VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
KOMPIUTERIJOS KATEDRA

Baigiamasis magistro darbas

Interaktyvus WEB Sistemos Kūrimas

Atliko: 2 kurso, 1 grupės studentas
Andrius Dvilevičius (parašas)

Darbo vadovas:
lekt. Linas Būtėnas (parašas)

Vilnius
2011

Turinys

Įvadas

Nuo 1993-1994 metų, kai atsirado Internetas, toks kokį mes jį žinome dabar, jis tapo bene svarbiausia ir greičiausiai besiplečiančia bendravimo, verslo plėtojimo ir reklamos sritimi. Vis daugiau daiktų ir paslaugų yra perkeliama į virtualią erdvę.

Temos aktualumas. Sparčiai plintant šiuolaikinėms technologijoms ir kylant kiekvieno gyventojo išsilavinimo ir kompiuterinio raštingumo lygiui, natūralu, kad vis daugiau sudėtingų procesų gali atlikti patys vartotojai. Dažniausiai vienintelė problema trukdanti tai padaryti yra paprastų įrankių stoka.

Internetinės parduotuvės ar elektroninės užsakymų sistemos jau senai nustojo stebinti vartotojus. Siauros paskirties ar vienai sričiai pritaikytos aplikacijos skirtos personaliniam vartojimui yra dažnai sutinkamos internete, bet sistemų supaprastinimas, jų pritaikymas visokioms situacijoms ir procesų automatizavimas leistu vartotojui lengvai tvarkyti vieną iš sudėtingesnių procesų ir tai būtų žingsnis link interneto supaprastinimo ir turinio gerinimo.

Nors rezervavimo sistemos yra dažnos internete, universalios sistemos, skirtos skirtingų rezervacijos situacijų įgyvendinimui nėra. Aplikacijos dažniausiai skirtos vienai iš rezervacijos reikalaujančių situacijų, pavyzdžiui grožio salono, įgyvendinimui, ir nėra pritaikomos kitoms situacijoms.

Dar viena didelė problema yra tai, kad daugelis aplikacijų yra pritaikyta personaliniam vartojimui, o tai reiškia, kad prie jų tiesiogiai gali prieiti tik tos aplikacijos vartotojas, o ne vartotojas norintis rezervuoti laiką.

Mokslinė problema. Atlikus nedidelę internetinių rezervavimo sistemų užsakovų apklausą paaiškėjo, kad 80% respondentų norėtų pamėginti rezervavimo sistemą susikurti ir susikonfigūruoti patys prieš kreipiantis į tokių sistemų kūrimo profesionalus, o esant patenkinamiems rezultatams, net 60% tokia sistema naudotųsi ir vėliau. Tik 20% respondentų visai nesinaudotų universalia sistema ir kreiptųsi į profesionalus iškart. Ši atlikta apklausa atskleidė esamą universalios rezervavimo sistemos potencialą ir galimas problemas, nusprendus pasinaudoti šia sistema savo reikmių įgyvendinimui.

Šiame darbe buvo apžvelgti teoriniai pagrindai reikalingi sudėtingos sistemos supaprastinimui ir pritaikymui paprastam vartotojui. Išnagrinėti teoriniai diegimo ir konfigūravimo, objektų kūrimo, duomenų bazių modeliai. Taip pat buvo apžvelgtos ir išanalizuoto realios situacijos ir galimi jų realizavimo būdai.

Siekiant tikslo buvo aprašytos ir išanalizuotos trys situacijos reikalaujančios rezervacijos, įgyvendinta grožio salono situacija su pilnai veikiančiomis rezervacijos ir administravimo funkcijomis. Trumpai aprašyti sistemos veikimo principai. Surinkta daug informacijos padėsiančios sudaryti galutinę, veikiančią sistemą.

Atliekant šį darbą bus nagrinėjami šie pagrindiniai klausimai:

1. Kokiais būdais įmanoma įgyvendinti rezervacijos ir tvarkaraščių sistemą ir kuris iš tų būdų yra tinkamiausias?
2. Kaip tūrėtų atrodyti sistemos diegimas ir konfigūravimas?
3. Kokio tipo objektų reikės sistemai?
4. Kokia yra objektų struktūra?
5. Kokia turi būti duomenų bazės struktūra?
6. Kokie algoritmai gali būti pritaikyti tvarkaraščių sudarymui?
7. Kaip pritaikyti minimalaus jungaus medžio paieškos algoritmus tvarkaraščių sudarymui?

Darbo tikslas – detaliai išanalizuoti galimus rezervacijos bei tvarkaraščių sudarymo sistemos įgyvendinimo, diegimo ir administravimo būdus. Remiantis informacija iš mokslinių šaltinių, bei duomenimis surinktais iš autorinio tyrimo sudaryti galutinės sistemos pagrindo planą ir surinkti kuo daugiau informacijos padėsiančios toliau plėtoti ir tobulinti sistemą.

Darbo uždaviniai:

1. Naudojantis moksline literatūra ir atliekant tyrimus surasti optimaliausią sistemos įgyvendinimo būdą;
2. Atlikus tyrimą surasti tinkamą, paprastą sistemos diegimo procesą, bei duomenų ir informacijos šalinimo schemą, diegimą nutraukus;
3. Išanalizuoti teorines situacijas ir identifikuoti sistemai reikalingus objektų tipus;
4. Tiriant situacijas sudaryti pradinę rezervacijos sistemos schemą, padedančią struktūrizuoti galutinį sistemos modelį;
5. Nagrinėjant naujausią programinę įrangą išrinkti tinkamiausią duomenų bazių valdymo įrankį;
6. Išanalizavus mokslinius šaltinius pritaikyti „Bin-packing“ ir minimalaus jungaus medžio algoritmus laiko tvarkaraščių sudarymui ir atlikti tyrimus su praktiniais duomenimis;
7. Dalies ar viso konfigūravimo ir administravimo etapų praktinis įgyvendinimas;

Tyrimo metodai. Darbe buvo panaudoti sekantys tyrimo metodai:

1. Mokslinės literatūros analizė kurios metu buvo surinkta ir susisteminta medžiaga apie galimus sistemos realizavimo būdus, algoritmų ir jų modifikacijų pritaikymą tvarkaraščių sudarymui, duomenų bazių modelio kūrimą ir kitų iškeltų uždavinių sprendimo būdus;
2. Teorinis algoritmų pritaikymas laiko rezervavimui ir praktinis jų įgyvendinimas kurio tikslas buvo išanalizuoti ir rasti labiausiai tinkantį algoritmą;

3. „CodeIgniter“ karkasas leidęs sistemos įgyvendinimui panaudoti „Model-View-Controller“ (MVC) architektūrą, leidusią išskirti vartotojo sąsają, duomenų apdorojimą ir bendravimą su duomenų bazėmis, taip labai palengvinusią atskirų šių modulių testavimą;

Darbo struktūra. Darbas sudarytas iš dviejų pagrindinių dalių. Pirmoji dalis yra skirta WEB sistemos sampratos išaiškinimui ir supažindinimui su būdais kuriais galima ją įgyvendinti. Taip pat, labai trumpai, paminimi populiariausi įrankiai skirti specifinio įgyvendinimo būdo realizacijai. Toliau labai detaliai nagrinėjama sistemos diegimo specifika ir proceso reikalavimai, bei pateikiami pasiūlymai šiai procedūrai supaprastinti. Toliau darbas tęsiamas detaliu teorinio objektų kūrimo modelio tyrimu į kurį įeina teorinių pavyzdžių analizė, reikiamų objektų struktūros sudarymas, duomenų bazių tipų palyginimas ir galutinės konfigūravimo schemos suformulavimas. Galiausiai pirmoji dalis yra užbaigiama teorinės bazės sudarymu apie algoritmus kurie gali būti pritaikyti tvarkaraščių generavimui. Antroje dalyje yra pristatomi autoriniai tyrimai, supažindinama su atliktų tyrimų metodika, tikslais ir aprašomi tyrimų rezultatai. Pateikiami ir aprašomi tyrimams naudoti algoritmai, bei aprašomos autoriaus atliktos modifikacijos skirtos algoritmų pritaikymui reikiamoms problemoms spręsti. Aprašomos galimos modifikacijos papildomiems algoritmams. Galiausiai pateikiamos išvados ir pasiūlymai, bei naudotos literatūros sąrašas.

Anotacija

Sparčiai plintant informacinėms technologijoms vis daugiau daiktų ir paslaugų yra perkeliama į virtualią erdvę. Šis darbas nagrinėja galimybę padaryti universaliu rezervacijos procesą. Pagrindinis tikslas sukurti paprastą sistemą, kurią skirtingoms situacijoms galėtų susikonfigūruoti pats vartotojas. Viena iš sudėtingiausių funkcijų kurią nagrinėja darbas yra galimybė sistemai generuoti automatinius tvarkaraščius. Sistemos praktiškumo būtinybė reikalavo gana plataus analizės spektro: sistemos įgyvendinimo principo tyrimo, programinės įrangos parinkimo, rezervavimo algoritmo sukūrimo, tvarkaraščių sudarymo algoritmo išvedimo. Galutinis rezultatas yra ištirti ir aprašyti konfigūravimo, administravimo bei rezervavimo procesai, jų tyrimų metodai ir gauti rezultatai.

Interactive WEB System Development

Andrius Dvilevičius

Paper for the Master's degree

Vilnius University, Faculty of Mathematics and Informatics, Department of Computer Science

Supervisor – lek. L. Būtėnas

Vilnius, 2011

Summary

With rapid spread of information technologies, more and more things and services are transferred to the virtual space. Online stores or online reservation systems has long ceased to surprise customers.

The main purpose of this master thesis is to analyze and find the way to universalize online reservation process using information gathered from scientific literature analysis and from author's research.

The main objective is to create a model of a simple system for the different situations that could be configured by administrator. Since most of internet users have great computer skills and new IT technologies are becoming more sophisticated and easier to use, most of processes could be performed by ordinary user with no IT education with use of simplified tools.

One of the most complex parts of this thesis is to examine the possibility of the system to generate automated schedules. Usability of the system called for the need of a relatively broad spectrum of analysis: the research of various ways of implementation of system, software selection, the creation of a reservation algorithm, scheduling algorithm modification and analysis.

In my work I analyze theoretical fundamentals needed for creation of sophisticated reservation system. I review installation and configuration, reservation, object creation and other processes. Also I examine real situations and possible their implementation.

The final result of my thesis is detailed information about preferred implementation way of the system and software needed for it, clear schema of the installation process, detailed an easy to use plan of configuration process, comprehensive information about administration and reservation of objects. Also practical implementation of some of the system modules and test results of these modules are included.

1. WEB Sistema

WEB sistemą galima įgyvendinti dviem būdais:

1. WEB aplikacija. Aplikacija, kurios klientas būtų „Thin“ tipo. Labai patogus įgyvendinimas, todėl, kad vietoj atskiros kliento aplikacijos užtektų tik interneto naršyklės. Tai labai supaprastina problemos sprendimą, nes:
   - Nereikia rašyti atskirų klientų skirtingoms operacinėms sistemoms
   - Vartotojui nereikia ieškoti ir siųstis kliento aplikacijos
   - Atnaujinus sistemą klientui nieko daryti nereikia, pakeitimus jis mato iškart
   - Saugumo atžvilgiu saugu, nes yra tik viena silpnoji grandis, kurią reikia apsaugoti – serveris
2. WEB servisas. Aplikacija, kurios klientas būtų „Fat“ tipo. Įgyvendinimas sudėtingesnis, tačiau turi keletą pliusų:
   - Serverio reikalavimai mažesnis
   - Galimybė dirbti neprisijungus prie interneto
   - Labai efektyviai dirba su aplikacijomis kurioms reikia didelių kompiuterio resursų ar spartaus interneto ryšio, pvz. žaidimai, video redagavimas.

Skirtingai nuo „Fat“ tipo kliento, „Thin“ klientas yra labai jautrus bet kokioms DDoS (angl. Distributed Denial-of-Service) atakoms. Atakuojant serverį nukenčia daug klientų, kurie nebegali pasiekti serviso arba praranda neišsaugotus pakeitimus.

Sistema naudojanti „Fat“ tipo klientą, skirtingai nuo „Thin“ tipo klientą naudojančių sistemų, didelę duomenų apdorojimo dalį atlieka kliento pusėje, o kiek duomenų yra apdorojama nusprendžia sistemos architektas. Dirbant tokiu principu klientui nebūtinas pastovus ryšys su serveriu ir sesijos metu dingus ryšiui su serveriu pakeitimai gali būti saugomi lokaliai, kol ryšys nebus atstatytas.

1.1. WEB aplikacijos pasirinkimo priežastys

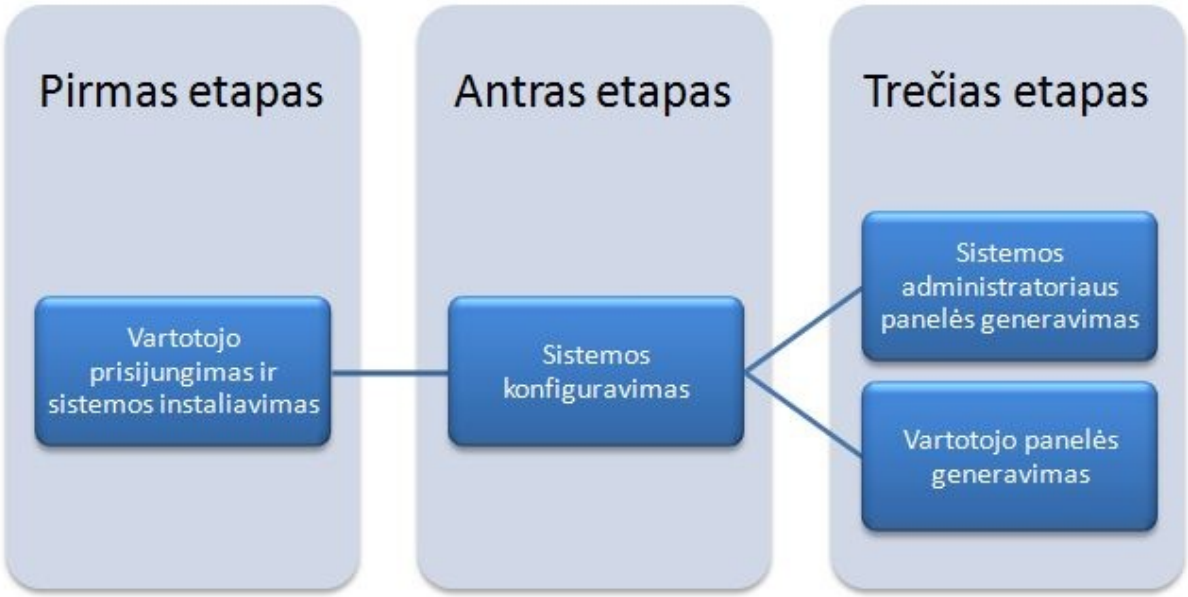
Tolimesniam nagrinėjimui buvo pasirinkta viena šaka, tai WEB aplikacija. Šį pasirinkimą įtakojo kelios priežastys:

- WEB aplikacija suteikia galimybę projektui būti pasiekiamam iš bet kur nereikalaujant papildomos programinės įrangos
- WEB sistema nereikalauja didelių kompiuterio resursų ar spartaus interneto ryšio
- WEB aplikacijoms kurti siūloma daug gerai suderintu programinių paketų, pvz. php, MySQL , Apache ar asp.NET , IIS, MsSQL.

1.2. Teorinis WEB sistemos modelis

Pagrindinė šio darbo užduotis, kurią pasirinkau, yra kuo paprastesnės, universalios WEB sistemos kūrimas, kuri leistų vartotojui lengvai sukurti rezervacijos ar tvarkaraščių sistemą. Sistemos sudėtingumą labai didina jos universalumas ir faktas, kad esant reikalui ir norint vartotojui sistema turi sudaryti rezervacijos grafikus. Yra daug situacijų kai žmogui reikalinga rezervacijos ar tvarkaraščių sudarymo sistema.

1.3. WEB sistemos diegimo ir konfigūravimo schema



Pav. 1 Sistemos diegimo ir gyvavimo ciklas (sudaryta autoriaus)

Vienas iš svarbiausių šio darbo tikslų yra užtikrinti, kad sistema būtų kuo paprastesnė ir kad kuo daugiau žmonių neturinčių jokios programavimo patirties galėtų ja naudotis.

Sistemos principas yra toks, kad vartotojas pats gali susikurti rezervuojamus objektus, jų savybes ir taisykles. Didžiausias iššūkis yra padaryti taip, kad paprastas vartotojas, neturintis programuotojo patirties galėtų viską pasidaryti pats.

1.4. WEB sistemos duomenų bazės diegimas

Kadangi visa sistema yra taikoma būti kuo paprastesne, tai ir sistemos diegimas turi reikalauti kuo mažiau įsikyšimo iš vartotojo pusės. Nors, teoriškai, sistemai tereikia sudėti failus į serverį ir sukurti pradines duomenų bazes, šis procesas gana nemalonus žmogui, nes jis reikalauja gilintis į painią informacinių technologijų terpę.

Augant perkamajai galiai ir pingant informacinėms technologijoms puslapių talpinimo paslaugas teikiančios įmonės vis dažniau siūlo vartotojams galimybę turėti daugiau negu vieną duomenų bazę tame pačiame serveryje. Tai reiškia, kad vartotojui suteikiami papildomi

pasirinkimai ir pradiniai duomenys suteikiami prisijungti prie duomenų bazės bus nepakankami puslapio veikimui. Vartotojui teks sukurti atskirą duomenų bazę, kuri turės tam tikrą prefiksą ir kuri turės atskirus prisijungimo duomenis.

Viena iš internetinių puslapių talpinimo lyderių, www.serveriai.lt, siūlo naudoti „phpMyAdmin“ įrankį (žr. 2 pav.).

Pav. 2 Duomenų bazės priklausančios vienam vartotojui („phpMyAdmin Project“, 2011)

Tačiau naudojant šį įrankį vartotojas neturi galimybės kurti savo duomenų bazės, nes įmonė reikalauja, kad visos duomenų bazės būtų kuriamos pasitelkiant kitą įrankį. Esant tokiai galybei įrankių ir kiekvienam iš jų turint atskirus prisijungimo duomenis, vartotojui tampa sunku orientuotis. Dabar vartotojas norėdamas sukurti savo duomenų bazę turi:

1. Prisijungti prie virtualaus serverio valdymo pulto;
2. Surasti duomenų bazių valdymo panelę;
3. Sukurti duomenų bazę specifikuojant prisijungimo duomenis kurie vėliau bus naudojami prisijungimui prie tos duomenų bazės. Duomenų bazei automatiškai bus priskiriamas prefiksas.

Paprastam vartotojui žinoti šiuos punktus yra nelogiška, o vadovautis jais būtų sudėtinga ir nepatogu. Vartotojo tikslas yra kuo greičiau pradėti naudotis sistema neapsikraunant bereikalinga informacija.

Šiuo atveju paprasčiausias įdiegimo būdas būtų, jeigu vartotojas pradėjęs instaliaciją turėtų įvesti tik pradinius prisijungimo prie sistemos duomenis.

1.5. WEB sistemos failų diegimas

Nesvarbu kaip nori supaprastinti sistemos diegimą yra žingsnių kurių praleisti ar supaprastinti neįmanoma. Vienas iš tokių žingsnių yra pradinių diegimo failų sukėlimas į serverį. Nesukėlus visų būtinų failų į serverį tolimesnis sistemos diegimas yra neįmanomas.

Deja sistemos diegimo failų įdėjimas į serverį yra vienas iš tų žingsnių, kurį padaryti paprastesniu yra labai sudėtinga ir, tiesiog, neverta (procesas gana nesudėtingas, bet jo automatizavimas reikalauja daug resursų ir bet kokiu atveju reikalautų vartotojo įsikišimo). Šiuo atveju paprasčiausia išeitis būtų šį nesudėtingą procesą trumpai aprašyti instrukcijomis kuriomis vadovaudamasis vartotojas turėtų reikiamus duomenis sudėtį į serverį.

Pilną sistemos diegimo schemą galima aprašyti septyniais žingsniais (žr. 3 pav.).

Pav. 3 Sistemos diegimo schema (sudaryta autoriaus)

Kadangi vartotojas bet kuriuo diegimo metu turėtų turėti galimybę nutraukti instaliaciją, po ketvirto žingsnio sekantys žingsniai privalo turėti galimybę failų sistemą ir duomenų bazę atstatyti į pradinę padėtį. Dėl to turi būti numatytos funkcijos duomenų bazių ir lentelių trynimui (angl. Drop) bei failų ir katalogų šalinimui iš serverio.

Nutraukus sistemos diegimą bet kuriuo metu ir bandant sistemą diegti iš naujo, vartotojas neturi pastebėti jokių skirtumų nuo pirmojo sistemos diegimo proceso.

1.6. Teorinis objektų kūrimo modelis

Kad susidarytumėme įspūdį kokie objektai gali dalyvauti visoje rezervavimo schemoje verta panagrinėti kelis pavyzdžius.

1.6.1. Teoriniai pavyzdžiai

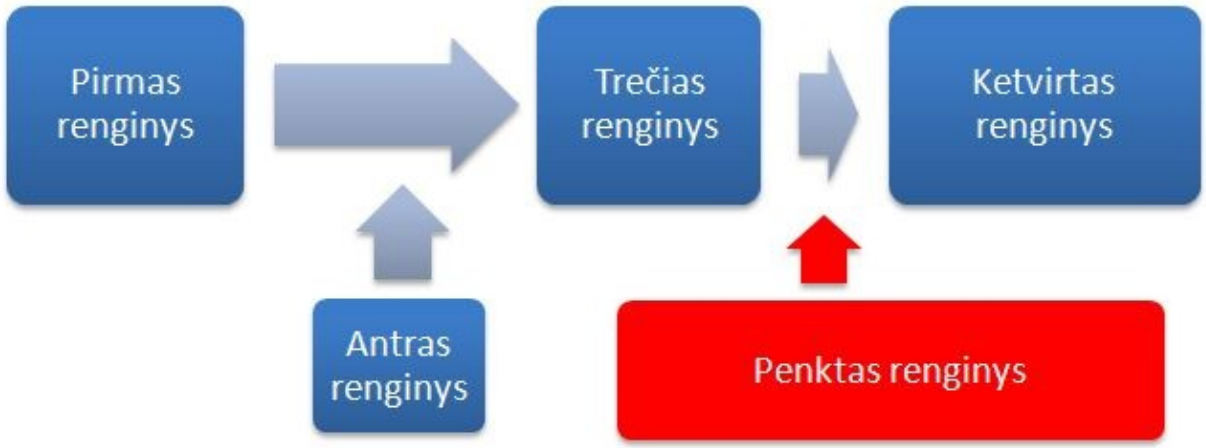
1.6.1.1. Pobūvių salė

Tarkim rezervuojamas objektas yra didelė pobūvių salė kuri užima visą pastatą. Tokioje situacijoje viskas gana paprasta. Administratorius kurdamas salės tipo objektą turi nustatyti:

- darbo pradžios laiką
- darbo pabaigos laiką
- talpinamų žmonių kiekį
- intervalą reikalingą tarp renginių
- klientui leidžiamo pasirinkti rezervacijos laiko intervalą, pvz. jei intervalas penkios minutės tai klientas galėtu rinktis 10:00, 10:05 ir t.t.

Šiuo atveju atėjęs klientas turėtų užpildyti paprastą formą kuri reikalautų personalinių duomenų, šiek tiek informacijos apie patį pobūvį ir, žinoma, laiko kada viskas vyks ir kiek tai truks.

Salės rezervavimą galima pavaizduoti paprasta schema (žr. 4 pav.).



Pav. 4 Rezervavimo pavyzdys su sankirtų atradimu (sudaryta autoriaus)

Atėjus klientui Pirmas, Trečias ir Ketvirtas renginiai jau buvo rezervuoti. Jei klientas norės rezervuoti salę laiku sutampančiu su Antro renginio laiku paveikslėlyje, sistema patikrinusi užimtumą leis jam tai padaryti, nes Pirmas renginys pasibaigia prieš Antro renginio pradžią ir antras renginys baigiasi prieš Trečiojo renginio pradžią. Tačiau užsisakant renginį Penktojo renginio laiku sistema turi aptikti, kad tuo metu salė yra užimta.

Sistemai reikia paprasto, bet greito proceso gebančio surasti ar užsakomu metu renginys gali būti įterptas. Užsakomas renginys bus atmetamas jeigu jo laikas tenkins nors vieną iš šių savybių:

1. Užsakomo renginio pabaiga yra vėliau nei kurio nors užsakyto renginio pradžia;

2. Užsakomo renginio pradžia yra anksčiau nei kurio nors užsakyto renginio pabaiga;
3. Užsakomo renginio pradžia yra anksčiau nei kurio nors renginio pradžia, o užsakomo renginio pabaiga yra vėliau nei to paties užsakyto renginio pabaiga;
4. Užsakomo renginio pradžia vėliau nei kurio nors užsakyto renginio pradžia, o užsakomo renginio pabaiga yra anksčiau to paties užsakyto renginio pabaigos;
5. Užsakomo renginio pabaiga yra vėliau nei salės darbo laiko pabaiga.

1.6.1.2. Kirpykla

Šis atvejis yra šiek tiek skirtingas nei pirmasis. Visų pirma, skirtingai nuo salės, čia rezervuojamų objektų, šiuo atveju kirpėjų, yra ne vienas, o daug, dėl to objekto kuriame talpinami rezervuojami objektai ir rezervuojamų objektų ryšys yra *1:n* tipo. Dėl šio skirtumo administratoriui kuriant rezervuojamą objektą atsiranda papildomų savybių:

A. Skirtingai nuo praeito pavyzdžio skirtingi kirpėjai gali turėti kitokį darbo pradžios ir pabaigos grafiką, dėl to turi būti galimybė administratoriui keisti atskirų darbuotojų darbo laikus.
B. Šiuo atveju vartotojui suteikti galimybę rinktis kaip ilgai jis bus aptarnaujamas būtų nelogiška. Tam reikalinga atskira sistema. Sprendimas labai paprastas. Sukūrus rezervuojamo objekto architektūrą galima sukurti atskiras procedūras kurias gali atlikti rezervuojami objektai, o vėliau tas procedūras priskirti konkretiems objektams, pavyzdžiui:
   a. Vyrų kirpimas – trukmė 45 minutės.
   b. Dažymas – trukmė 1 valanda ir t.t.

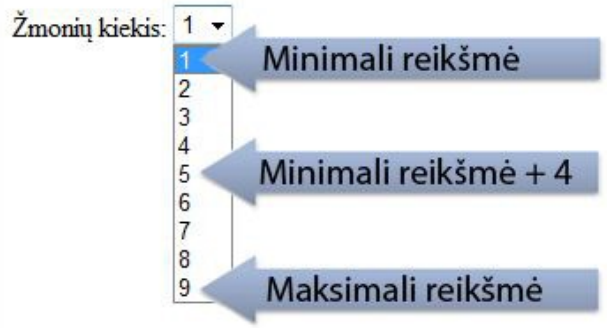
   Šis abstraktus modelis tinka ne tik kirpyklai, bet visoms situacijoms kuriose vienas rezervuojamas objektas aptarnauja vieną klientą, o laiko sąnaudas apsprendžia administratorius. Pavyzdžiui soliariumas kuriame rezervuojami objektai yra deginimuisi skirta įranga. Administratoriui tereikia sukurti procedūrą, tarkim, Soliariumo procedūra – 15 minučių, ir priskirti ją atitinkamoms mašinoms.
C. Kadangi rezervuojamų objektų yra daugiau nei vienas, administratoriui reikia pasirinkti ar leidžiama vartotojui laisvai rinktis objektą ar sistema pati jį parenka. Jei vartotojui leidžiama rinktis tai rezervavimo algoritmas nesikeičia nuo pirmojo pavyzdžio. Jei vartotojui teisė rinktis nesuteikiama tai algoritmui reikalingi nedideli patobulinimai. Kodas panašus, tačiau reikia atkreipti dėmesį, kad tikrinant skirtingų darbuotojų užimtumą iš duomenų bazės paimama informacija apie jo darbo laiką. Kadangi sistema ieško tik pirmo laisvo darbuotojo, visų darbuotojų sąrašas yra išrikiuojamas atsitiktine tvarka, o sąrašas sudaromas iš duomenų bazės paimant

darbuotojus, kuriems priskirta vartotojo norima procedūra. Išėjus iš ciklo sistema tikrina ar specialiam kintamajam buvo priskirta unikalus darbuotojo identifikatoriaus reikšmė. Šis kintamasis keičiamas tik radus laisvą darbuotoją, dėl to, jei kintamasis neturi reikšmės reiškia, kad laisvų darbuotojų rasta nebuvo.

1.6.1.3. Restoranas

Ši situacija turi daug panašumų su antrąja. Objektai šiuo atveju yra rezervuojami staliukai. Staliukų kaip ir kirpėjų yra ne vienas, o daug, bet staliukų darbo laikas yra toks pats kaip restorano. Todėl kuriant rezervuojamus objektus reikia leisti rinktis administratoriui ar tie objektai turi specifinį darbo laiką.

Skirtingai nuo kirpėjo staliukas gali „aptarnauti“ keletą žmonių. Šioje vietoje galima sistemą daryti dviem būdais:



Pav. 5 Sąrašas sugeneruotas algoritmo žmonių kiekio pasirinkimui (sudaryta autoriaus)

A. Staliukam priskiriamos procedūros. Šiuo atveju rezervacijos metu vartotojui būtų leista rinktis iš sąrašo kuriame būtų visos galimos žmonių kiekio reikšmės nuo minimalaus žmonių kiekio galinčio sėdėti prie bet kurio staliuko iki maksimalaus kiekio. Tai darytume nes netgi jei nėra staliuko talpinančio specifiškai keturių žmonių, juos gali talpinti ir staliukas kuris talpina penkis žmones. Taip pat vietoj to, kad staliukas išsiskleidžiančiame meniu turėtų atskirą eilutę kurioje būtų įrašytas maksimalus staliuko talpinimas žmonių kiekis, kuris kartotųsi, būtų tvarkingai surikiuoti, logiški ir unikalūs pasirinkimai. Skirtingai nuo kirpyklos pavyzdžio, šios procedūros turi ne laiko, o kiekio apribojimą. Maksimalų skaičių žmonių aptarnauti galintis staliukas turės visas procedūras, o minimalų skaičių aptarnaujantis turės tik vieną. Tai darytume dėl to, kad du žmones talpinantį staliuką galėtų rinktis ir vienas pietauti atėjęs žmogus.
B. Kuriant staliuko objektą sukurti savybę kuri aprašytų maksimalų talpinamų žmonių kiekį. Tuomet klientui būtų leidžiama įvesti žmonių kiekį arba generuoti išsiskleidžiančio meniu formą kaip ir kitame variante.

1.6.2. Objektai

Iš aukščiau panagrinėtų pavydžių galima susidaryti bendrą vaizdą apie objektus kurie bus rezervuojami ir tuo pačiu privalės būti sukurti sistemoje prieš pradedant ja naudotis.

Iš panagrinėtų pavyzdžių matyti, kad kiekvienas rezervuojamas objektas yra skirtingas, bet yra pagrindinės trys jų rūšys:

1. rezervuojami objektai
2. rezervuojamų objektų konteineriai
3. rezervuojami rezervuojamų objektų konteineriai.

1.6.2.1. Objektų kūrimas

Kuriant visus objektus, nepriklausomai nuo to ar jie konteineriai ar rezervuojami objektai, administratorius privalės sukurti ir nurodyti laukus kurie yra būtini ir tuo pačiu galės sukurti pasirenkamus laukus.

Kadangi sistema yra skirta rezervavimui, pagrindinis visų objektų atributas yra laikas. Kiekvienas rezervuojamas objektas, rezervuojamų objektų konteineris ir rezervuojamas rezervuojamų objektų konteineris yra artimai susietas su laiku ir pirmiausiai kuriant juos vartotojas privalo nustatyti tris dalykus:

1. Savaitės dienas kada konteineriai, konteineriui priklausantys objektai ar paprasti objektai gali būti rezervuojami.
2. Darbo laiko valandas. Čia reikia atkreipti dėmesį, kad objekto darbo laikas nesikirstų su jo konteinerio darbo laiku. Šiuo atveju vėl galima būtų panaudoti šiek tiek modifikuotą 1.6.1.1 skyrelyje aptartą sankirtų paiešką. Čia užtektų tikrinti ar atskiro objekto darbo laikas neprasideda anksčiau už jo konteinerio ir ar nesibaigia vėliau.
3. Nedarbo dienas ir šventes.

Reikia atkreipti dėmesį, kad yra du tipai rezervuojamų objektų:

a. Gyvi žmonės. Tai gali būti kirpėjos, dantistai, treneriai ir t.t.
b. Daiktai. Tai gali būti staliukai restorane, pirtys pirčių komplekse, pobūvių salės ir t.t.

1.6.2.1.1. Rezervuojami žmonės

Skirtingai nuo rezervuojamų daiktų, rezervuojami žmonės turi daug unikalių savybių:

- skirtingi darbo laikai
- skirtingi laikai pietums
- skirtingos atostogų datos
- skirtingos atliekamos funkcijos ir t.t.

Žmonės visada atlieka kažkokias prieš tai nustatytas funkcijas – procedūras, kurios būna apibrėžtos laiko atžvilgiu. Tarkim kirpėjos atliekamas vyrų kirpimas yra procedūra kuriai gali

būti skirtas tam tikras laiko intervalas. Tokiu pačiu principu vadovaujantis galima aprašyti beveik visas situacijas kur galima rezervuoti žmones, ar tai būtų kirpykla ar grožio salonas ar dantisto kabinetas.

1.6.2.1.2. Rezervuojami daiktai

Dažniausiai rezervuojami daiktai neturi tokių unikalių savybių kaip skirtingas darbo laikas, kaip tai turi žmonės, todėl administratoriui kuriant objektus kurie atlieka tas pačias funkcijas, turi būti leidžiama pasirinkti, kad darbo laiką, atostogų dienas ir t.t. kopijuoti nuo to objekto konteinerio. Pavyzdžiui kuriant restorane stovinčių staliukų objektus būtų nelogiška kiekvienam iš jų priskirti skirtingą darbo laiką, nebent yra papildomų sąlygų, tarkim į darbo pabaigą, sumažėjus klientų skaičiui, neleisti rezervuotis tam tikrų staliukų.

Skirtingai nuo rezervuojamų žmonių, rezervuojami daiktai gali būti rezervuojami specifinėms procedūroms ir tam tikram laikui. Pavyzdžiui staliukas restorane gali būti rezervuojamas tam tikram laiko tarpui, o įrenginys skirtas deginimuisi soliariume jau turės tam tikrą procedūrą su jau nustatytu laiko kiekiu.

Kuriant rezervuojamo daikto objektą reikia nurodyti kiek žmonių vienu metu jis gali aptarnauti. Be to reikia nurodyti ar vienas objektas gali priimti užsakymus iš kelių skirtingų vartotojų. Pavyzdžiui staliukas restorane gali talpinti tam tikrą žmonių kiekį, bet, netgi, jeigu staliukas nėra užpildytas visiškai, kitas vartotojas vietų prie to staliuko rezervuotis negali. O, tarkim, pirčių komplekse esančiose pirtyse, talpinančiose tam tikrą žmonių kiekį, tas pats rezervuojamas objektas, pirtis, gali priimti užsakymus iš skirtingų vartotojų.

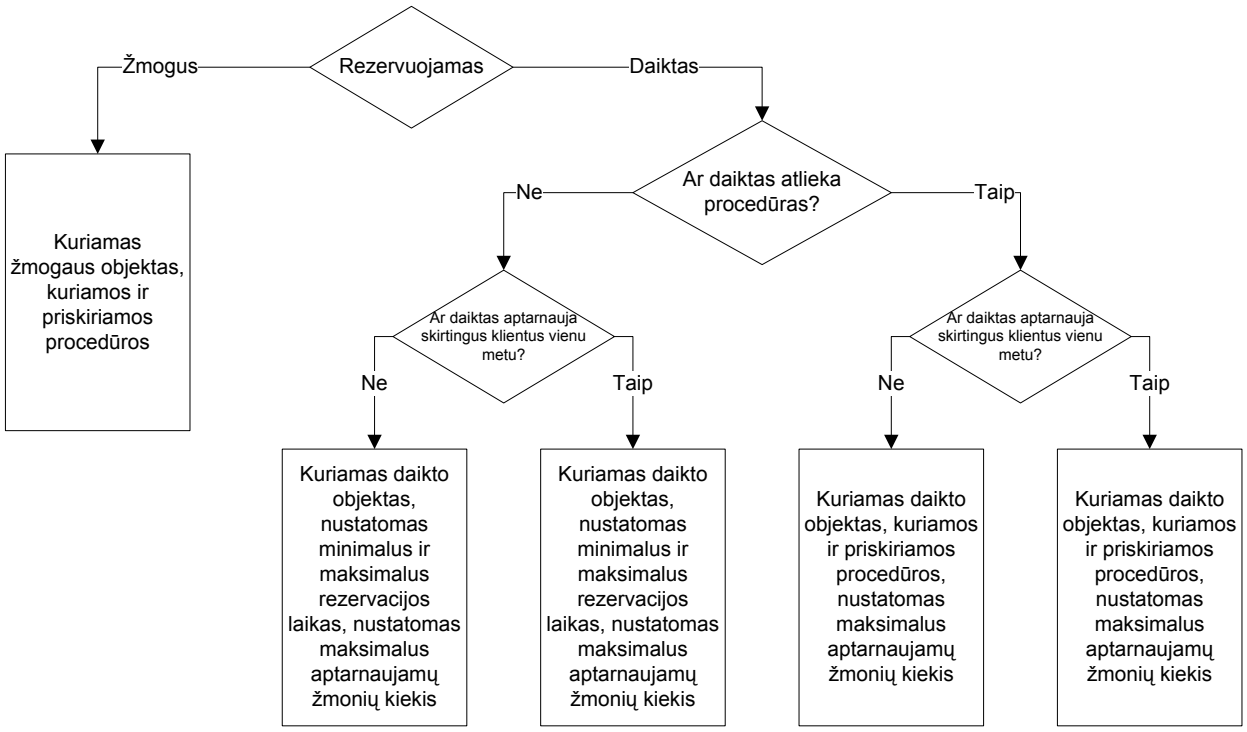
Rezervuojamų daiktų pavidalu galima įgyvendinti ir kelias kitas situacijas kuriose rezervuojami ne patys daiktai.

Viena iš tokių situacijų yra kelionių agentūra kurioje galima rezervuotis vietas kelionėje. Ši situacija turi savo išskirtinumų:

1. Pačios kelionės neturi rezervuojamų vietų limito, limitą turi tik transporto priemonės, viešbučiai ir t.t. kurios aptarnaus vartotojus;
2. Kelionėms negalioja jų konteinerio savybės (darbo laikas ir t.t.);
3. Kelionės turi fiksuotą pradžios ir pabaigos laiką, kuris nustatomas ne kuriant kelionės objektą, o kuriant pačią kelionę.

1.6.2.2. Objektų kūrimo schema

Apžvelgus du rezervuojamų objektų tipus ir jų ypatybes, objektus galima kurti pagal tam tikrą schemą:



Pav. 6 Trumpa objektų kūrimo schema (sudaryta autoriaus)

Daikto savybė leidžianti aptarnauti skirtingus klientus vienu metu neturi įtakos galutiniam rezervuojamo objekto kūrimo procesui, tačiau turi didelę svarbą galutiniam rezervacijos algoritmui.

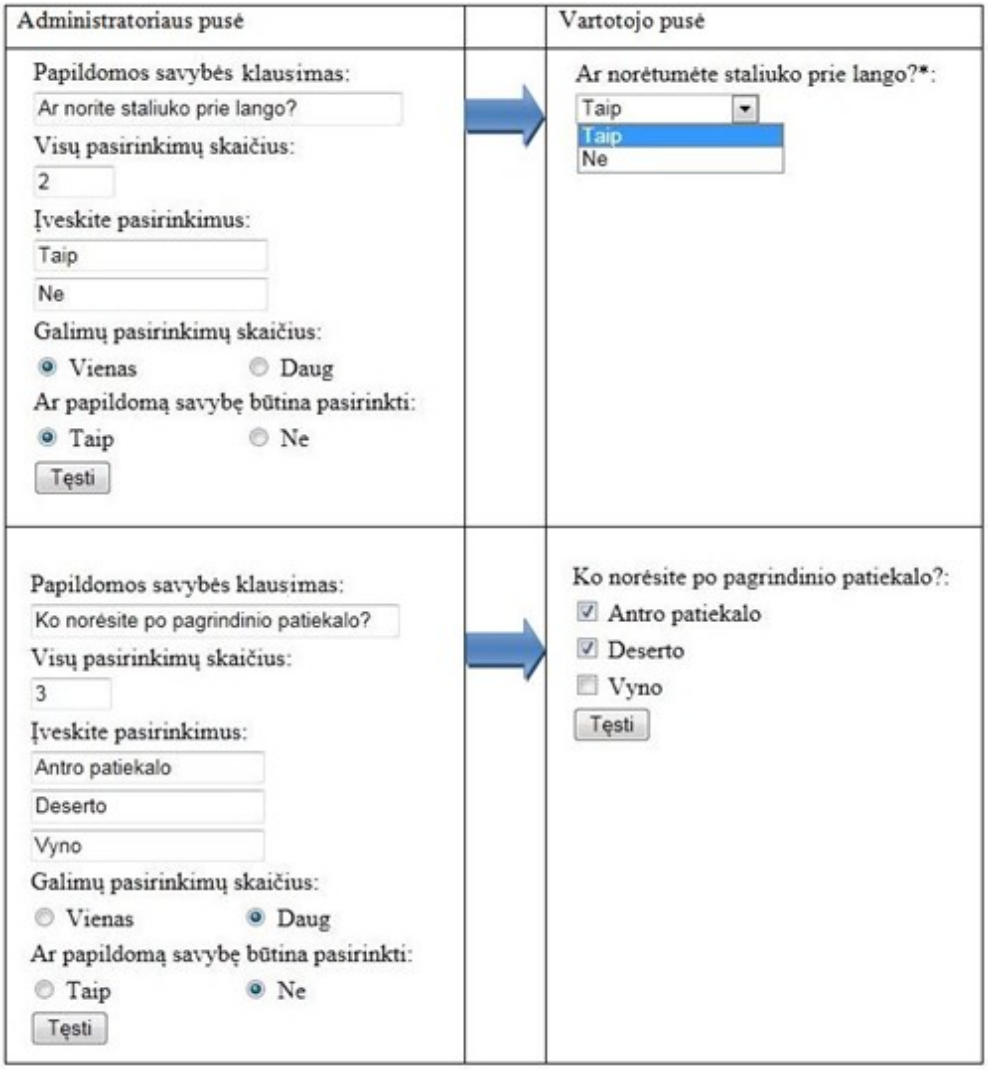
1.6.2.3. Papildomos rezervuojamų objektų savybės

Kurdamas rezervuojamą objektą administratorius turi turėti galimybę jam priskirti unikalias savybes, kurios, galbūt netinka kitiems objektams. Tarkim keli restorano staliukai yra prie lango ir administratorius nori, kad rezervuojant staliukus žmogus galėtu rinktis, ar jis nori sėdėti prie lango ar ne.

Prieš kurdamas papildomas savybes vartotojas jau turi žinoti kiek bus skirtingų tos papildomos savybės pasirinkimo rūšių ir kiek iš tų rūšių rezervuodamasis galės rinktis žmogus. Galimi du variantai:

1. kelios papildomos pasirinkimo rūšys ir vienas galimas pasirinkimas
2. kelios papildomos pasirinkimo rūšys ir keli galimi pasirinkimai

Pirmojo atvejo pavyzdys ir galėtų būti pasirinkimas ar norima sėdėti restorane prie lango. Antrojo atvejo pavyzdys galėtų būti pasirinkimas restorane, kur vartotojas galėtų rinktis ar jis valgys desertą ir ar gers vyną.

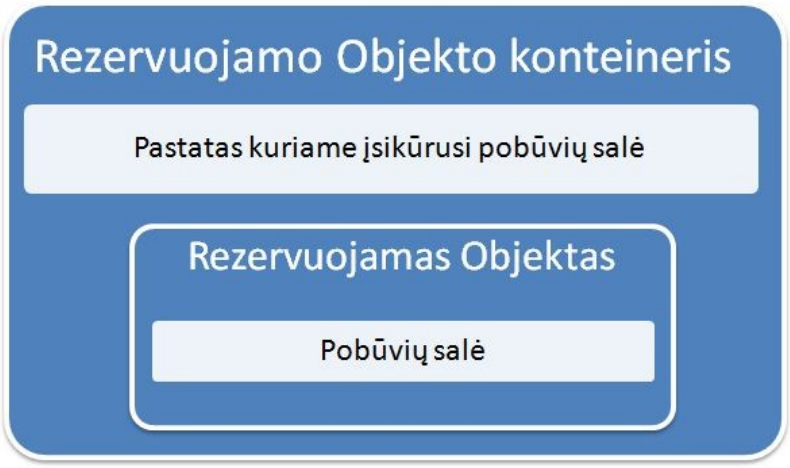


Pav. 7 Papildomos savybės kuriant administratoriui ir rezervuojant vartotojui (sudaryta autoriaus)

1.6.2.4. Skirtingų tipų objektai

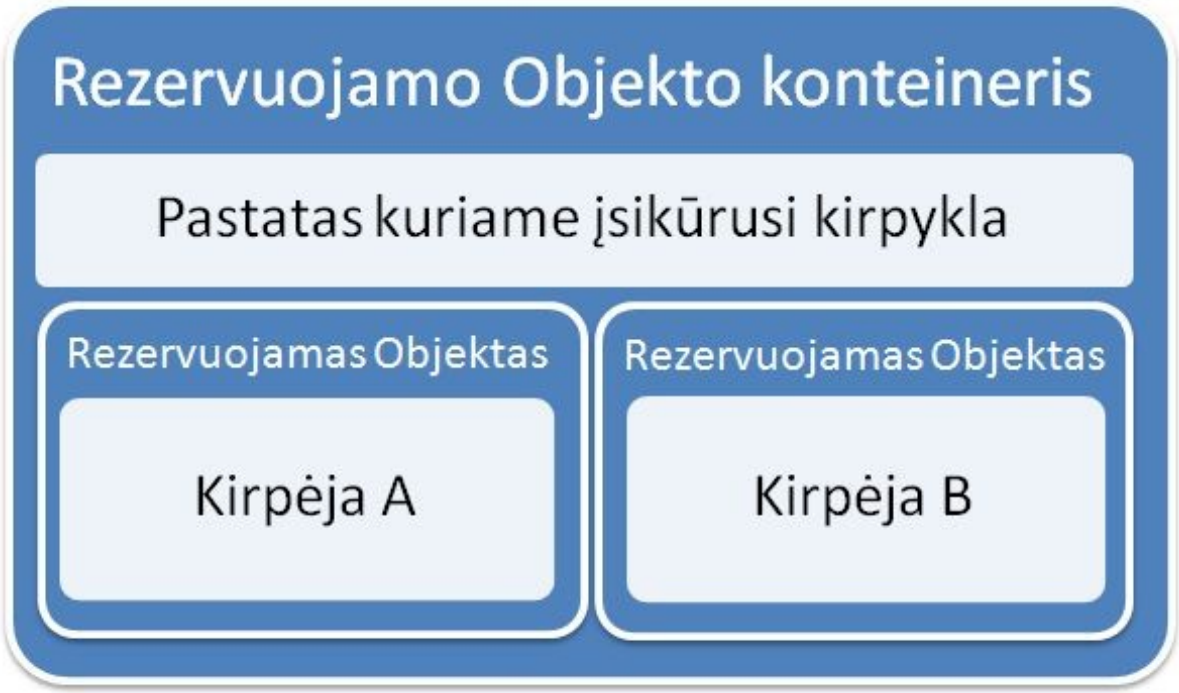
Šio, 1.6.2, skyrelio pradžioje trumpai išvardinau tris objektų rūšis kurios pasitaiko rezervacijos sistemose. Rezervuojami rezervuojamų objektų konteineriai bei rezervuojamų objektų konteineriai prie visos sistemos kūrimo schemos prideda dar pora klausimų. Konteineriai pasidaro aktualūs tada, kai egzistuoja daugiau nei vienas rezervuojamas objektas.

Ne visais atvejais, objektui reikia kurti rezervuojamo objekto konteinerį. Pobūvių salės atveju ir objekto konteineris ir pats rezervuojamas objektas turi identiškas savybes, dėl to kurti du atskirus objektus yra neracionalu (žr. 8 pav.).



Pav. 8 Rezervuojamo objekto konteineris ir rezervuojamas objektas pobūvių salės atveju (sudaryta autoriaus)

Šis atvejis lengvai pritaikomas bet kokioms situacijoms kuriose tėra vienas rezervuojamas objektas, tai gali būti dantistas ar pobūvių salė.



Pav. 9 Rezervuojamo objekto konteineris ir rezervuojamas objektas kirpyklos atveju (sudaryta autoriaus)

Kirpyklos atveju, rezervuojamo objekto konteineris būtinas. Nors skirtingi objektai gali turėti skirtingas savybes jos niekada nebus pirminės prieš hierarchiškai aukščiau esančio rezervuojamo objekto konteinerio savybes. Skirtingi darbuotojai gali turėti skirtingus darbo laikus, tačiau nei vienas iš jų darbo nepradės anksčiau ar nebaigs vėliau nei nustatyta rezervuojamų objektų konteinerio savybėse. Šiuo atveju reikia sukurti rezervuojamų objektų konteinerį kuriam ir priklausys visi darbuotojai.

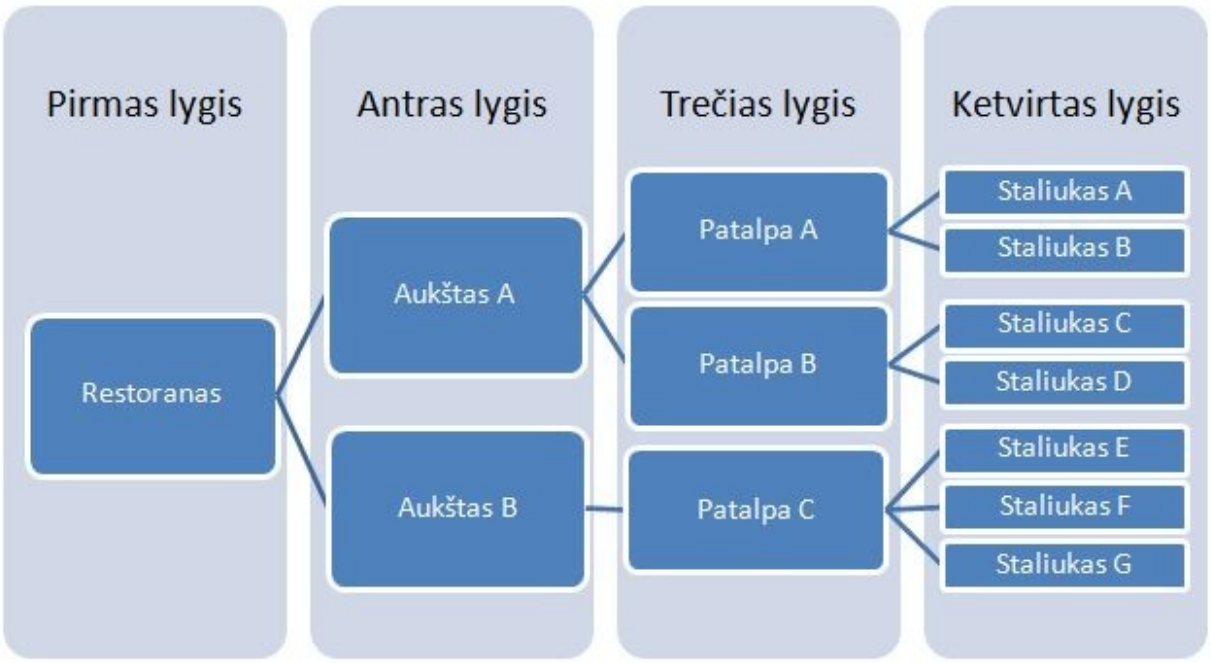
Trečiame pavyzdyje atsiranda rezervuojamas rezervacijos objekto konteineris. Skirtingai nuo pirmojo pavyzdžio, šiame yra daug rezervuojamų objektų ir skirtingai nuo antrojo pavyzdžio visi rezervuojami objektai čia atlieka tą pačią funkciją. Reikia atkreipti dėmesį, kad rezervuojamų rezervacijos objektų konteinerių gali būti ne vienas. Kaip matyti dešimtame paveikslėlyje, rezervuojamam rezervacijos objektų konteineriui gali priklausyti, ne tik rezervuojami objektai, bet ir kiti rezervuojamų rezervacijos objektų konteineriai. Todėl, trečiame pavyzdyje, kambarys yra rezervuojamas rezervacijos objekto konteineris ir jam priklauso staliukai esantys tame kambaryje, o vartotojas norėdamas užsisakyti visus staliukus esančius patalpoje gali, tiesiog, rezervuoti rezervuojamą rezervacijos objektų konteinerį - kambarį. Tačiau esant reikalui, vartotojas, taip pat, gali rezervuoti ir visą restoraną, rezervuojamą rezervacijos objektų konteinerį, kuriam priklauso kiti rezervuojami rezervacijos objektų konteineriai ir kuriem, galiausiai, priklauso patys rezervuojami objektai – staliukai. Tuo pačiu žmogus negali prarasti galimybės rezervuotis atskirus staliukus.

Viename hierarchiniame lygyje gali egzistuoti ir rezervuojamas rezervuojamų objektų konteineris ir patys rezervuojami objektai. Pavyzdžiui, jeigu restoranas turi vieną atskirą patalpą kurią skiria pokyliams, o visi likę staliukai gali būti rezervuojami atskirai. Šiuo atveju ir kambarys ir staliukai tiesiogiai priklauso restoranui.

Pav. 10 Rezervuojamas rezervuojamo objekto konteineris ir rezervuojamas objektas restorano atveju (sudaryta autoriaus)

1.6.2.5. Sistemos apribojimai

Teoriškai, rezervuojamų rezervacijos objektų konteinerių gali būti be galo daug ir kiekvienas iš jų gali priklausyti skirtingam rezervuojamam rezervacijos objektų konteineriui, kas padarytų sistema be galo didelę. Dėl to būtų logiška apibrėžti maksimalų sistemos gylį. Sistemos gylis tai žingsnių skaičius reikalingas nueiti nuo pirmojo rezervuojamų objektų konteinerio iki mažiausio rezervuojamo objekto.



Pav. 11 Situacija kurios gylis yra keturi (sudaryta autoriaus)

Yra daug pavyzdžių kurie yra vieno, dviejų ar trijų žingsnių gylio. Labai retai pasitaiko keturių žingsnių gylio situacijų. Vienas iš keturių žingsnių gylio pavyzdžių būtų kelių aukštų restoranas, kuriame visi aukštai suskirstyti į atskiras patalpas. Vartotojas galėtų rezervuoti visą restoraną, vieną ar kelis jo aukštus, atskirą patalpą arba vieną staliuką.

Kadangi ir keturių žingsnių reikalaujančių pavyzdžių yra gana mažai, tai būtų logiška visą sistemą apriboti būtent keturiais žingsniais.

1.6.3. Teorinis duomenų bazės modelis

DBVS (Duomenų Bazių Valdymo Sistema) yra programų rinkinys, skirtas informacijos organizavimui, laikymui, apdorojimui ir išrinkimui iš duomenų bazės. DBVS yra skirstomos pagal jų duomenų struktūras arba tipus. DBVS priima duomenų užklausas iš atitinkamų programų ir nurodo operacinei sistemai persiųsti atitinkamą informaciją. Naudojant DBVS, atsiradus reikalui, organizacijų naudojamos informacinės sistemos gali būti keičiamos labai paprastai. Naujos duomenų kategorijos duomenų bazėje gali būti pridedamos labai lengvai, nesugadinant jau esamų duomenų ir sistemos.

Organizacijos neretai vartoja kelių DBVS sistemų junginį: kasdieninėms transakcijoms apdoroti pasitelkiama viena DBVS, o vėliau informacija apie transakcijas perkeliama į kitą sistemą, labiau tinkančią atitinkamoms užklausoms ir analizei. Duomenų bazes modeliuoja ir prižiūri duomenų bazių administratoriai.

Šiuo metu yra trys populiariausios DBVS rūšys:

1. Reliacinės duomenų bazių valdymo sistemos – RDBVS;
2. Objektinės duomenų bazių valdymo sistemos – OODBVS;
3. Ne-Reliacinės duomenų bazių valdymo sistemos – NoSQL.

1.6.3.1. Reliacinės duomenų bazių valdymo sistemos

Reliacinės duomenų bazės jungia duomenis pagal bendras jų charakteristikas esančias duomenų aibėje. Taip sugrupuoti duomenis yra kur kas lengviau suprantami daugeliui žmonių. Pavyzdžiui, duomenų aibė kurioje yra visi duomenys apie nekilnojamo turto prekybą mieste, gali būti sugrupuoti pagal metus kada įvyko transakcija, pagal parduoto turto vertę arba turtą įsigijusio žmogaus pavardę ir t.t. Toks grupavimas naudoja reliacinį modelį (schemą) ir dėl to yra vadinamas reliacine duomenų baze.

Programinė įranga kuri tokiu būdu grupuoja duomenis yra vadinama Reliacine duomenų bazių valdymo sistema (RDBVS).

Dvyliktame paveikslėlyje pavaizduota duomenų bazės lentelė, kuri vėliau bus naudojama praktiniame modelyje. Matyti aštuoni lentelės įrašai (angl. Tuple) kurie yra išrikiuotos duomenų aibės. Kiekvienas duomenų aibės elementas yra įrašų atributas.

	id	name	surname	worker_time_start	worker_time_end
	1	Rasa		09:00:00	15:00:00
	2	Vita		08:00:00	18:00:00
	3	Artūras		08:00:00	18:00:00
	4	Kristina		08:00:00	18:00:00
	5	Teresa		08:00:00	18:00:00
	6	Diana		08:00:00	18:00:00
	7	Milda		08:00:00	18:00:00
	8	Andželika		08:00:00	18:00:00

Pav. 12 RDBVS „MySQL“ duomenų bazės lentelė (sudaryta autoriaus)

Reliacinės duomenų bazės dominuoja duomenų saugojimo rinką ir yra plačiausiai naudojamos saugoti įvairaus tipo informacijai, nuo personalios informacijos iki gamybinių ir logistinių duomenų.

1.6.3.2. Objektinės duomenų bazių valdymo sistemos

Objektinė duomenų bazė (dar žinoma kaip objektiškai-orientuota duomenų bazė) yra duomenų bazių modelis kuriame informacija yra atvaizduojama kaip objektas, toks koks yra naudojamas objektiškai-orientuotuose programavimo kalbose.

Kai duomenų bazių galimybės yra sujungiamos su objektiškai-orientuotos programavimo kalbos galimybėmis, gaunamas rezultatas yra Objektinė duomenų bazių valdymo sistema (OODBVS). Šiandien labai paplitęs objektiškai-orientuotas programavimas daro OODBVS idealiu produktu programuotojams, nes jie gali kurti produktus ir juos saugoti kaip objektus.

Šiais laikais informacija nėra sudaryta vien iš teksto. Dabar paplitę vaizdo, garso failai, nuotraukos. Visi šie duomenų tipai yra kompleksiniai. Be papildomų modifikacijų RDBVS neturi galimybės palaikyti šiuos kompleksinius duomenų tipus. Kadangi objektinė duomenų bazė yra integruota su objektiškai-orientuota programavimo kalba, programuotojas gali išlaikyti duomenų tipų vientisumą, nes ir OODBVS ir programavimo kalba naudos tą patį objekto atvaizdavimo modelį. RDBVS projektai naudojantys kompleksinius duomenų tipus turėtų būti skaidomi į dvi atskiras užduotis: duomenų bazių modelis ir aplikacijos modelis.

Su intranetu kartu auga ir WEB-grįstų technologijų vartojimas. Didžiosios kompanijos pradėjo domėtis OODBVS kaip priemone galinčia atvaizduoti jų kompleksinius duomenis. OODBVS, kuri buvo suprojektuota saugoti duomenims kaip objektams, suteikia pranašumą kompanijoms kurios yra susijusios su multimedijos pristatymu ir kurios naudoja kompiuterinį modeliavimą (angl. Computer-Aided Design).

1.6.3.3. Ne-reliacinės duomenų bazių valdymo sistemos

Terminas „NoSQL“ yra naudojamas, kad atskirti DBVS kurios skiriasi nuo klasikinių RDBVS. Šios sistemos gali nereikalauti fiksuotų lentelių schemų, dažniausiai vengia jungimo (angl. Join) operacijų ir tipiškai yra plečiamos horizontaliai. Dažnai tokios duomenų bazės vadinamos struktūrinėmis saugyklomis. Tokio tipo duomenų bazės poaibis galėtų būti paprasta reliacinė duomenų bazė.

Labiausiai nusisekę tokio tipo duomenų bazių įgyvendinimo pavyzdžiai (Lith A., Mattsson J., 2010):

1. „Google BigTable“
2. „Amazon Dynamo“
3. „Apache Cassandra“

Dabar plačiausiai naudojamos RDBVS rodo labai prastus rezultatus su tam tikromis aplikacijomis kurios intensyviai skaito ir rašo duomenis, didelių dokumentų indeksavime, informacijos pateikime didelių vartotojų srautų sulaukiančiuose puslapiuose ir taip pat

puslapiuose kuriuose naudojamas pastovus duomenų srautas (angl. Streaming media). Tipinės RDBVS būna pritaikytos dažniems mažų duomenų kiekiams įrašyti ar nuskaityti arba didelėms transakcijoms kurios retai įrašomos ar skaitomos. Kita vertus NoSQL palaiko dažnas ir dideles rašymo ir skaitymo apkrovas. NoSQL aptarnauja 3 terabaitus skirtus „Digg“ puslapio žaliesiems ženkliukams, „Facebook“ socialinio tinklo 50 terabaitų skirtų paieškai pašto dėžučių turinyje ir 2 petabaitus „eBay“ puslapio informacijos.

Kai kurios NoSQL sistemos naudoja paskirstytą architektūrą kuri duomenis paskirsto per kelis serverius dažnai naudojant paskirstytas maišos lenteles (angl. Hash tables). Šiuo principu sistema gali lengvai plėstis pridedant papildomus serverius, o vieno iš serverių netekimas gali būti toleruotinas.

1.6.3.4. Tinkamos DBVS pasirinkimas

Visos trys DBVS turi savo nišas ir yra labai efektyvios tam tikrose situacijose, tačiau mano kuriamoje sistemoje labiausiai tinka RDBVS. Nors OODBVS yra greitesnė už RDBVS, tačiau pagrindinė informacija kuri bus saugoma duomenų bazėje - tekstiniai įrašai, o ne multimedija. Taip pat visa sistema programuojama plačiai paplitusia „PHP“ kalba, o ne viena iš geriausiai suderinamų objektiškai-orientuotų kalbų (Ruby, Python, Perl, Java, C#, Visual Basic .NET, C++).

NoSQL atkrenta, dėl to, kad sistemoje saugomas duomenų kiekis yra labai nedidelis. Vienintelė situacija kurioje, galbūt, pasiteisintų NoSQL, yra, jeigu saugotume visas įmanomas rezervuojamas procedūrų rezervacijos galimybes.

Procedūra turi laiką reikalingą tai procedūrai atlikti (angl. Duration) (žr. 13 pav.). Dabar rezervuojantis laiką vartotojas tiesiog pasirenka kada jis nori pradėti procedūrą, o sistema automatiškai rezervuoja rezervuojamą objektą tam laikui, todėl gaunasi minimalus kiekis įrašų, nes sukuriamas tik vienas rezervaciją atitinkantis įrašas.

	Laukas	Tipas	Palyginimas	Atributai	Null	Nutylint	Papildomai
☐	id	int(11)			Ne	*None*	auto_increment
☐	belongs_to	int(11)			Ne	*None*	
☐	name_lt	varchar(255)	utf8_general_ci		Ne	*None*	
☐	description_lt	text	utf8_general_ci		Ne	*None*	
☐	price	varchar(10)	utf8_general_ci		Ne	*None*	
☐	duration	time			Ne	*None*	
☐	active_from	date			Ne	*None*	
☐	active_until	date			Ne	*None*	
☐	auto_generated_holiday	int(1)			Ne	0	

Pav. 13 Procedūrų lentelės schema (sudaryta autoriaus)

Tačiau yra kitas būdas padaryti tą patį. Galima saugoti visus įmanomus variantus kada gali būti rezervuojama procedūra. Tarkim tikslumas kuriuo vartotojas gali rezervuoti yra penkios minutės, darbo pradžios laikas yra 8:00, o darbo pabaiga 18:00. Šiuo atveju būtų saugoma 120 įrašų susijusių su konkrečia procedūra ir diena.

Pav. 14 Duomenų bazės, generuojančios didelius informacijos kiekius, modelis (sudaryta autoriaus)

Jei egzistuotų dešimt procedūrų, tai per mėnesį būtų sukaupiama per 36000 įrašų susijusių vien su rezervacijos laikais. Esant tokiems duomenų kiekiams NoSQL taptu šiek tiek pranašesnė už RDBVS. Tačiau toks informacijos kaupimo principas yra neracionalus, nes daugelis įrašų bus niekada nepanaudoti ir neperskaityti.

1.7. Teorinis sistemos konfigūravimo funkcionalumas

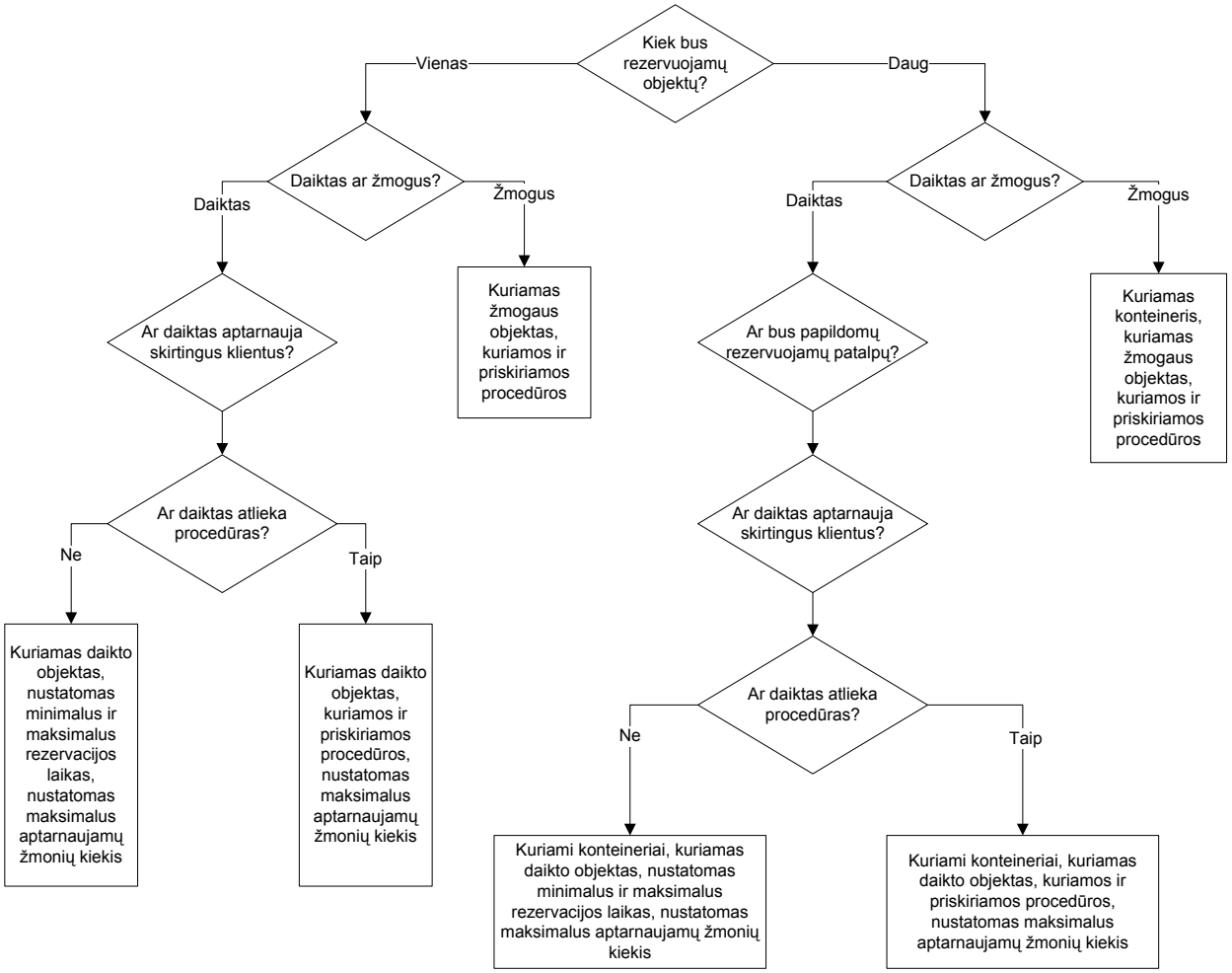
Kadangi visa sistema yra taikoma žmonėms be specifinių programavimo ar informatikos žinių, sistemos konfigūravimas irgi turi būti kuo paprastesnis ir suprantamas kuo daugiau žmonių. Sistemos neužtenka detaliai aprašyti, reikalingi žingsniai suprastinantys konfigūravimo procesą iki paties paprasčiausio.

Vienas iš būdų labai supaprastinti konfigūravimą yra iš anksto paruoštų šablonų taikymas. Sistemoje galima įvesti keletą šablonų dažniausiai pasitaikančioms situacijoms ir leisti administratoriui tą šabloną naudoti kaip pagrindą savo sistemai. Tarkim dviejų restoranų šablonas galėtų būti tas pats, o skirtųsi tik tam tikros detalės, tokios kaip darbo laikas, nuomojamų patalpų skaičius ar panašiai.

Leisti žmogui pasirinkti šabloną yra labai patogu, bet, kad sistema su šablonais būtų efektyvi, reikia sukurti daug šablonų. Žinoma pagrindinius nustatymus administratorius galės keisti ir sistemą jau įdiegus. Pavyzdžiui restorane esančią patalpą lengvai galima padaryti rezervuojama ir atvirkščiai, tačiau jei šablonas numatė, kad yra restoranas, o jame rezervuojami staliukai, pridėti atskiros rezervuojamos patalpos nebebus įmanoma.

Kad visiškai atitikti kiekvieno administratoriaus norus ir, tuo pačiu, išsaugoti sistemos konfigūravimo paprastumą labai patogu sistemos pagrindą sukurti pagal administratoriaus atsakymus į prieš tai paruoštus klausimus.

Klausymų schema:



Pav. 15 Klausimynas skirtas surinkti informacijai apie sistemos pagrindą (sudaryta autoriaus)

Administratoriui atsakius į šiuos klausimus galima toliau tęsti sistemos konfigūravimą. Pirmasis klausimas skirtas išsiaiškinti ar bus reikalingi konteineriai. Administratorius turės galimybę kurti ir redaguoti konteinerius tik jei atsakinėdamas į klausimus pasirinks variantą, kad rezervuojamų objektų bus daug, nes akivaizdu, kad konteinerių nereikia vienam objektui.

Kadangi rezervuojamo žmogaus objektas gali atlikti tik tam tikras procedūras, klausimas ar rezervuojamas objektas bus daiktas ar žmogus leidžia nustatyti netik ar bus naudojamos būtent procedūros, bet ir ar gali būti galimybė, kad reikės kurti rezervuojamus rezervuojamų objektų konteinerius, bei ar paties aukščiausio lygio konteineris gali būti rezervuojamas.

1.8. Automatinis tvarkaraščių sudarymas.

1.8.1. „Greedy“ tipo algoritmai

„Greedy“ (liet. gobšusis) algoritmas yra bet koks algoritmas kuris vadovaujasi problemos sprendimo meta-euristika pagal kurią renkasi lokalius optimalius pasirinkimus, kiekvienoje sprendimo fazėje, tikėdamasis atrasti globalųjį optimumą. Sprendžiant keliaujančio pirklio

uždavinį „Greedy“ algoritmas atrodytų taip: „Kiekvienoje stadijoje aplankyti nelankytą miestą esantį arčiausiai miesto kuriame yra pirklys“ (Ponomarev D., 2007).

Dažniausiai „Greedy“ algoritmas turi penkis skiriamuosius bruožus:

1. Kandidatų masyvą iš kurių bus sudaromas sprendinys;
2. Funkciją kuri atrenka kurie kandidatai bus pridėti prie sprendinio;
3. Tinkamumo apskaičiavimo funkciją, kuri yra naudojama nustatyti kiek kandidatas tinkamas sprendimui;
4. Funkciją įvertinančią galutinio atsakymo gerumą;
5. Sprendinio funkciją kuri nurodo kada mes atradome sprendimą.

„Greedy“ algoritmai yra tinkami spręsti kai kurioms matematinėms problemoms. Dauguma problemų kurių „Greedy“ algoritmai išspręsti negali pasižymi dviem savybėmis (Matuszek D. 2007):

- Gobšaus pasirinkimo savybė - mes galim daryti bet kurį pasirinkimą kuris atrodo tinkamiausias einamu momentu ir išspręsti iškilusias sub-problemas vėliau. Sprendimas padarytas „Greedy“ algoritmo gali priklausyti nuo pasirinkimų kuriuos mes padarėme bet ne nuo sprendimu kuriuos galėsime priimti ateityje. Algoritmas iteruodamas priima vieną „gobšų“ pasirinkimą po kito taip mažindamas problemą į vis mažesnę. Kitaip sakant „Greedy“ algoritmas niekada nekeičia padarytų pasirinkimų. Tai yra didžiausias skirtumas nuo dinaminio programavimo, kuris yra nuodugnus ir garantuoja sprendimo radimą. Po kiekvienos stadijos dinaminio programavimo algoritmai daro sprendimus priklausomai nuo visų prieš tai priimtų sprendimų. Esant reikalui šie algoritmai gali koreguoti prieš tai priimtus sprendimus taip keisdami algoritmo kelią iki sprendimo.
- Optimali sub-struktūra – „Problema turi optimalią sub-struktūrą jei optimalus problemos sprendimas turi optimalų sprendimą sub-problemai“. Kitaip tariant problema turi optimalią sub-struktūrą jei geriausias kitas ėjimas visada veda prie optimalaus sprendimo.

1.8.2. „Bin-packing“ algoritmas

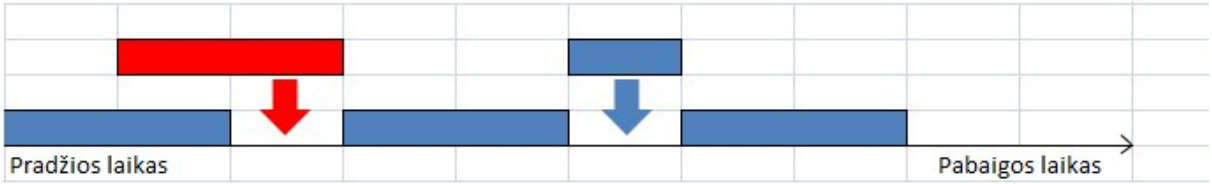
Algoritmų teorijoje, „Bin-packing“ (dėžės pakavimo) problema yra kombinatorinė „NP-hard“ (angl. non-deterministic polynomial-time hard) tipo problema. Šio tipo uždaviniuose skirtingo dydžio objektai turi būti sutalpinti į baigtinį skaičių dėžių kurių dydis yra V, taip, kad minimizuotų sunaudotų dėžių kiekį. Yra daug dėžių pakavimo uždavinių atmainų, tokių kaip: 2D pakavimas, linijinis pakavimas, pakavimas pagal svorį, kainą ir t.t. Šis uždavinys padeda

situacijose kai reikia apskaičiuoti optimalius būdus užpildyti konteinerius, pakrauti automobilius su atitinkamu svorio apribojimu, failų kopijų išdėstymą, mūsų atveju - laiko paskirstymą ir t.t.

Yra specifinis atsargų ribojimo uždavinys. Dėžių skaičius yra apribojimas iki vienos, o kiekvienas objektas yra charakterizuojamas dydžiu ir verte. Uždavinys kai reikia maksimizuoti sutalpintų objektų vertę yra žinomas kaip „Knapsack“ (liet. kuprinės) problema.

Nepaisant fakto, kad ši problema yra NP-hard, optimalūs sprendimai gali būti rasti pasitelkus atitinkamus algoritmus. Yra nemažai euristinių sprendimų, tokių kaip pirmo tilpimo (angl. first fit) algoritmas, greitas, tačiau dažniausiai neoptimalus sprendimas kuris talpina objektą į pirmą dėžę kurioje jis telpa. Šiam algoritmui reikia $\Theta(n \log n)$ laiko (Chazelle B. 1983), kur n yra elementų, kuriuos reikia sutalpinti, kiekis. Prieš talpinant elementus į dėžę juos galima surikiuoti pagal atitinkamas savybes. Tarkim mūsų atveju, elementus galima išrikiuoti pagal: pradžios laiką, pabaigos laiką, trukmę. Tai negarantuoja optimalaus sprendimo, o ilgiems sąrašams gali net pailginti algoritmo veikimo laiką, tačiau atlikus šiuos veiksmus galima dramatiškai pagerinti rezultatus.

1.8.2.1. 1D „Bin-packing“

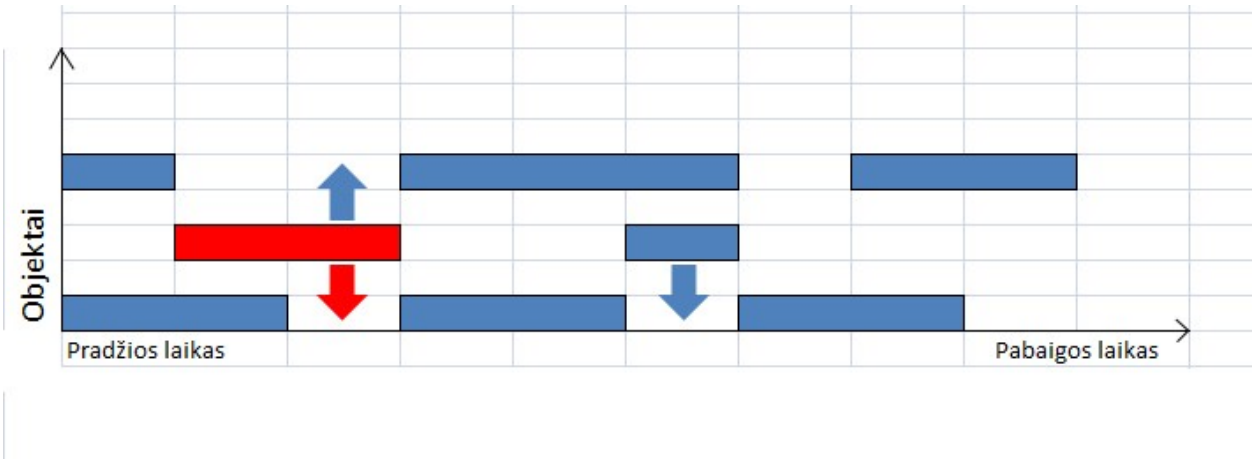


Pav. 16 1D „Bin-packing“ intervalų įterpimo pavyzdys (sudaryta autoriaus)

Šiuo atveju problema yra sutalpinti visus laiko intervalus taip, kad jie nesikirstų x ašyje. Tai gali būti pobūvių salės, dantisto kabineto ir kitų situacijų, schema. Paveikslėlyje matyti, kad 1D modelyje intervalui esant per ilgam jis turi būti atmetamas.

1.8.2.2. 2D „Bin-packing“

Esant daugiau nei vienam rezervuojamam objektui intervalas kuris kertasi su vieno objekto rezervuotu laiku gali būti suderinamas su kito objekto užsakymais. Tai gali būti restorano, kirpyklos, soliariumo ir kitų situacijų schema.



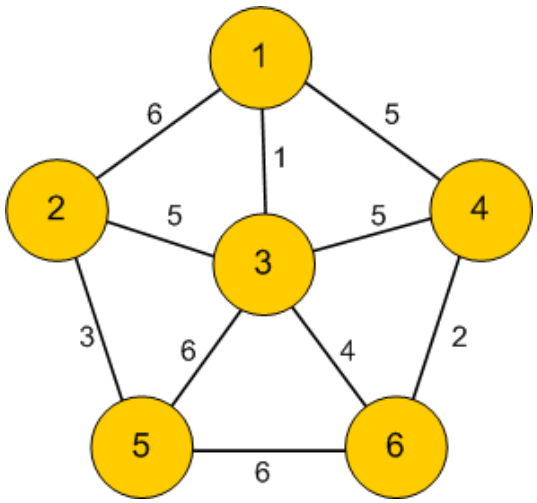
Pav. 17 2D „Bin-packing“ intervalų įterpimo pavyzdys (sudaryta autoriaus)

1.8.3. Grafai

Grafas tai abstrakti duomenų aibės reprezentacija kurioje objektai turi ryšį tarpusavyje. Grafai susideda iš viršūnių (angl. vertex) ir juos jungiančių briaunų (angl. edge). Briaunos gali būti vienkryptės (asimetrinės) ir dvikryptės (simetrinės arba nekryptinės). Grafais galima atvaizduoti labai daug ką: kelius tarp miestų, tinklo maršrutizatorius ir netgi tvarkaraščius. Yra kelios grafų rūšys: išbaigti, jungūs, kryptiniai ir t.t.

Grafu galime vadinti išrikiuotą porą $G = (V, E)$ sudarytą iš V viršūnių masyvo ir briaunų masyvo E. V ir E dažniausiai yra baigtiniai. Grafo laipsnis yra $|V|$ (viršūnių skaičius). Grafo dydis yra $|E|$ (briaunų skaičius). (Diestel R., 2010)

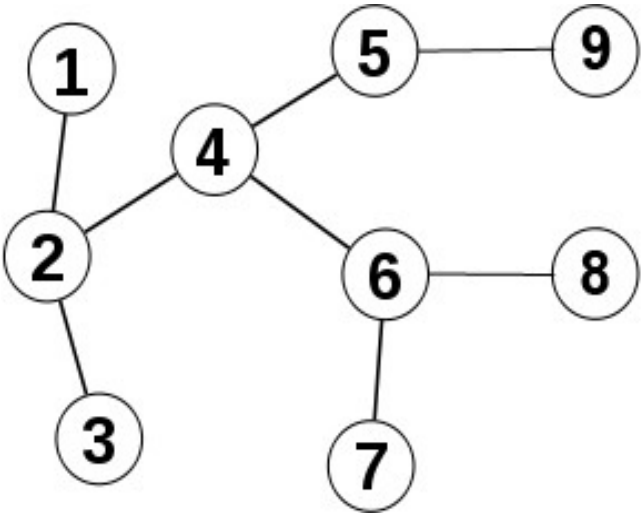
Dvikryptis grafas $G = (V, E)$ turintis šešias viršūnes $V = \{1, 2, 3, 4, 5, 6\}$ ir devynias briaunas $E = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 5\}, \{5, 3\}, \{5, 6\}, \{6, 3\}, \{6, 4\}\}$ pavaizduotas aštuonioliktame paveikslėlyje.



Pav. 18 Dvikryptis, jungus, išbaigtas grafas su svoriais (Hadorn B. – „Cell Computing Model or Zeus-Framework“ 2011)

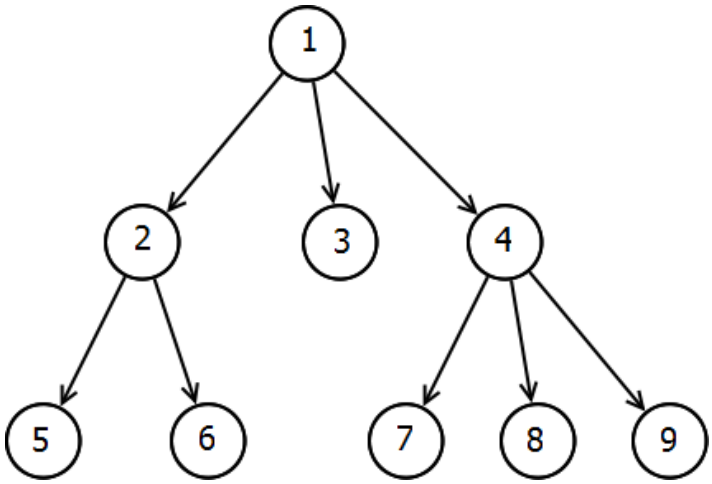
1.8.3.1. Medžiai

Grafų teorijoje ir matematikoje, medis yra nekryptinis (dvikryptis) grafas, kuriame bet kurios dvi viršūnės yra sujungtos lygiai viena briauna. Kitaip tariant bet kuris jungus grafas neturintis ciklų yra medis. Miškas yra grafas sudarytas iš vieno ar kelių medžių.



Pav. 19 Medis pagal grafų teorijos apibrėžimą (sudaryta autoriaus)

Įvairios duomenų struktūros, kurios kompiuterių moksle vadinamos medžiais, yra labai panašios į medžius grafų teorijoje, tačiau daugelyje tų duomenų struktūrų medžiai turi vienkryptes briaunas. Nors tokie medžiai neatitinka grafų teorijos medžio apibrėžimo, tačiau jie yra vadinami išrikiuotais kryptiniais medžiais.



Pav. 20 Duomenų struktūra medis (Barus P. N. – „Data Structures - Tree“ 2010m.)

Medis yra nekryptinis, paprastas grafas *G* jeigu jis tenkina bet kurią iš sekančių sąlygų (Diestel R., 2010):

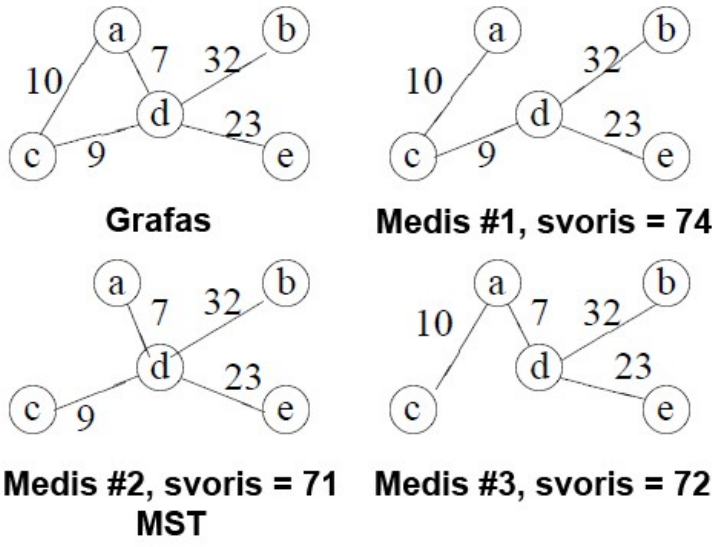
- *G* yra jungus ir neturi ciklų;
- *G* neturi ciklų ir paprastas ciklas yra suformuojamas jei nors viena briauna yra pridedama į *G*;

- *G* yra jungus, tačiau nustoja būti jungiu jei yra pašalinama nors viena briauna;
- *G* yra jungus ir trijų viršūnių išbaigtas grafas K_3 nėra grafo *G* minoras. Grafų teorijoje nekryptinis grafas *H* yra vadinamas grafo *G* minoru jeigu *H* yra izomorfiškas grafui kuris gali būti išgautas nulio ar vienos briaunos sutraukimu.
- Medis su N viršūnių visada turi N-1 briauną;
- Tarp bet kurių dviejų medžio viršūnių egzistuoja unikalus kelias.

1.8.3.2. Minimalus jungus medis

Duotam jungiam, nekryptiniam grafui, jungus medis yra sub-grafas kuris yra medis ir kuris jungia visas viršūnes. Jeigu grafas turi svorius, tam grafui galima surasti minimalųjį jungųjį medį (angl. Minimum Spanning Tree – „MST“). Minimalus jungus medis netik sujungia visas medžio viršūnes, bet ir minimizuoja visų grafo jungčių svorių sumą. Kitaip sakant minimalaus jungaus medžio briaunų suma yra mažesnė arba lygi kitų to grafo medžių briaunų sumoms. Vienas grafas gali turėti daug jungių medžių (Cherition D. ir Tarjan R. E., 1976).

Bet koks nekryptinis grafas turi minimalųjį jungųjį mišką. Grafui nebūtina būti jungiam. Minimalus jungus miškas yra minimalių jungių medžių sąjunga.



Pav. 21 Grafas, jo jungūs medžiai ir minimalus jungus medis (sudaryta autoriaus)

2. Praktiniai tyrimai

Praktinėje dalyje buvo įgyvendinta situacija reikalaujanti rezervavimo bei sistemos konfigūravimo procesas. Tai pat buvo suprogramuota porą „Bin-packing“ euristikų. Galiausiai buvo nagrinėjama galimybė pritaikyti minimalaus jungaus medžio paieškos algoritmus tvarkaraščių sudarymui.

2.1. Grožio salonas

Įgyvendinti rezervavimo reikalaujančią situaciją buvo nuspręsta dėl to, kad reikėjo išsiaiškinti kokių duomenų sistemai reikės ir būtų galima surinkti daugiau informacijos, kaip sistema tūrėtų kurti ir konfigūruoti objektus. Galiausiai, sudarytus algoritmus galima panaudoti galutinėje sistemoje.

Situaciją kurią pasirinkau įgyvendinti yra grožio salonas. Grožio salono savybės:

1. Grožio salone dirba daug darbuotojų.
2. Grožio salone darbuotojai atlieka procedūras.
3. Procedūros priklauso procedūrų grupėms.

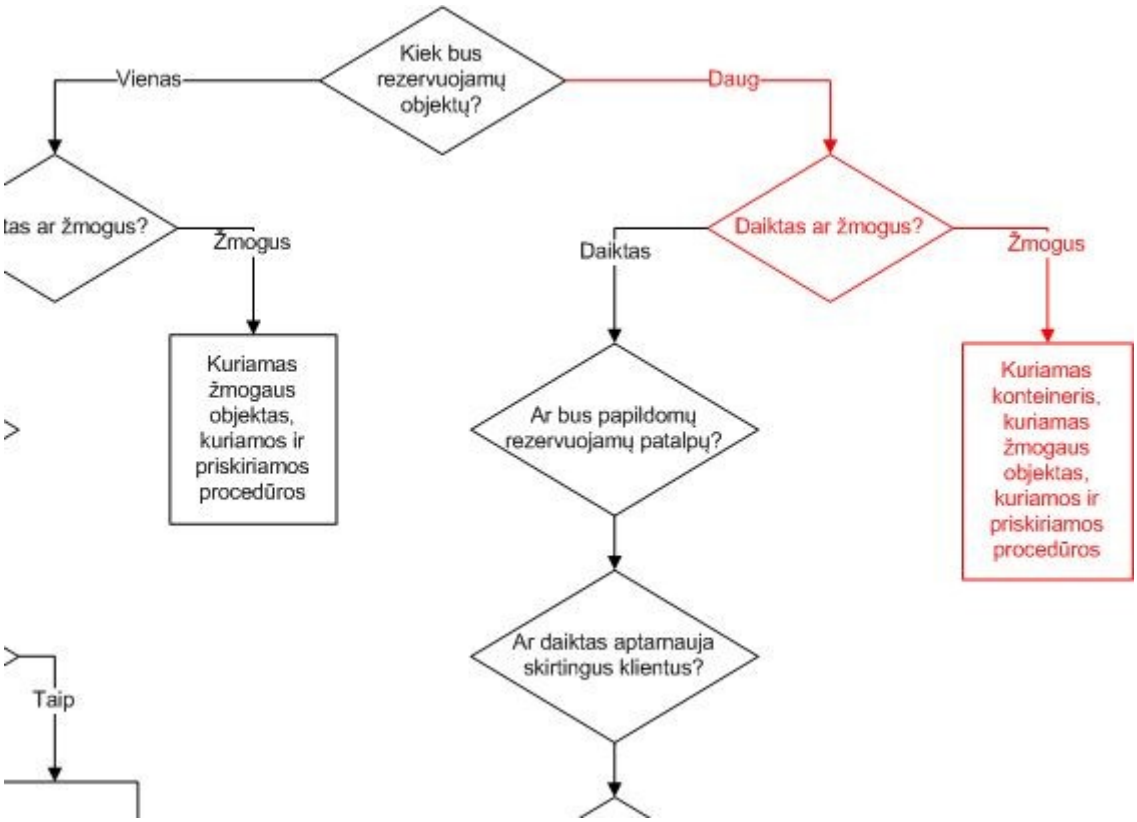
Procedūrų grupes buvo nuspręst atskirti, nes vartotojai dažnai nori suskirstyti siūlomas paslaugas į logines grupes, kurios gelbsti vartotojui greičiau rasti reikiamą informaciją.

Paveikslėlyje (žr. 22 pav.) pavaizduotas kelias kurį, kuriant sistemą, turėtų nueiti vartotojas ir po kurio būtų sukurtas pagrindas reikalingas:

1. Konteinerio redagavimui;
2. Procedūrų, ir šiuo atveju procedūrų grupių, kūrimui ir redagavimui;
3. Darbuotojų kūrimui ir redagavimui;
4. Rezervacijų administravimui;

Iš vartotojo pusės rezervavimas vyksta penkiais žingsniais:

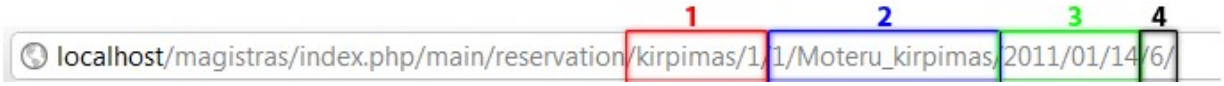
1. Vartotojas pasirenka procedūrų grupę;
2. Vartotojas pasirenka procedūrą;
3. Vartotojas pasirenka rezervacijos datą;
4. Vartotojas pasirenka norimą darbuotoją (jei vartotojas nori, darbuotojas gali būti parenkamas automatiškai);
5. Vartotojas suveda informaciją apie save ir baigia rezervaciją.



Pav. 22 Kelias kurį reikėtu nueiti kuriant ir konfigūruojant sistemą (sudaryta autoriaus)

2.1.1. Rezervavimas

Viso rezervacijos proceso metu, pagrindinę informaciją sistema pasiekia iš URL adreso kuris automatiškai generuojamas ir atnaujinamas po kiekvieno žingsnio.



Pav. 23 Rezervacijos pabaigoje sugeneruotas visas URL adresas (sudaryta autoriaus)

Kaip matyti URL saugoma informacija apie keturis žingsnius per kuriuos žmogus pasirenka rezervacijos duomenis. URL adresas susideda iš:

1. Procedūrų grupės pavadinimo kuris yra paimamas iš duomenų bazės ir kiekvieną sykį modifikuojamas taip, kad neturėti jokių specialių ženklų. Po procedūrų grupės pavadinimo eina pasvirasis brūkšnelis, o po jo procedūrų grupės unikalus identifikatorius.
2. Procedūros unikalus identifikatorius ir, po pasvirojo brūkšnelio, jos pavadinimas. Procedūros pavadinimas, kaip ir procedūrų grupės pavadinimas, yra paimamas iš duomenų bazės ir išvalomas nuo nepageidaujamų simbolių.
3. Lietuviško formato (yyyy/MM/dd - ISO 8601) data.
4. Unikalus darbuotojo, kurį pasirinko vartotojas, identifikatorius.

Rezervacija prasideda nuo to, kad vartotojas pasirenka procedūrų grupę, kuriai priklauso, norima rezervuoti procedūra. Tada sistema automatiškai nukreipia vartotoją tam tikru adresu. Po

kiekvieno rezervacijos žingsnio sistema patikrina ar neatsirado papildomos informacijos URL adrese.

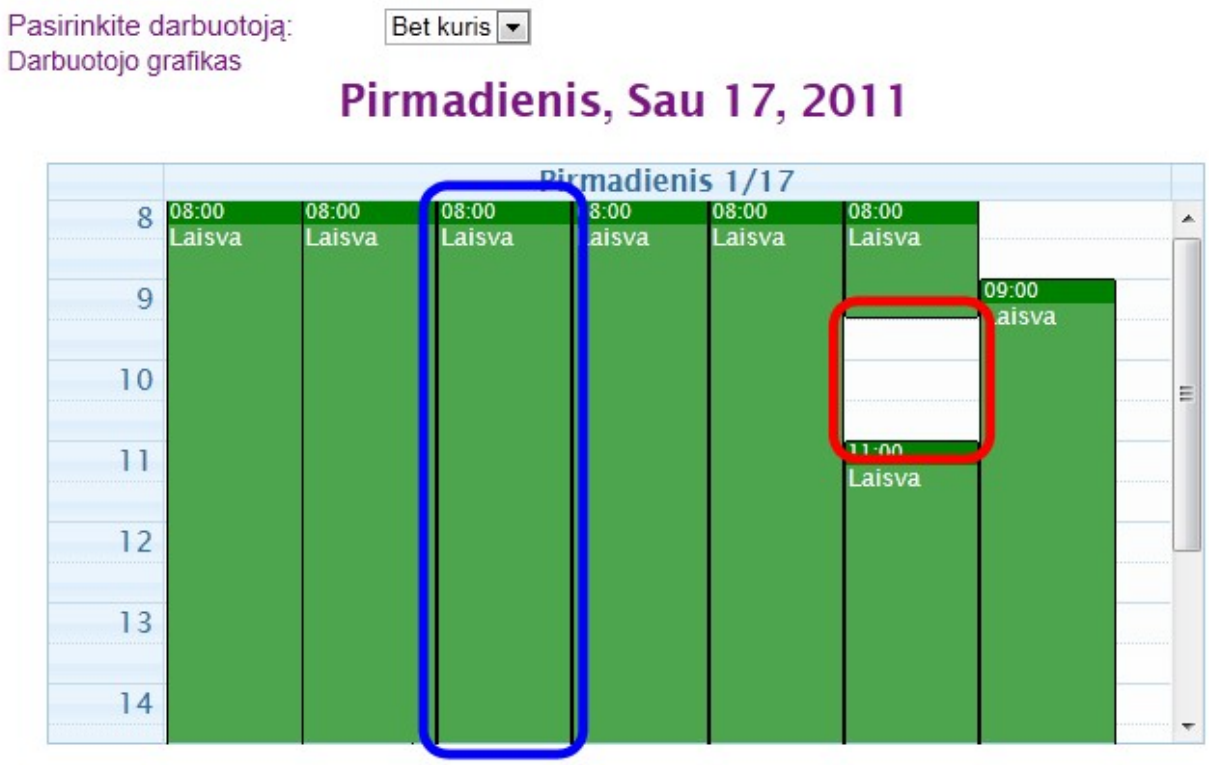
Po to kai yra pasirenkama procedūrų grupė ir vartotojas yra nukreipiamas nauju adresu, sistema aptinka naują informaciją ir pagal ją generuoja tolimesnius žingsnius. Pasiėmus grupės unikalų identifikatorių sistema atrenka tik tas procedūras kurios priklauso tai grupei.

Vėliau, pasirinkus procedūrą, sistema generuoja ir užpildo kalendorių reikiama informacija. Šioje vietoje buvo pasirinkta vartotojui leisti rinktis datą šešis mėnesius į priekį, tačiau galutinėje sistemoje šį kiekį galima leisti nusistatyti pačiam vartotojui. Kiekviena data turi nuvesti vartotoją unikaliu adresu skirtu būtent tai datai (žr. 24 pav.).

Pav. 24 Sugeneruotas kalendorius su unikaliais datos parametrais (sudaryta autoriaus)

Pasirinkus atitinkamą datą vartotojo nereikalaujama rinktis tam tikro darbuotojo, kad tęsti rezervaciją. Pirmasis pasiūlymas pateiktas rezervaciją atliekančiam asmeniui yra „Bet kuris darbuotojas“. Pasirinkus datą sistema atkreipia dėmesį į paskutini adreso segmentą ir jeigu jis yra lygus nuliui tada vartotojui siūlomas „Bet kurs darbuotojas“. Taip pačiai kaip ir su procedūromis, darbuotojų sąrašas yra generuojamas pagal tai ar jie sugeba atlikti procedūras.

Kad rezervaciją atliekančiam vartotojui būtų paprasčiau, sistema, priklausomai nuo situacijos, pateikia informaciją apie darbuotojų užimtumą. Jeigu vartotojui nesvarbus jį aptarnausiantis darbuotojas, sistema rodo visų darbuotojų, gebančių atlikti procedūrą, laisvo laiko intervalus (žr. 25 pav.).



Pav. 25 Laiko užimtumo grafikas kai vartotojas nepasirenka darbuotojo (sudaryta autoriaus)

Jeigu vartotojas dar nepasirinko jam reikiamo žmogaus sistema parodo visų darbuotojų užimtumo grafiką, kuris leidžia žmogui įvertinti laiką kada jam vertėtų rezervuoti paslaugas. Mėlynai apvestas laiko intervalas yra vieno darbuotojo neužimtas laiko intervalas. Raudonai apvestas intervalas yra jau rezervuotas laikas.

Vartotojui pasirinkus norimą darbuotoją rodomos tik rezervacijos kurių metu darbuotojas yra užimtas. Dvidešimt šeštame paveikslėlyje pavaizduotas tas pats, raudonai pažymėtas, užimtas, intervalas iš dvidešimt penkto paveikslėlio.

Pav. 26 Laiko užimtumo grafikas kai vartotojas pasirenka darbuotoją (sudaryta autoriaus)

Kad sudaryti laisvų laikų grafiką, sistema pradžioje išrenka visus darbuotojus kurie atlieką pasirinktą procedūrą. Iš eilės pereidama visus darbuotojus ji surenka informaciją apie visas rezervacijas su kuriomis susijęs darbuotojas. Informacija yra išrikiuojama pagal procedūrų pradžios laiką. Pirmiausiai sistema patikrina ar yra tarpas tarp pirmos procedūros ir darbuotojo darbo pradžios laiko, tada patikrinama ar yra laisvas tarpas tarp paskutinės procedūros ir darbuotojo darbo pabaigos laiko, o jau tada yra tikrinama ar yra laisvų tarpų tarp visų iš eilės einančių procedūrų.

Su atskiru darbuotoju yra daug paprasčiau, nes sistema tiesiog atvaizduoja visas rezervuotas to darbuotojo procedūras.

Toliau sistema patikrina ar vartotojas teisingai suvedė visus duomenis. Vardo, pavardės, telefono, elektroninio pašto, ir, saugumo sumetimais, CAPTCHA, laukai yra privalomi įvesti. Elektroninio pašto adresas būtinai privalo turėti @ ženklą ir atitikti formatą vartotojas@tiekėjas.lt. Telefono numeris dar turi būti ir atitinkamo ilgio, o CAPTCHA laukas turi sutapti su sistemos sugeneruotu kodu.

Jeigu duomenys praeina patikrą, sistema žiūri kokį darbuotoją pasirinko vartotojas. Jeigu vartotojas pasirenka darbuotoją, sistema surenka informaciją apie rezervacijas kurias jis atlieka ir tikrina ar nors vienos rezervuotos procedūros laikas kertasi su laiku kurį nori rezervuoti vartotojas. Tuomet sistema tikrina ar rezervuojamas laikas nesikerta su darbuotojo darbo laiku. Jeigu tenkinamos šios dvi sąlygos, tada informacija apie rezervaciją ir vartotoją yra keliama į duomenų bazę ir vartotojas yra įspėjamas apie sėkmingą rezervaciją. Jeigu nors viena iš sąlygų nėra tenkinama vartotojui pranešama kodėl jo rezervacija nepavyko.

Jei vartotojas nepasirenka konkretaus darbuotojo, sistema viską atlieka panašiu principų kaip tai būtų jeigu vartotojas būtų pasirinkęs darbuotoją. Esminis skirtumas tas, kad pradžioje sistema išrenka visus darbuotojus galinčius atlikti procedūrą ir išrikiuoja juos atsitiktine tvarka. Tuomet sistema iš eilės tikrina visus darbuotojus, ar jie yra laisvi rezervacijos metu ir ar rezervacija nesikerta su jų darbo laiku. Suradus pirmą kandidatą kuris atitinka abu reikalavimus, reikiama informacija yra saugoma duomenų bazėje. Jeigu tinkamų kandidatų nerasta, vartotojas yra įspėjamas, kad turėtų mėginti kitą laiką.

2.1.2. Administravimas

Sistemai administruoti sukūriau keturis pagrindinius modulius:

1. Procedūrų kūrimas, redagavimas ir administravimas;
2. Darbuotojų kūrimas, redagavimas ir administravimas;
3. Rezervacijų administravimas;
4. Atostogų administravimas;

Procedūrų modulyje leidžiama kurti ir redaguoti procedūras, kurias atliekant rezervaciją, mato ir gali pasirinkti vartotojas. Leidžiama redaguoti procedūros pavadinimą, aprašymą ir trukmę.

Kuriant naują ar redaguojant darbuotoją galima nustatyti jo darbo laiką, bei priskirti jau sukurtas procedūras prie darbuotojo atliekamų procedūrų sąrašo. Taip pat redaguojant darbuotoją galima jį visiškai pašalinti iš sistemos.

Rezervacijų administravimas susideda iš trijų funkcijų:

1. Naujos rezervacijos kūrimas;
2. Rezervacijos atšaukimas;
3. Informacija apie visas rezervacijas.

Naujos rezervacijos kūrimas administratoriaus pusėje atrodo labai panašiai į vartotojo pusės rezervavimą, o pašalinimas yra tiesiog įrašo iš duomenų bazės šalinimas.

Atostogų modulis, kol kas, labai primityvus. Administratoriui tenka rezervuoti visas dienas, kada darbuotojas nori atostogauti, atskirai. Tačiau tokiu būdu administratorius darbuotojui gali suteikti ne visą laisvą dieną, o kažkokį laiko intervalą.

2.2. Laiko rezervavimo algoritmų praktinis tyrimas

Optimalaus laiko rezervavimo algoritmo galima ieškoti pagal du pagrindinius kriterijus:

1. Didžiausią įvykdomų užduočių kiekį;
2. Optimaliausiai išnaudojamą laiką.

Teoriškai, pirmuoju atveju, bet koks „Greedy“ tipo algoritmas gali gauti optimalų sprendimą, jeigu pradinė intervalų aibė bus išrikiuota pagal intervalų pabaigos laiką, todėl visi atlikti bandymai yra orientuoti į optimaliai išnaudojamo laiko paieškos algoritmų išskyrimą.

Bandymus atlikinėjau su dviem kandidatų aibėmis. Pirmoji aibė turi tik rezervacijas tarp kurių visai nėra tarpų, kitaip sakant viena rezervacija eina iškart po kitos. Antroji aibė yra mišri, kai kurios rezervacijos seka viena paskui kitą, o kai kurios turi atliekamo laiko tarp jų.

1 lentelė. **Pirmoji duomenų aibė.**

Id	Pradžios laikas	Pabaigos laikas	Trukmė
1.	14:00	15:00	1
2.	9:00	10:30	1:30
3.	12:00	13:00	1
4.	13:00	14:00	1
5.	9:30	12:00	1:30
6.	15:00	17:00	2

1 lentelės tęsinys. **Pirmoji duomenų aibė.**

Id	Pradžios laikas	Pabaigos laikas	Trukmė
7.	16:00	17:00	1
8.	8:00	9:30	1:30
9.	10:30	12:30	2
10.	12:30	14:30	2
11.	14:30	16:00	1:30
12.	8:00	9:00	1

2 lentelė. Antroji aibė

Id	Pradžios laikas	Pabaigos laikas	Trukmė
1.	16:00	17:00	1
2.	9:30	11:30	2
3.	11:30	12:30	1
4.	8:00	9:00	1
5.	15:30	16:30	1
6.	12:00	13:00	1
7.	10:00	11:00	1
8.	12:30	14:30	2
9.	8:30	9:30	1
10.	14:00	15:00	1
11.	13:30	14:00	0.5
12.	15:00	16:00	1
13.	16:30	17:00	0.5

2.2.1. Praktiniai „Bin-packing“ tyrimai

Laiko rezervavimo analizei pasirinkau praktiškai įgyvendint „Bin-packing“ algoritmą kaip vieną iš paprastesnių ir kaip kontrastą sudėtingam grafų ir medžių algoritmui.

Algoritmui reikalingi kintamieji:

- *E* – Visų laiko intervalų aibė
- S – Sprendinių aibė, kuri pradžioje yra tuščia

while $\exists\, i \in E$

pasirenkam $i \in E$

pridedam i į S aibę

iš aibės E šalinam visus laiko intervalus kurie kertasi su laiko intervalu i

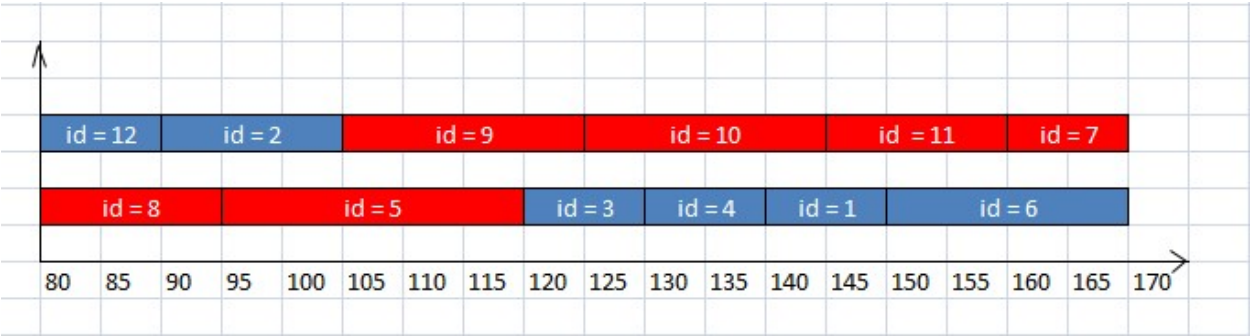
return S

Pav. 27 „Bin-packing“ algoritmo laiko intervalams pseudo-kodas (Chazelle B., 1983)

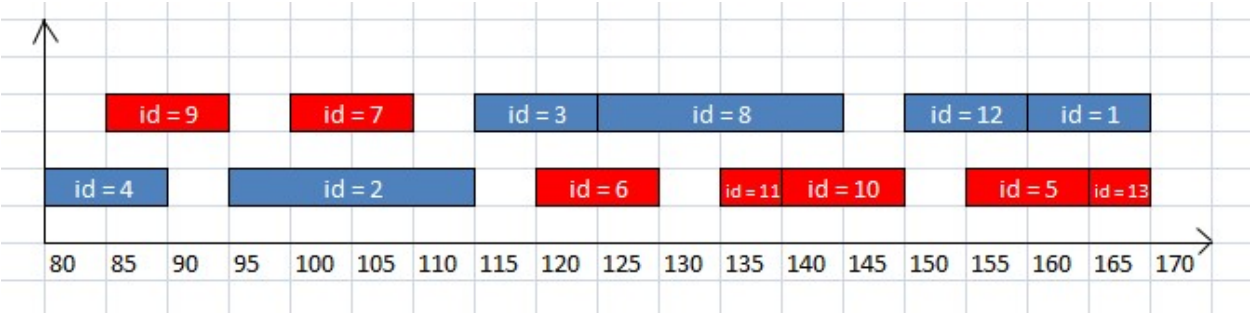
2.2.2. „Bin-packing“ tyrimų rezultatai

Bandymam didelės įtakos turi duomenų išrikiavimas, dėl to abejose aibėse esantys duomenys yra išrikiuoti atsitiktinai, o bandymai buvo atlikti su duomenimis išrikiuotais:

1. Atsitiktinai.
2. Pagal anksčiausią intervalų pradžios laiką.
3. Pagal anksčiausią intervalų pabaigos laiką.
4. Pagal intervalų ilgius.

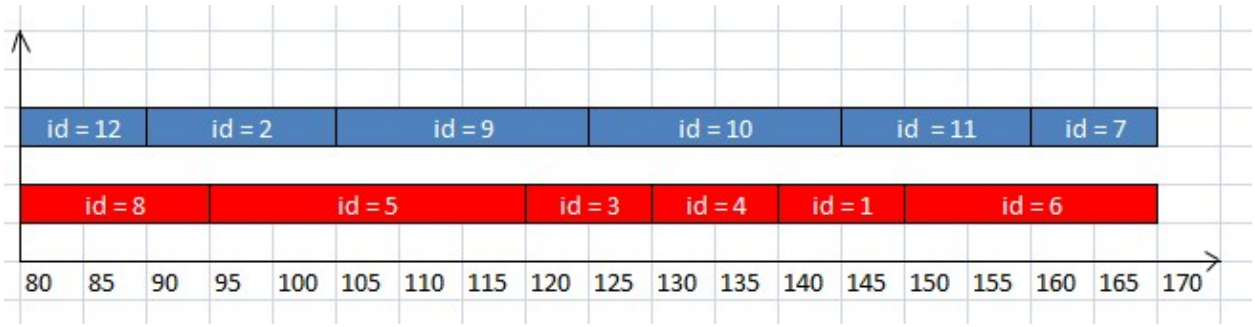


Pav. 28 Pirmoji duomenų aibė – atsitiktinis rikiavimas (sudaryta autoriaus)

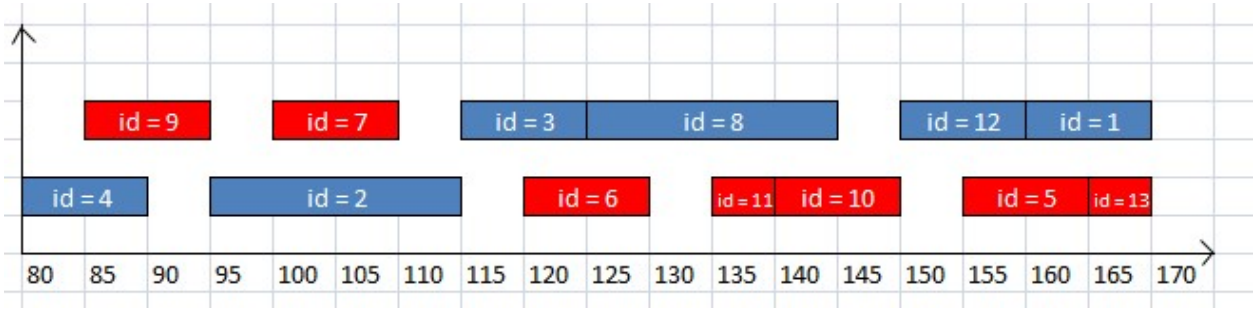


Pav. 29 Antroji duomenų aibė – atsitiktinis rikiavimas (sudaryta autoriaus)

Paveikslėliuose matyti gauti rezultatai kai duomenų aibė yra išrikiuota atsitiktinai. Mėlyni laukai žymi laiko intervalus kuriuos pasirinko algoritmas. Stebėtinai, su antrąja duomenų aibe gautas rezultatas gavosi optimalus, tačiau pirmuoju atveju ne. Atsakymų aibės laiko suma pirmuoju atveju – septynios su puse valandos, antruoju – aštuonios.

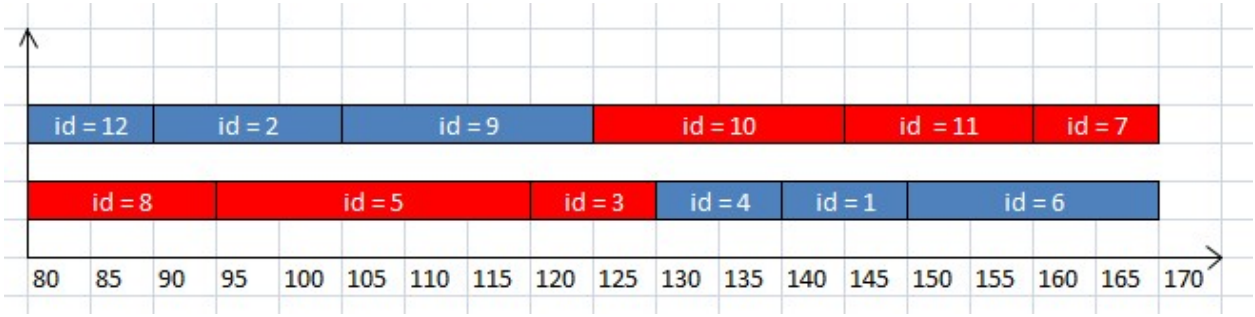


Pav. 30 Pirmoji duomenų aibė - rikiavimas pagal anksčiausią intervalų pradžios laiką (sudaryta autoriaus)

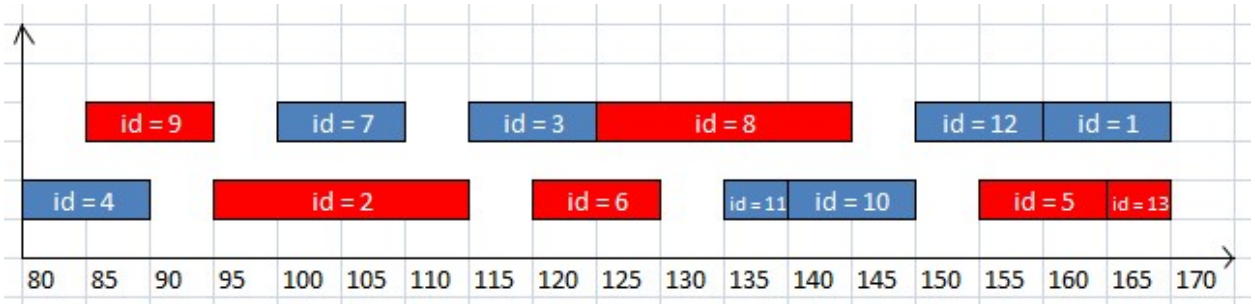


Pav. 31 Antroji duomenų aibė – rikiavimas pagal anksčiausią intervalų pradžios laiką (sudaryta autoriaus)

Išrikiavus duomenis pagal anksčiausią intervalų pradžios laiką abiejų duomenų aibių atvejais pasiekti optimalūs rezultatai. Pirmosios aibės atsakymų suma – devynios, antro – aštuonios valandos.

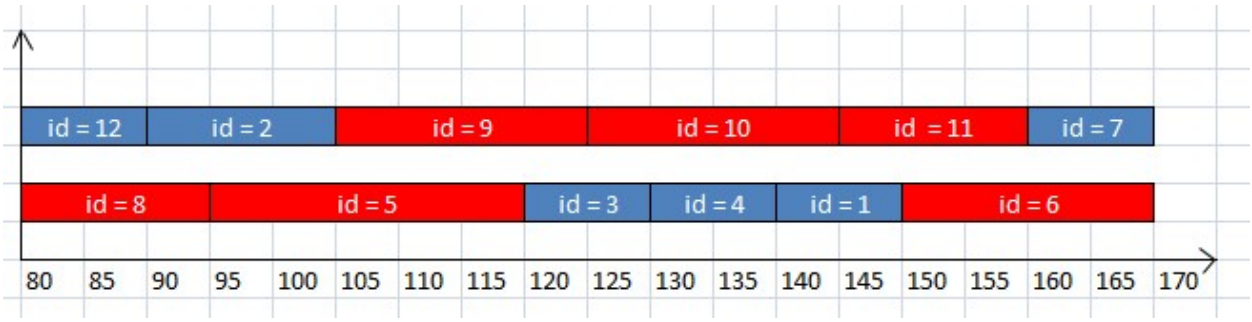


Pav. 32 Pirmoji duomenų aibė – rikiavimas pagal anksčiausią intervalų pabaigos laiką (sudaryta autoriaus)

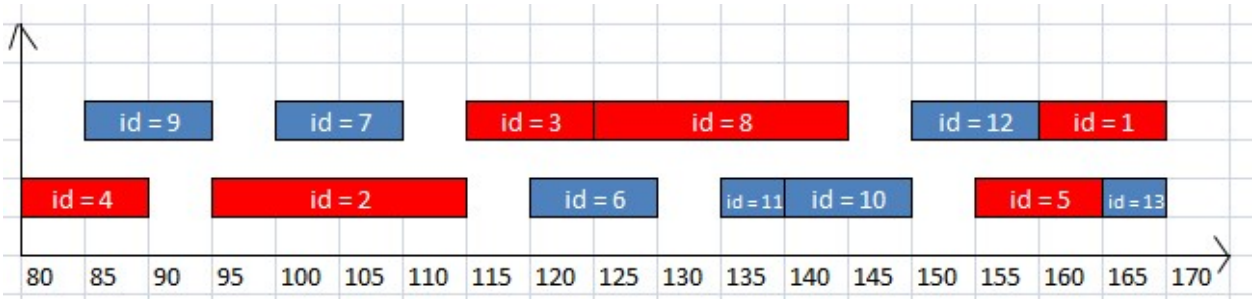


Pav. 33 Antroji duomenų aibė – rikiavimas pagal anksčiausią intervalų pabaigos laiką (sudaryta autoriaus)

Deja šiuo atveju nei pirmajai nei antrajai aibei gautos atsakymų aibės nebuvo optimalios. Pirmajai aibei rezervuoto laiko trukmė buvo aštuonios su puse valandos, o antruoju septynios su puse. Šios euristikos gauti atsakymai buvo labai arti optimalaus sprendimo.



Pav. 34 Pirmoji duomenų aibė – rikiavimas pagal trumpiausią intervalą (sudaryta autoriaus)



Pav. 35 Antroji duomenų aibė – rikiavimas pagal trumpiausią intervalą (sudaryta autoriaus)

Deja šiuo atveju sistema atliko beveik priešingą darbą. Gautas atsakymas abiem atvejais yra bene prasčiausia galima intervalų aibė.

Iš gautų rezultatų matyti, kad geriausiai pasirodė „Greedy“ metodas su duomenimis kurie buvo išrikiuoti pagal anksčiausią intervalų pradžios laiką, abiem aibėm gavęs optimalius rezultatus.

Testai buvo atlikti bandant rezervuoti vieną objektą, tačiau esant reikalui algoritmą lengva perorientuoti, kad jis ieškotų sprendimų keletui objektų.

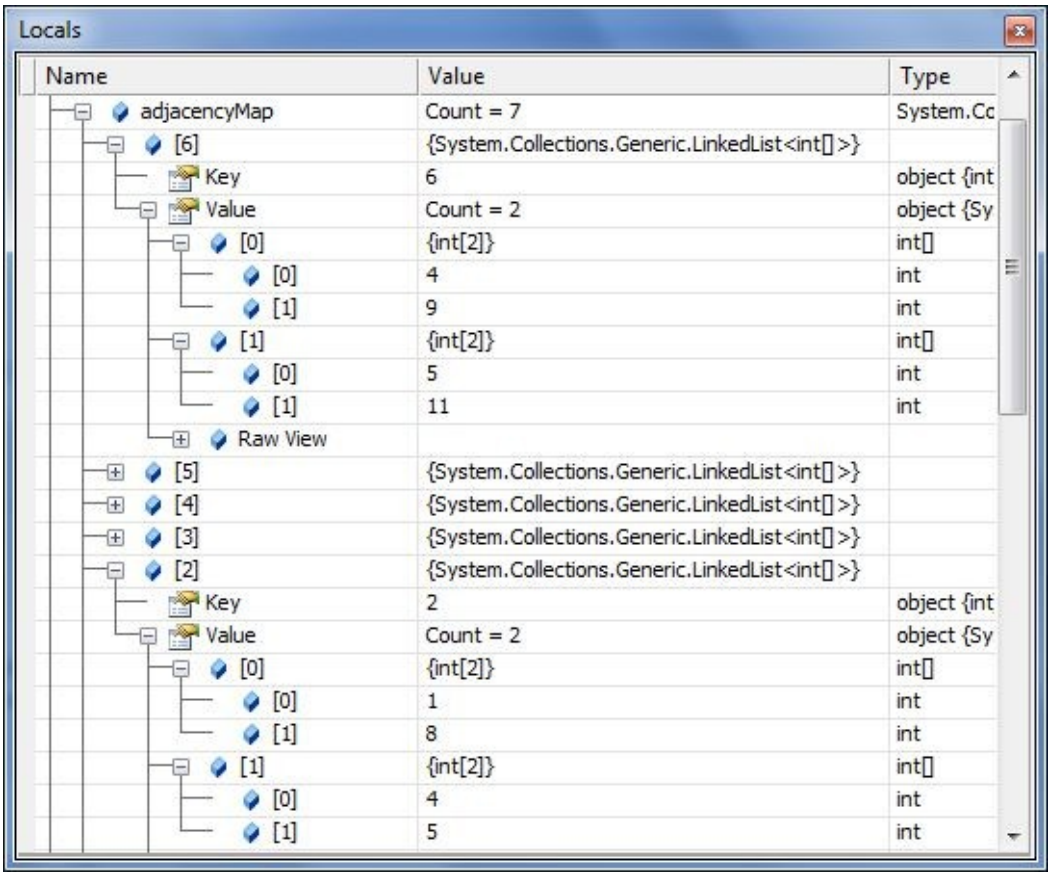
2.3. Praktinis grafų ir medžių pritaikymas tvarkaraščių sudarymui

Siekiant panaudoti grafus, o tiksliau vieną jų šaką – minimalius jungius medžius, visų pirma reikėjo įgyvendinti grafo duomenų struktūrą.

Grafą įgyvendinau pasitelkdamas Maišos lentelės (angl. Hash table) duomenų struktūra. Kaip skiriamasis lentelės raktas (angl. Key) yra naudojamas eilės numeris kuriuo atitinkamas taškas buvo įvestas. Lentelės įrašas susideda iš:

1. Skiriamojo rakto;
2. Sąrašo susidedančio iš masyvų.

Sąrašas yra sudarytas iš masyvų, saugančių viršūnes, su kuriomis jungiasi skiriamojo rakto viršūnė, ir briaunų, jungiančių tas viršūnes, svoriais.



Pav. 36 Įgyvendinta grafo struktūra (sudaryta autoriaus)

Trisdešimt šeštame paveikslėlyje matyti, kad šeštoji viršūnė turi briaunas su dvejomis kitomis viršūnėmis, ketvirta ir penkta. Briaunos tarp šeštos ir ketvirtos viršūnių svoris yra devyni, o tarp šeštos ir penktos yra vienuolika.

Grafų generavimui įgyvendinau keletą funkcijų:

3 lentelė. **Funkcijos grafo generavimui.**

Id	Funkcija	Reikalingi kintamieji
1.	addVertex	Viršūnės numeris
2.	removeVertex	Viršūnės numeris, viršūnių skaičius
3.	addEdge	Pradžios viršūnė, pabaigos viršūnė, briaunos svoris
4.	removeEdge	Pradžios viršūnė, pabaigos viršūnė
5.	getEdgeWeight	Pradžios viršūnė, pabaigos viršūnė
6.	setEdgeWeigth	Pradžios viršūnė, pabaigos viršūnė, briaunos svoris

Paminėtos funkcijos buvo naudojamos tik grafo generavimui. Visos kitos funkcijos yra reikalingos minimalaus jungaus medžio paieškai.

Vienas iš pagrindinių šio darbo uždavinių – rasti būdą kaip pritaikyti medžius laiko tvarkaraščių sudarymui. Kad tai padaryti reikėjo išspręsti dvi problemas:

1. Rasti būdą kaip transformuoti laiko intervalų aibę į grafą;
2. Modifikuoti standartinį algoritmą, kad jis rastu tinkamus ir teisingus minimalius jungius medžius.

2.3.1. Laiko intervalų aibės vertimo į grafą algoritmas

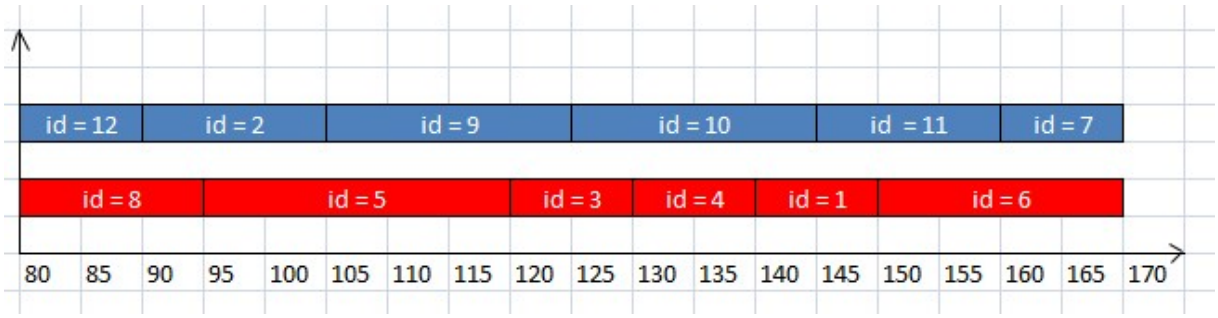
Norint sudaryti minimalųjį jungųjį medį, visų pirma, reikia turėti netuščią grafą. Logiškiausia yra laiką tarp intervalų naudoti kaip briaunų svorį. Tuomet nereikia įvesti jokio papildomo kintamojo, o laiko tarpas tarp laiko intervalų yra geriausias intervalo tinkamumo rodiklis.

Paprasčiausias būdas sudaryti grafą buvo saugoti kiekvieno intervalo ryšį su kitais intervalais. Briaunoms tarp nesikertančių intervalų galima priskirti laisvo laiko tarp tų intervalų reikšmę, o besikertančių laiko intervalų briaunoms duoti dideles reikšmes.

Tačiau šis būdas labai neefektyvus, nes esant daug trumpų intervalų bus saugoma labai daug neefektyvių briaunų. Kitaip sakant bus saugoma daug briaunų kurios niekada nepriklausys minimaliam jungiam medžiui.

Antras sprendimo būdas yra briaunomis jungti tik arčiausiai esančius intervalus. Tokiu būdu galima išvengti neefektyvių briaunų saugojimo ir sutaupyti laiko bei vietos. Tačiau šioje vietoje iškyla klausimas kiek briaunų reikėtų saugoti vienai viršūnei? Efektyviam įgyvendinimui statiškos reikšmės parinkimas nėra optimalus. Briaunų skaičius turėtų priklausyti nuo situacijos.

Panagrinėjęs įvairias situacijas priėjau išvadą, kad briaunų skaičius turėtų būti maksimalus laiko intervalų kolizijų skaičius padidintas vienetu.



Pav. 37 Nagrinėjamos aibės kolizijų pavyzdys (sudaryta autoriaus)

Iš trisdešimt penkto paveikslėlio matyti, kad maksimalus mano nagrinėjamų aibių bet kurio intervalo kolizijų skaičius yra vienas, todėl, kiekvienas laiko intervalas turi turėti po dvi briaunas. Dvi briaunos reikalingos tam, kad esant reikalui, sistema, pagal tam tikrus kriterijus, galėtų parinkti geriausią sprendimą. Taipogi tai palieka vietos algoritmo tobulinimui.

2.3.2. „Prim-Jarnik‘s“ algoritmas

„Prim-Jarnik‘s“ algoritmas yra vienas iš dviejų populiariausių minimalaus medžio paieškos algoritmų. Šio algoritmo veikimo principas yra labai paprastas.

Algoritmui reikalingi kintamieji:

- V – Visų viršūnių aibė
- E – Visų briaunų aibė
- *Viršūnė* $\in V$ (nuo jos bus pradėtas konstruoti medis)

while $V_{new} \neq$ Jungi grafo dalis
Pasirenkame briauną (u, v) su minimaliu svoriu ir tokią, kad $u \in V_{new}$, o $v \notin V_{new}$
Pridedame v į V_{new}, o briauną (u, v) į E_{new}
return V_{new} ir E_{new}

Pav. 38 „Prim-Jarnik's" algoritmo pseudo-kodas (Prim R. C. 1957)

Šiuo algoritmu medis konstruojamas nuo pasirinktos viršūnės. Algoritmas tada suranda viršūnę kurį yra mažiausio svorio ir yra jungi su naujai konstruojamu medžiu.

Transformavus laiko intervalų aibę sudaryti grafą ir sukonstruoti minimalųjį jungų medį buvo gana paprasta. Logiškiausia šiuo algoritmu konstruojant medį pirmąją viršūnę pasirinkti pati anksčiausią ir ilgiausią laiko intervalą.

2.3.3. „Kruskal" algoritmas

„Kruskal" algoritmas yra antras populiarusis minimalaus medžio paieškos algoritmas. Šio algoritmo veikimo principas kiek skiriasi nuo „Prim-Jarnik's" algoritmo. Šio algoritmo principas yra, kad jis konstruoja medį ne ieškodamas artimiausios viršūnės, bet kurdamas ir atnaujindamas medžius kuriem priklausytų pačios trumpiausios briaunos.

Algoritmui reikalingi kintamieji:

- V – Visų viršūnių aibė
- E – Visų briaunų aibė

while $\exists (u, v) \in E$
Pasirenkame lengviausią briauną $(u, v) \in E$
if ($u \notin$ jokiam medžiui ir $v \notin$ jokiam medžiui) kuriam naują medį ir priskiriam (u, v) tam medžiui
else
if ($u \in$ kokiam nors medžiui, o $v \notin$ jokiam medžiui) priskiriam (u, v) medžiui kuriam priklauso u
else
if ($u \notin$ jokiam medžiui, o $v \in$ kokiam nors medžiui) priskiriam (u, v) medžiui kuriam priklauso v
else
if ($u \in$ kokiam nors medžiui ir $v \in$ kokiam nors medžiui ir $u, v \notin$ tam pačiam medžiui) jungiam medžius kuriem priklauso u ir v
šalinam (u, v) iš E
return Medžių masyvą

Pav. 39 „Kruskal" algoritmo pseudo-kodas (Osipov V., Sanders P. ir Singler J., 2008)

Deja, pritaikius šį algoritmą laiko intervalams jis tampa labai neefektyvus. Visų pirma neefektyviu tampa viršūnių rūšiavimas pagal jų svorius, nes svoris tampa ne vieninteliu kriterijumi medžio auginimui. Dabar pasirinkus lengviausią briauną ir radus kokiems medžiams priklauso kiekviena viršūnė, reikia tikrinti ar lengviausios briaunos laiko intervalas nesikerta su kuriais nors laiko intervalais jau turimuose medžiuose. Tai reiškia, kad reikia apeiti visas

pasirinktų medžių briaunas ir tikrinti ar jos nesikerta. Jei briaunos nesikerta, tai medis konstruojamas įprastai.

Jei briaunos kertasi, įprastu atveju, reikėtų kurti naują medį, tačiau laiko intervalams reikia tikrinti, ar briauna gali būti prijungta prie kito medžio. Konstruojant paprasta jungųjį medį, visos briaunos yra prijungiamos prie vienintelių tinkamų medžių, o jei tinkamų medžių nėra, jie yra sukuriami. Laiko intervalams tinkamų medžių gali būti daug, nes jei medis tinkamas dėl jo jungumo su briauna, jis gali būti netinkamas dėl laiko intervalų kirtimosi. Tai reiškia, kad radus medį prie kurio galima būtų prijungti briauną, bet ją atmetus dėl intervalų kirtimosi, reikia tikrinti ar briauna nėra jungi su kitais medžiais. Radus kitus jungius medžius juos vėl reikia tikrinti ar jie tinkami briaunų jungumo atžvilgiu. Ir tik atlikus visus šiuos veiksmus ir neradus tinkamų medžių galima kurti naują medį.

Visas procesas gaunasi labai neefektyvus, nors jo rezultatas būtų sudarytas kelių darbuotojų laiko grafikas ir taip pat, atsakymas kiek mažiausiai reikia žmonių, kad atlikti visus darbus.

2.4. Galutinės sistemos modelis ir įgyvendinimas

Galutinė sistema kurią planavau įgyvendint ir kurią įgyvendinau nėra standartinių rezervavimo algoritmų rinkinys. Visi su rezervavimu susiję algoritmai buvo išvesti išanalizavus įvairias situacijas ir atlikus tyrimus.

2.4.1. Pradinis sistemos konfigūravimas

Kad supaprastint sistemos konfigūravimą, jis buvo įgyvendintas klausimų principu. Priklausomai nuo situacijos, konfigūravimas turi iki penkių etapų.

Pirmasis etapas susideda iš klausimų pateiktų keturioliktame paveikslėlyje. Patogumo dėlei klausimas apie papildomas rezervuojamas patalpas buvo nukeltas į klausimyno galą.

Klausimynas susideda iš penkių klausimų, bet atsakius teigiamai į tam tikrus klausimus šis etapas gali baigtis po dviejų. Po kiekvieno etapo vartotojui leidžiama apžvelgti jo pasirinkimus ir esant reikalui grįžti į to etapo pradžią.

Patvirtinus pirmojo etapo atsakymų korektiškumą duomenų bazėje yra sukuriama lentelė „configuration_string“. Joje saugomi atsakymai kuriuos pasirinko vartotojas.

Pav. 40 Klausimyno klausimai su galimais atsakymais (sudaryta autoriaus)

Nuo pirmojo klausimo atsakymo priklauso ar vartotojui reikės ką nors konfigūruoti antrame etape. Antrasis etapas skirtas konteinerio konfigūravimui, o esant tik vienam rezervuojamam objektui konteineriai nėra kuriami.

Visas etapas susideda iš penkių klausimų. Priklausomai nuo to kokia atsakymai buvo pasirinkti pirmame etape, vartotojui gali tekti atsakyti daugiausiai į keturis klausimus.

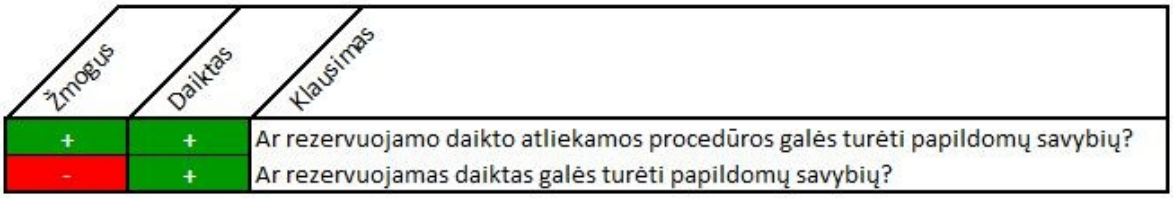
Antro konfigūravimo etapo klausimai:

1. Pagrindinio konteinerio pavadinimas (gali būti salės, pastato pavadinimas ir pan.)?
2. Darbo laikas (nuo - iki)?
3. Pietų laikas (nuo - iki)?
4. Turi rezervuojamų patalpų?
5. Ar yra rezervuojama patalpa?

Kokie klausimai bus pateikti priklauso nuo rezervuojamų objektų tipo.

Pav. 41 Antrojo konfigūravimo etapo klausimai (sudaryta autoriaus)

Trečiasis konfigūravimo etapas skirtas gauti informacijai apie papildomas daiktų ir procedūrų savybes. Priklausomai nuo rezervuojamo objekto tipo, konfigūruojant, gali tekti atsakyti į vieną arba du klausymus.



Pav. 42 Trečiojo konfigūravimo etapo klausimai (sudaryta autoriaus)

Paskutinysis etapas susideda iš vieno klausimo skirto išsiaiškinti ar bus naudojama automatinė tvarkaraščių sudarymo sistema.

Visas konfigūravimo procesas susideda iš keturių etapų. Kiekvieno etapo reikalavimus galima pamatyti paveikslėlyje (žr. 41 paveiksliuką).

Konfigūravimo etapas	Klausimynas	Konteinerio konfigūravimas	Procedūrų konfigūravimas	Tvarkaraščių sudarymas
Būtini pasirinkimai	Prieinama visiems	Daug rezervuojamų objektų	Rezervuojami objektai atlieka procedūras	Prieinama visiems

Pav. 43 Visas konfigūravimo procesas ir reikalavimai atskiriems etapams (sudaryta autoriaus)

Išvados ir pasiūlymai

1. WEB sistema – sistema leidžianti vartotojui lengvai sukurti rezervuojamus objektus, bei sudaryti jų rezervavimo tvarkaraščius. Tokią sistemą efektyviai įgyvendinti yra du būdai: WEB aplikacija ir WEB servisas. Dėl paprastumo, prieinamumo, įgyvendinimo ir talpinimo būdų populiarumo, bei optimalumo tokio tipo sistemai įgyvendinti labiausiai tinka WEB aplikacija.
2. Išanalizavus sistemos diegimo specifiką ir šio proceso reikalavimus buvo sudaryta ir detaliai aprašyta šio proceso schema (žr. 3 pav.). Schema apima visą sistemos diegimo kelią nuo reikiamų failų įkėlimo į serverį iki duomenų bazės kūrimo ir diegimo pabaigos. Analizuojant procesą taip pat buvo aprašyti reikalavimai duomenų bazių ir failų šalinimui proceso nutraukimo metu.
3. Atlikus tiek praktinę tiek teorinę rezervacijos reikalaujančių situacijų analizę galima teigti, kad bet kokiai situacijai sumodeliuoti užtenka trijų objektų tipų. Tai: rezervuojami objektai, rezervuojamų objektų konteineriai, rezervuojami rezervuojamų objektų konteineriai. Galutinės sistemos konfigūravimo metu rezervuojamų objektų konteineriai ir rezervuojami rezervuojamų objektų konteineriai, efektyvumo ir taupumo dėlei, yra saugomi vienoje duomenų bazės lentelėje. Kiekvienas konteineris, duomenų bazėje, turi kintamuosius identifikuojančius kokio lygio yra konteineris ir ar jis yra rezervuojamas.
4. Tolimesnė objektų ir situacijų analizė atskleidė, kad tam tikrais atvejais objektai ar procedūros gali tūrėti papildomų savybių. Atsižvelgiant į tai, sistemos modelis buvo papildytas galimybe konfigūruojant nustatyti ar objektai ir procedūros turės papildomų savybių. Buvo nustatytas būdas papildomų savybių saugojimui ir suprojektuota duomenų bazių lentelių struktūra.
5. Atlikus kelių praktinių situacijų analizę buvo sudarytas sąrašas būtinų funkcijų reikalingų rezervavimo procesui. Išskirti ypatumai specifiniai kiekvienai situacijai, bei ištirta kaip kiekvienas ypatumas galėtų būti pritaikytas galutinės, universalios sistemos rezervavimo procese.
6. Apibendrinus informaciją iš mokslinių šalinių buvo identifikuotos trijų duomenų bazių valdymo sistemų rūšių stipriosios savybės. Tos savybės buvo detaliai išnagrinėtos ir buvo išanalizuota galimybė pritaikyti kiekvienos rūšies duomenų bazių valdymo sistemas šiam projektui. Rezervavimo sistemos specifikacija nulėmė tai, kad buvo pasirinkta reliacinė duomenų bazių valdymo sistema.

7. Atlikus mokslinės literatūros analizę buvo nagrinėjama galimybė panaudoti įvairius „Greedy“ tipo algoritmus tvarkaraščių sudarymui. Buvo išnagrinėtas ir praktiškai pritaikytas „Bin-packing“ algoritmas. Taip pat buvo atlikti tyrimai su praktiniais duomenimis ieškant optimaliausiai laiką išnaudojančios algoritmo atmainos. Buvo nagrinėjama galimybė pritaikyti grafų teorijoje naudojamą medį ir minimalaus jungaus medžio paieškos algoritmą tvarkaraščių sudarymui. Buvo pateikti pasiūlymai kaip transformuoti duomenų bazės įrašus apie rezervavimą į grafą iš kurio sudaromas medis. Taip pat buvo pritaikytas „Prim-Jarnik‘s“ minimalaus jungaus medžio paieškos algoritmas. Buvo pasiūlytos modifikacijos „Kruskal“ minimalaus jungaus miško paieškos algoritmui tam, kad jis būtų pritaikytas tvarkaraščių sudarymui.
8. Atliekant darbą buvo įgyvendinta pora svarbiausių sistemos modulių. Sistemos konfigūravimas vyksta pagal truputi modifikuotą sistemos konfigūravimo schemą (žr. 15 pav.). Administravimas ir rezervavimas vyksta grožio salono situacijos principu.

Pasiūlymai tolimesniam darbui:

1. Toliau tęsti darbą pilnu ir detaliu aprašytos sistemos įgyvendinimu.
2. Toliau tęsti minimalaus jungaus miško paieškos algoritmo - „Kruskal“, pritaikymo tvarkaraščių sudarymui, galimybių. Išnagrinėti galimybę panaudoti deterministinius lokalaus optimumo paieškos metodus bandant rasti globalųjį optimalų sprendimą.
3. Sistemą papildyti galimybe registruoti kelias skirtingas vartotojų grupes tam, kad būtų galima plačiau išnaudoti tvarkaraščių sudarymą.
4. Galutinę sistemą papildyti funkcionalumais susijusiais su grafinės vartotojo sąsajos manipuliavimu.

Literatūros sąrašas

[Bar05] R.Baronas - „Duomenų bazių valdymo sistemos“ 2005 m.

[Bar10] P. N. Barus – „Data Structures - Tree“ 2010 m.

[BK00] B. Burgi, A. Kulikauskas „Kompiuterija“ 2000 m.

[BW05] J. O. Berkey, P. Y. Wang „Two-Dimensional Finit Bin-Packing Algorithms“ 2005m.

[Cha83] B. Chazelle - The Bottom-Left Bin-Packing Heuristic 1983 m.

[Cod11] CodeIgniter User Guide, 2011 m. URL: http://codeigniter.com/user_guide/

[CT76] D. Cherition and R. E. Tarjan - Finding minimum spanning trees 1976 m.

[Die10] R. Diestel – „Graph Theory“ 2010 m.

[DP06] M.Davis, J. Phillips „O‘Reilly – Learning PHP and MySQL“ 2006 m.

[Had11] B. Hadorn – „Cell Computing Model or Zeus-Framework“ 2011 m.

[Leg02] W. Legierski – „Constraint-based reasoning for timetabling“ 2002 m.

[LM10] A. Lith, J. Mattsson – „Investigating storage solutions for large data“ 2010 m.

[LZ09] X. Li, H. Zhao - „Greedy Algorithm Solution of Flexible Scheduling Problem“ 2009 m.

[Mat07] D. Matuszek – „Programming Languages & Techniques - Greedy Algorithms“ 2007 m.

[OSS08] V. Osipov, P. Sanders and J. Singler – „The Filter-Kruskal Minimum Spanning Tree Algorithm“ 2008 m.

[Pri57] R. C. Prim – „Shortest connection networks and some generalizations“ 1957 m.

[Pon07] Dr. D. Ponomarev – “Design and Analysis of Algorithms - Introduction to Greedy Algorithms“ 2007 m.

[PMA11] “phpMyAdmin project”, 2011 URL: http://www.phpmyadmin.net/home_page/docs.php

[Sur08] D. Sureau „History of Computer Languages and Their Evolution“ 2008m.

[Zha08] X. L. H. Zhao – Greedy Algorithm Solution of Flexible Scheduling Problem 2008m.