jų adresai ir prisijungimo informacija. Pavyzdinis *pymyrestrc.conf* failo turinys:

```
[Server]
ServerPort = 8111
ServerDebug=True
ServerMode=Multi
AllowRegistration=True
```

18 pav. *pymyrestrc.conf failo užpildymo pavyzdys*

Šiame konfigūraciniame faile nurodomi parametrai:

- *ServerPort* -- nurodomas darbinis programos portas, kurio klausysis programa išorinių klientų užklausų. Šiuo atvėju nurodytas *8111*.
- *ServerDebug* -- nurodomas testavimo rėžimas. Paprastai jis būna išjungtas (False). Šiuo atveju jis yra įjungtas (True).
- *ServerMode* -- nurodomas programos darbinis rėžimas. Programa gali dirbti vienu iš dviejų rėžimų -- *multi* ir *solo*. *Multi* rėžimas leidžia programai būti prisijungus prie kelių duomenų bazių. Solo rėžime, programa dirba tik su viena duomenų baze.
- *AllowRegistration* -- nurodoma ar programai leisti užregistruoti naujus projektus darbo metu. Šiuo metu yra įjungta (True). Norint išjungti reikia nurodyti *False*.

Pavyzdinis *hosts.conf* failas:

```
[MAIN]
Hostname=localhost
Database=main
Username=root
Password=root

[MAIN2]
Hostname=localhost
Database=test
Username=root
Password=root
```

19 pav. *hosts.conf failo pavyzdys*

Šiame faile nurodomos dvi duomenų bazės. Kadangi *pymyrestrc.conf* faile programos darbinis rėžimas pavyzdyje buvo nurodytas *Multi*, programa prisijungs prie abiejų duomenų bazių iš karto. Failuose nurodomi parametrai:

- *MAIN* ir *MAIN2* -- nurodo prisijungimų vardus. Per šiuos pavadinimus bus galima pasiekti šias duomenų bazes. Pavyzdžiui *http://serveris/MAIN/* ir *http://serveris/MAIN2/*. Būtina atkreipti dėmesį į tai, jog pavadinimuose didžiosios ir mažosios raidės yra interpretuojamos skirtingai.
- *Hostname* -- nurodomas duomenų bazės adresas. Gali būti nurodomas ir DNS adresas ir IP adresas.
- *Database* -- nurodoma prie kurios duomenų bazės jungtis serveryje.
- *Username* -- nurodomas prisijungimo vardas.
- *Password* -- nurodomas prisijungimo slaptažodis.

IŠVADOS

1. Susipažinus su REST architektūra ir sistemomis, turinčiomis REST architektūra paremtus API galima pastebėti, kad tokia galimybė dažnai yra kuriama didelėms, paslaugas teikiančioms sistemoms. Tuo tarpu egzistuojančios duomenų bazių valdymo sistemos pagal nutylėjimą dažnai REST architektūros nepalaiko. Tad naudojant įrankius, skirtus REST architektūros palaikymui būtų galima naudoti duomenų bazę, kaip apslaugą kitoms sistemoms ar vartotojams.
2. Sistemos reikalavimų ir architektūros specifikacijos aprašytos naudojant Volere reikalavimų šablono elementus. Tai turėtų padidinti šių specifikacijų suvokimą skirtingo lygio vartotojams, nes šiose specifikacijose yra naudojamos ne tik UML diagramos, bet ir jų aprašai.
3. Realizuota sistema palaiko visas pagrindines operacijas su duomenų bazės duomenimis. Joje nėra palaikomos tik specifinės operacijos, tokios kaip vartotojų valdymas, ar procedūrų kūrimas. Tačiau tai padaryta sąmoningai, nes vartotojams leidžiant laisvai naudotis visomis duomenų bazių valdymo sistemos funkcijomis, galimas netinkamas jų panaudojimas, išjungiant visą duomenų bazių valdymo sistemą, pašalinant jos vartotojus ir pan.
4. Atliktas sistemos testavimas parodė, kad visos funkcijos veikia korektiškai, o sistemos pridedamas papildomas vykdymo laikas neturi didelės įtakos duomenų atrankos ir manipuliavimo operacijoms. Iš to galima teigti, kad sistema yra tinkama naudojimui, bei neturėtų sukelti papildomų problemų.

LITERATŪRA

1. restSQL Overview [žiūrėta 2014-04-28]. Prieiga per internetą <http:// restsql.org/ doc/ Overview.html>
2. Concepts [žiūrėta 2014-04-28]. Prieiga per internetą <http:// restsql.org /doc/Concepts.html>
3. HTTP Interface [žiūrėta 2014-04-28]. Prieiga per internetą <http:// docs.mongodb.org/ ecosystem/ tools/http-interfaces/>
4. HTTP Interface [žiūrėta 2014-04-28]. Prieiga per internetą <http:// www.slashdb.com/ solutions/overcome-data-silos/>
5. Tech - Why is Database Layer as a REST API not common [žiūrėta 2014-04-28]. Prieiga per internetą <http://www.swaroopch.com/2013/04/18/database-as-rest-api/>
6. REST-Style Web API [žiūrėta 2014-04-28]. Prieiga per internetą <http://exist-db.org/ exist/apps /doc/ devguide_rest.xml>
7. What Are RESTful Web Services? [žiūrėta 2014-04-28]. Prieiga per internetą <http:// docs.oracle.com/ javaee/6/tutorial/doc/gijqy.html>
8. Tesla Model S REST API [žiūrėta 2014-04-28]. Prieiga per internetą <http://docs. timdorr.apiary. io/>
9. REST API v1.1 Resources [žiūrėta 2014-04-28]. Prieiga per internetą <https:// dev.twitter .com/docs/ api/1.1>
10. REST API [žiūrėta 2014-04-28]. Prieiga per internetą <http:// docs.aws.amazon.com/ AmazonS3/ latest/ API/ APIRest.html>
11. Atom (standard) [žiūrėta 2014-04-28]. Prieiga per internetą <http://en.wikipedia.org/ wiki/ Atom_standard>
12. Common Object Request Broker Architecture [žiūrėta 2014-04-28]. Prieiga per internetą <http:// en.wikipedia.org/ wiki/ Common_Object_Request_Broker_Architecture>

TERMINŲ IR SANTRUMPŲ ŽODYNĖLIS

SQL - Structured Query Language - struktūrizuota užklausų kalba.

REST - Representational state transfer architektūrinis stilius duomenims perduoti, pasinaudojant HTTP protokolu.

IE – Internet Exploder naršyklė.

POSIX - standartų rinkinys, skirtas apibrėžti API taikomųjų programų suderinamumui tarp įvairių UNIX operacinių sistemų variantų užtikrinti.

IP adresas - kompiuterio identifikatorius IP tinkluose.

Domenas - nuolatinis nekeičiamas kompiuterio vardas internete.

REGEXP – reguliari išraiška.

CSS - CSS (angl. Cascading Style Sheets) – kalba, skirta nusakyti kita struktūrine kalba aprašyto dokumento vaizdavimą.

API - application programming interface (API) nurodo kaip vienos programos turėtų bendrauti su kitomis.

AI – Auto Increment trumpinys. Automatiškai pasididinantis.

KISS - „Keep it simple, stupid“ trumpinys.

Slave – Nuo kito serverio priklausantis serveris.

Boolean – Kintamojo tipas galintis turėti tik „Teisinga“ arba „Neteisinga“ tipo reikšmes.

Int – Kintamojo tipas, galintis turėti bet kokią natūralaus skaičiaus reikšmę.

Varchar – Kintamojo tipas, galintis turėti bet kokią reikšmę, kurią galima išreikšti simbolių eilute.

W3C – (World Wide Web Consortium) konsorciumas leidžiantis programinės įrangos standartus („rekomendacijas“, kaip jie jas vadina) žiniatinkliui (angl. World Wide Web)

stderr - Virtualus failas, kuriame programos spauzdina darbo metu užregistruotas klaidas.

stdout - Virtualus failas, kuriame programos spauzdina darbo metu sugeneruotą tekstą.

stdin - Virtualus failas, iš kurio programos darbo metu nuskaito informaciją.

SOAP - angl. Simple Object Access Protocol.

RPC - angl. Remote procedure call.

CORBA - angl. Common Object Request Broker Architecture.

JSON - angl. JavaScript Object Notation.