

KAUNO TECHNOLOGIJOS UNIVERSITETAS

KOMPIUTERIŲ KATEDRA

**PROGRAMŲ SISTEMŲ INŽINERIJA** (**P175B015**)

Tvarkaraščių išdėstymo ir lengvo redagavimo sistema – „TimeTable“

Vadovas:

Kęstutis Valinčius

Vertino:

Virginija Limanauskienė

Atliko:

Lukas Adeikis IFF – 5/1 gr.

Valdemaras Ambraziūnas IFF – 5/1 gr.

Paulius Venclovas IFF – 5/1 gr.

Mažvydas Vaicekauskas IFF – 5/1 gr.

KAUNAS 2017

Turinys

[1. Įvadas 3](#_Toc483258945)

[1.1. Idėja, jos raida 3](#_Toc483258946)

[1.2. Trumpas realizuotos sistemos aprašas 4](#_Toc483258947)

[2. Techninis aprašymas, reikalavimai 5](#_Toc483258948)

[2.1. Funkciniai reikalavimai 5](#_Toc483258949)

[2.2. Nefunkciniai reikalavimai 5](#_Toc483258950)

[2.3. Išsamus techninis aprašas 6](#_Toc483258951)

[3. Sistemos diagramos 7](#_Toc483258952)

[3.1. Išdėstymo diagrama 7](#_Toc483258953)

[3.2. Duomenų bazės diagrama 8](#_Toc483258954)

[3.3. Būsenų diagrama 9](#_Toc483258955)

[3.4. Panaudos atvejų diagrama 10](#_Toc483258956)

[3.5. Klasių diagrama 11](#_Toc483258957)

[4. Grafinė vartotojo sąsaja 13](#_Toc483258958)

[4.1. Sistemos GUI nuodugnus paaiškinimas 13](#_Toc483258959)

[5. Algoritmai 17](#_Toc483258960)

[5.1. Paskaitų laiko paskirstymo algoritmas 17](#_Toc483258961)

[5.2. Paskaitos laiko keitimo algoritmas 19](#_Toc483258962)

[5.3. Dviejų paskaitų keitimas vietomis 22](#_Toc483258963)

[5.4. Algoritmas, kuris suranda galimus keitimus 27](#_Toc483258964)

[5.5. Undo algoritmas 30](#_Toc483258965)

[6. Testai 34](#_Toc483258966)

[7. Šaltiniai 36](#_Toc483258967)

[8. Išvados 36](#_Toc483258968)

# Įvadas

## Idėja, jos raida

Pradinė projekto idėja kilo samprotaujant, diskutuojant apie problemas, su kuriomis susiduria studentai. Aptariant problemas pastebėjome, jog buvo kelios problemos, kurios, mūsų nuomone, studentams sukėlė daugiau vargo, nepasitenkinimo, nei kitos aptartosios. Viena iš šių problemų buvo nepatogus paskaitų tvarkaraštis. Visi buvome tai patyrę, kai po paskaitos, sekančios tenka laukti kelias valandas, dėl tvarkaraštyje esančių langų. Pagalvojome, kad kompiuteris turėtų gebėti sudėlioti tvarkaraščius taip, kad visi būtų kiek įmanoma daugiau patenkinti, todėl nusprendėme pabandyti išspręsti tvarkaraščio generavimo uždavinį.

Iš pirmo žvilgsnio mums pasirodė, kad mūsų idėjos realizacija nebus velniškai sunki, tačiau kuo daugiau gilinomės į savo užduoties detales, tuo daugiau supratome, kokia ji sudėtinga. Pradėjome nuo dabartinio tvarkaraščių sudarymo būdo analizės. Kad sužinotumėme, kokiu būdu sudaromi tvarkaraščiai pakalbėjome su už tai atsakingu žmogumi, prodekanu. Vos pasakėme jam savo idėją, gavome labai aiškų atsakymą: „mission impossible“. Mums paaiškino, jog tvarkaraščių automatiniam sudarymui reikia numatyti labai daug parametrų, todėl bendro sprendimo nėra, o sudaryti specializuotą sprendimą užtrunka labai daug laiko ir resursų. Prodekanas pasakė, kad universitetas specializuotos programinės įrangos šiam reikalui neturi, o tvarkaraščiai pildomi rankiniu būdu. Šio pokalbio neužteko, kad visiškai atkalbėti mus nuo savo idėjos. Nusprendėme dar panagrinėti jau realizuotus algoritmus ir pažiūrėti ar yra prieinamas būdas juos pritaikyti savo idėjai. Po perskaitytų straipsnių analizės, pastebėjome tendenciją – tokio tipo užduotims naudojami genetinio pobūdžio algoritmai. Tokio algoritmo realizacija mūsų komandai buvo visiškai nepažįstama, tad nusprendėme atsisakyti šios idėjos, nes jos realizacija būtų pernelyg sudėtinga ir užimtų per daug laiko.

Savo užduotį keitėme atsižvelgdami į prodekano pastabą, kad reikia sistemos, kurioje būtų lengva manipuliuoti tvarkaraščiu. Nutarėme tvarkaraščio generavimo dalį perduoti jau užbaigtai atviro kodo programai, o patys pasirūpinti, jog gautą tvarkaraštį bus nesunku redaguoti. Tvarkaraščių generavimui pasirinkome programą „unitime“, bet pradėję jos integraciją į savo projektą supratome, kad tai pernelyg didelė programa, kad galėtume ją perprasti, suderinti su savo projektu, per turimą laiko tarpą. Šią programą sudarė šimtai failų, o jos duomenų bazę daugiau kaip 50 lentelių. Dėl šių priežasčių nusprendėme galutinai pakeisti savo tikslą į: web programos realizavimą, kuri leistu administratoriaus teises turintiems vartotojams sukurti paskaitas, jas automatiškai išdėlioti tvarkaraštyje ir stebėti kaip buvo išdėliotos paskaitos, esant reikalui, šį sudarytą tvarkaraštį nesunkiai redaguoti.

## Trumpas realizuotos sistemos aprašas

Darbas su sistema prasideda prisijungus prie tinklapio. Vartotojui patekus į tinklapį reikia prisijungti/prisiregistruoti, taip atlikus yra 2 pagrindiniai programos naudojimo keliai.

Kelias A: prisijungęs asmuo yra studentas arba dėstytojas:

* Paprastai tokio tipo vartotojams suteikiamos tvarkaraščių peržiūrėjimo teisės, tad jų galimybės ir yra ribojamos šių teisių.
* Žinoma įmanomas ir mažiau tikėtinas atvejis, kai administratorius suteikia dalį savo privilegijų ir vartotojams.

Kelias B: prisijungia vienas administratorių, kuris gali atlikti šiuos veiksmus:

* Iš duomenų bazėje esančių auditorijų, grupių, dėstytojų ir modulių sudaryti paskaitas.
* Pildyti duomenų bazę naujomis auditorijomis ar kitomis paskaitų sudedamosiomis.
* Naikinti nereikalingą informaciją iš duomenų bazės
* Generuoti tvarkaraščius
* Redaguoti tvarkaraščius
* Valdyti vartotojų privilegijas

# Techninis aprašymas, reikalavimai

## Funkciniai reikalavimai

Programa savo veikimu turi patenkinti šį funkcinių reikalavimų sąrašą:

* Realizuotos vartotojų paskyros, leidžiančios kiekvienam vartotojui stebėti duomenis, kurie yra jam aktualūs.
* Suprojektuota paskyrų hierarchija taip, kad būtų lanksti (galimybė keisti privilegijas), saugi(neprivilegijuoti asmenys negali keisti informacijos, kurios keitimas apsunkintų programos naudojimą kitiems vartotojams).
* Paslauga prieinama vartotojams bet kada ir bet kur, be jokių specialių programinės įrangos paketų.
* Paprasta ir intuityviai suprantama grafinė sąsaja, kurioje vartotojai nesunkiai gali peržiūrėti savo tvarkaraščius ar atlikti kitus pageidaujamus veiksmus.
* Realizuota sąsaja su duomenų baze, kurios dėka privilegijuoti asmenys gali redaguoti duomenų bazės turinį, neturėdami jokios patirties duomenų bazių valdyme.
* Duomenų išdėstymas toks, kad administratorius gali vienu metu stebėti didelį duomenų kiekį, be problemų suprasti jų pasiskirstymą.
* Realizuotos keli duomenų peržiūrėjimo variantai(peržiūros/peržiūros ir redagavimo).

## Nefunkciniai reikalavimai

Programa savo veikimo metodika turi patenkinti šį nefunkcinių reikalavimų sąrašą:

* Duomenys saugomi duomenų bazėje.
* Duomenų mainai tarp front-end ir back-end atliekami vadovaujantis JSON duomenų perdavimo protokolu.
* Programos realizavimui naudotos technologijos:
  + Microsoft SQL management studio,
  + ASP.NET MVC,
  + jQuery,
  + HTML5, CSS3.
* Darbo metu kodas talpinamas „BitBucket“ repositorijoje.

## Išsamus techninis aprašas

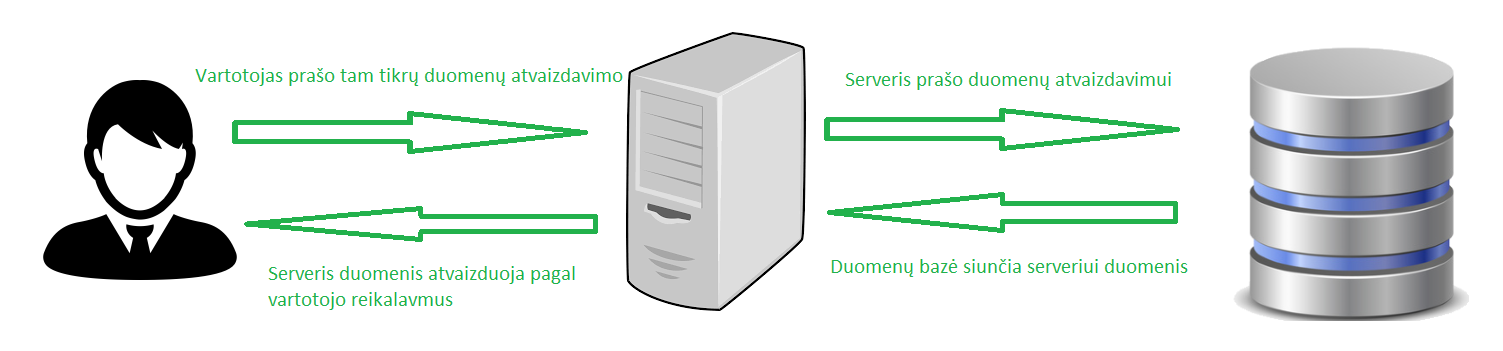
Projektas yra sudarytas iš dviejų dalių: klientinės ir serverinės. Klientinėje dalyje yra vaizduojami pagrindinis tvarkaraštis, įvykių konsolė ir duomenų bazių lentelių valdymo puslapiai , kuriuose galima pridėti, pašalinti, redaguoti, peržiūrėti laukus. Duomenų bazės lentelių valdymo puslapiuose duomenys yra persiunčiami HTTP POST protokolu URL šifruote. Įvykių žurnalas yra nuolatos atnaujinamas, kas 5 sekundes siunčiant AJAX užklausas serveriui. Užklausos rezultatas yra sudarytas iš žinučių masyvo, kur kiekviena žinutė yra šitokio formato: žinutės numeris, pakeistų paskaitų id, veiksmo aprašymo ir vėliavėlės, kur nurodo ar galima atšaukti pakeitimus. Pagrindinį tvarkaraštį sudaro redagavimo mygtukas ir esamos tvarkaraščio būsenos atvaizdavimas. Paspaudus mygtuką galima žymėti paskaitas, kurios bus galbūt apkeistos, arba pažymėti paskaitą ir laiką į kurį norima perkelti. Pažymėjus paskaitas ar laikus į serverį siunčiama AJAX užklausa, kad būtų patikrinta, ar galima apkeisti ir sėkmės atveju apkeistų paskaitas vietomis. Taip pat virš tvarkaraščio yra mygtukai, kurie šiuos veiksmus leidžia atlikti nežymint paskaitų tvarkaraštyje.

Serverinėje dalyje yra sukurta realizacinė MySQL duomenų bazė, kurios lentelių parametrus galima pridėti arba šalinti tiesiogiai koduojant juos į Models aplanke esančias klases. Čia sukurtas Base kontroleris, kurį paveldi visi likę kontroleriai, kuris užtikrina serverio bendravimą su duomenų baze.

Visos užklausos, kurios siunčiamos iš klientinės dalies, patenka į klasę LecturesController, kuri apdoroja duomenis ir sukuria Log klasės kintamąjį, kuris išsaugo paskaitų būseną prieš pakeitimą (jei vartotojas ateityje norės atšaukti keitimą). Pagal gautus duomenis yra kreipiamasi į algoritmus skirtus teisingai suvaldyti duomenų bazės duomenis. Šie algoritmai yra klasėje SwapService. Vykstant bet kuriam iš keitimo algoritmų yra pildomas Log objekto pranešimas, kad vartotojas būtų informuotas apie keitimo eigą, jo rezultatus. Pasibaigus keitimui, nepriklausomai sėkmingai ar nesėkmingai, informacija apie 10 paskutinių log‘ų yra išsiunčiami JSON formatu į konsolę, kurią mato vartotojas. JSON formatu yra siunčiamas ir sąrašas galimų keitimų, kai keitimas vartotojo dalyje buvo atliktas naudojant mygtukus, o ne spaudant tvarkaraštį. Be to, LecturesController bendrauja ir su kitu servisu (GenerateService), kuris atlieka duomenų bazės įrašų validacija, bei laikų priskyrimą paskaitoms, kurios dar yra be laiko. Taip pat, gavus užklausą čia atliekamas ir atšaukimo (undo) veiksmas. Jis nėra toks paprastas, kaip gali pasirodyti, nes reikia įsitikinti, kad jis galimas dėl atliktų keitimų po to keitimo, kurį norime atšaukti. Dėl šios priežasties, vėl yra sekamas visas algoritmas ir jo rezultatai fiksuojami, kad vartotojas galėtų matyti, kodėl jis negali atlikti atšaukiamojo veiksmo.

# Sistemos diagramos

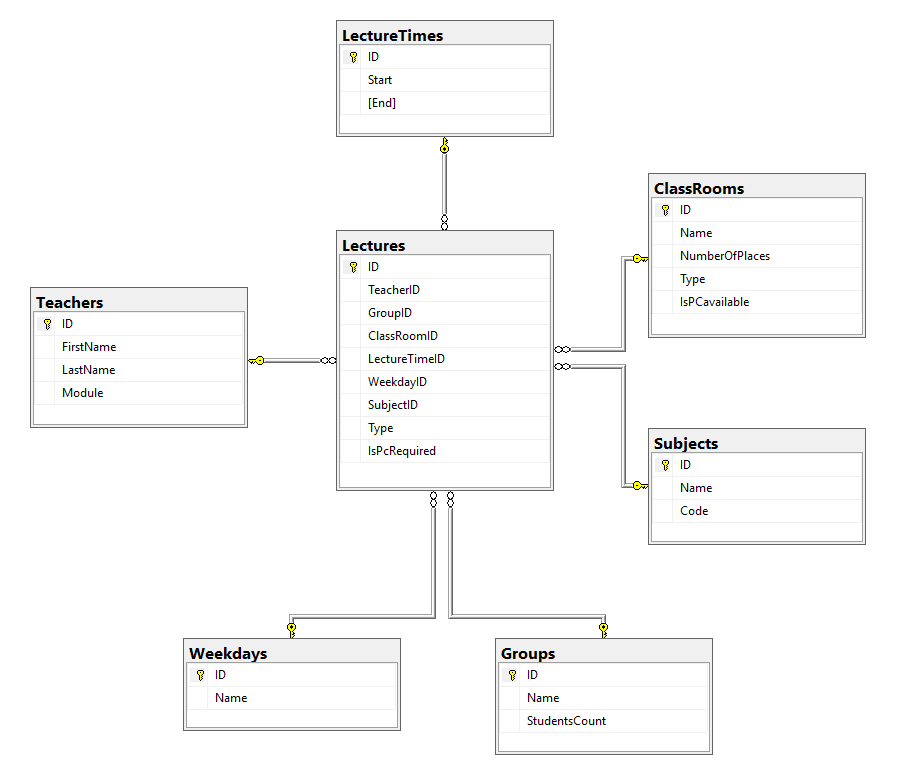
## Išdėstymo diagrama

Supaprastintą sistemos veikimą apibūdina ši diagrama:

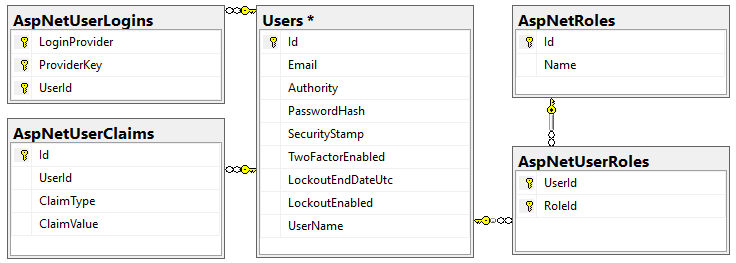
Sakykime, kad vartotojas nori pažiūrėti savo tvarkaraštį, jis spaudžia peržiūros mygtuką (prašo tam tikrų duomenų atvaizdavimo), serveris pastebėjęs šį įvykį kreipiasi į duomenų bazę (prašo duomenų atvaizdavimui). Iš duomenų bazės pagal kriterijus atrenkami duomenys ir grąžinami serveriui (duomenų bazė siunčia serveriui duomenis). Serveris gautus duomenis apdoroja, sudeda į vaizdingą šabloną ir pateikia vartotojui (atvaizduoja pagal vartotojo reikalavimus).

## Duomenų bazės diagrama

Šioje diagramoje matome kaip, duomenų bazėje, yra išsidėsčiusios lentelės. Žiūrėdami į išsidėstymą matome, koks yra projekto duomenų saugojimo modelis.

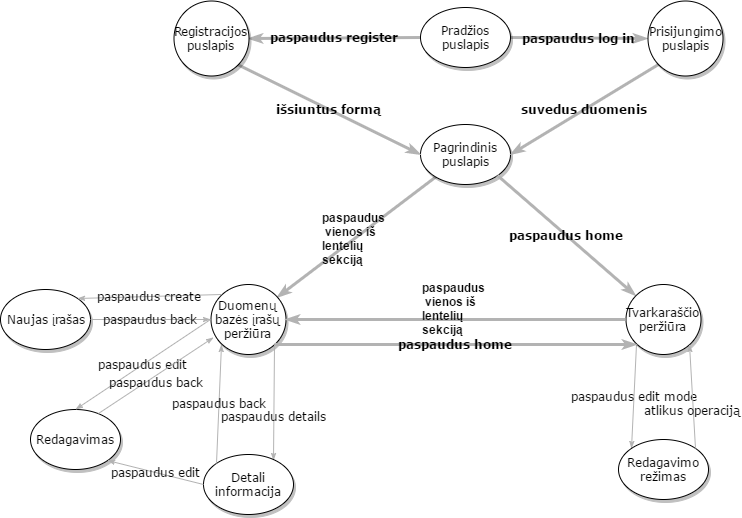


Šioje diagramoje matome duomenų bazės dalį, kurioje saugomi vartotojų prisijungimo duomenys:



## Būsenų diagrama

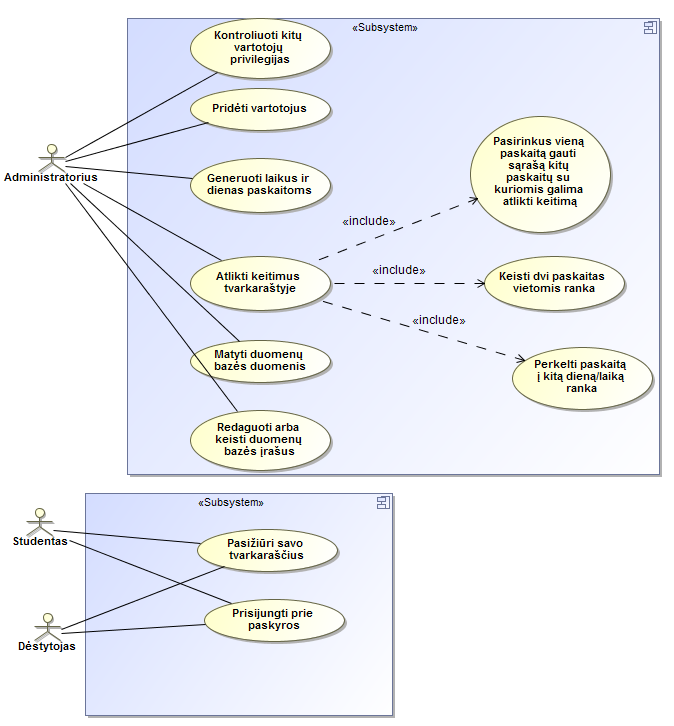
Ši diagrama parodo, į kokias būsenas gali patekti web aplikacija, valdoma kliento:



Pradedant naudotis programa vartotojas yra vartotojas yra nukreipiamas į pradžios puslapį, kad galėtų pradėti darbą su sistema, vartotojas privalo prisijungti prie serverio. Tai gali atlikti arba per prisijungimo puslapį arba susikurdamas naują paskyrą, registracijos puslapyje. Prisijungęs vartotojas patenka į pagrindinį puslapį, kuriame gali rinktis ką daryti toliau. Autorizuoti vartotojai gali peržiūrėti duomenų bazės turinį, jį redaguoti. Tai atlieka spausdami ant dominančios lentelės pavadinimo meniu navigacinėje juostoje. Taip pat yra galimybė peržiūrėti tvarkaraščius, tai galime atlikti spausdami home mygtuką, navigacinėje meniu juostoje. Peržiūrėdami tvarkaraščius privilegijuoti vartotojai gali tvarkaraštį ir redaguoti. Kad patekti į redagavimo būseną reikia spausti mygtuką: edit mode. Atlikus redagavimo veiksmą arba paspaudus edit mode mygtuką bus grįžtama į tvarkaraščio peržiūros stadiją.

## Panaudos atvejų diagrama

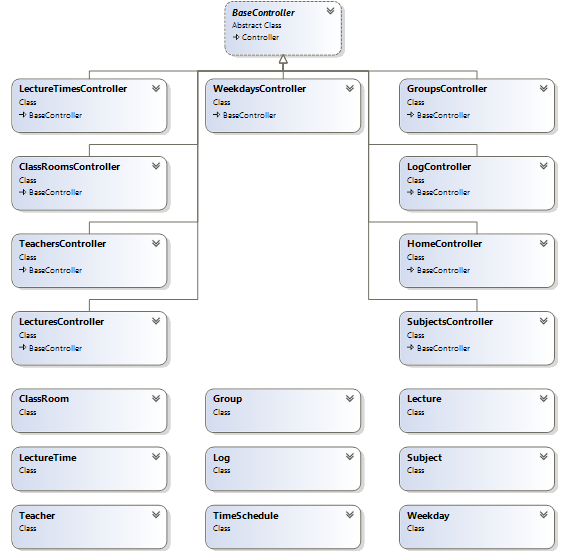
Sistemos panaudos atvejų diagrama:

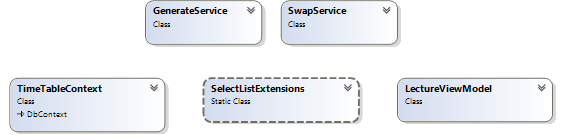


Diagramoje gerai matosi vartotojų tipai ir jiems suteikiamos privilegijos. Visi vartotojų tipai gali prisijungti prie savo paskyros. Dėstytojai ir studentai gali stebėti savo tvarkaraščius. Daugiausia privilegijų turi administratorius: jis gali sąveikauti su duomenų bazės informacija, vartotojais ir tvarkaraščiais.

## Klasių diagrama

Šioje diagramoje matome visas klases, jų atliekamas funkcijas bei ryšius su kitomis klasėmis:





Klasių diagramos aprašymas:

**BaseController**: kontroleris, kuris turi priėjimą prie duomenų bazės, ir šį priėjimą suteikia visiems kitiems kontroleriams.

**HomeController**: kontroleris, kuris sudeda duomenų bazės įrašus į sąrašus, kurie vėliau yra atvaizduojami.

**LecturesController**: kontroleris, kuriame aprašyti mygtuko paspaudimo veiksmai, susiję su tvarkaraščio paskaitų keitimu, tvarkaraščio validacija ar paskaitų laikų sudėliojimu. Jis priima duomenis, kuriuos vartotojas nurodė grafinėje sąsajoje ir nusiunčia juos į servisus, kur atliekami veiksmai.

**Visi likę Controller** (+ jau minėtas LecturesController): kontroleriai yra atsakingi už savo pavadinimo duomenų bazės lentelės funkcionalumą. Tai yra, visuose juose aprašyti atnaujinimo (edit), šalinimo (delete), įterpimo (insert), ar informacijos peržiūrėjimo (details) veiksmai.

**ClassRoom**: klasės/auditorijos objektų klasė, kurioje aprašyti su klase susiję kintamieji/parametrai.

**Klasės Group, Lecture, Subject, LectureTime, Weekday, Teacher** savo prasme yra identiškos klasej ClassRoom, tik aprašo sau būdingus parametrus.

**Log**: log objektų klasė, kurioje aprašyti kintamieji reikalingi sekti atliktus keitimus, kad vėliau vartotojas galėtų juos atšaukti.

**GenerateService**: servisas, kuriame yra aprašyta validacijos ir paskaitų laikų sudėliojimo logika.

**SwapService**: servisas, kuriame aprašyta logika susijusi, su paskaitų laikų keitimais: dviejų paskaitų tarpusavyje, vienos paskaitos laiko keitimo, pasirinkimo iš sąrašo, bei keitimo atšaukimo (undo).

**TimeTableContext**: apibrėžia paskaitos dalyvių reikšmes, nes duombazėje saugomi id iš kitų lentelių.

**LectureViewModel**: modelis, kuris apibrėžia paskaitos dalyvių pasirinkimą, kuriant naują paskaitos įrašą duomenų bazėje.

**SelectListExtension**: klasė, kuri į sąrašo pradžią prideda numatytą elementą.

# Grafinė vartotojo sąsaja

## Sistemos GUI nuodugnus paaiškinimas

Visos sistemos grafinę sąsają galime suskaldyti į tokius modulius:

|  |
| --- |
| Prisijungimo |
| C:\Users\Lukas\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ss (2017-05-22 at 06.18.46).png |
| Šiame modulyje galime prisijungti prie savo paskyros, įvesdami nurodytą informaciją. Taip pat jei neturite paskyros galite spausti nuorodą ir patekti į registracijos puslapį. |

|  |
| --- |
| Registracijos |
| C:\Users\Lukas\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ss (2017-05-22 at 06.18.29).png |
| Šiame puslapyje galime prisiregistruoti prie sistemos, įvesdami nurodytą informaciją. |

|  |
| --- |
| Sąrašų peržiūros |
| C:\Users\Lukas\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ss (2017-05-22 at 03.13.58).png |
| Šiame modulyje galime peržiūrėti visus tam tikros duomenų bazės įrašus (yra kiekvienai lentelei individuali versija). Taip pat galima į lenteles pridėti naujų įrašų, redaguoti esančius, sužinoti daugiau informacijos apie kažkurį iš įrašų arba naikinti pageidaujamus. |

|  |
| --- |
| Naujo įrašo |
| C:\Users\Lukas\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ss (2017-05-22 at 03.14.12).png |
| Naujo įrašo langas, kaip ir tikimasi, suteikia galimybę pridėti naujų įrašų į duomenų bazę (realizuota su visomis lentelėmis). Redagavimo lango išvaizda, veikimo principas labai panašūs, tad jo nenagrinėsime kaip atskiro modulio. |

|  |
| --- |
| Detalios informacijos |
| C:\Users\Lukas\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ss (2017-05-22 at 03.51.45).png |
| Šis modulis leidžia pažiūrėti visą informaciją apie norimą duomenų bazės įrašą. Jei įrašas yra grupė, dėstytojas ar auditorija, taip pat matome jų užimtumo tvarkaraštį. |

|  |
| --- |
| Paskaitų sąrašo peržiūros |
| C:\Users\Lukas\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ss (2017-05-22 at 03.16.10).png |
| Šiame modulyje matome visų paskaitų sąrašą, turime galimybę kurti naujas paskaitas, redaguoti esančias, norimas pašalinti. Žalias mygtukas atsakingas už tai, kad jį paspaudus, paskaitos, kurios neturi nurodyto laiko, būtų sudėliojamos į laisvas vietas tvarkaraštyje. |

|  |
| --- |
| Paskaitų kūrimo |
| C:\Users\Lukas\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ss (2017-05-22 at 03.16.19).png |
| Šis langas yra atsakingas už naujos paskaitos sukūrimą. |

|  |
| --- |
| Bendro tvarkaraščio peržiūrėjimo |
| C:\Users\Lukas\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ss (2017-05-22 at 03.18.01).png |
| Šiame lange galime stebėti visų paskaitų informaciją tvarkaraščio formoje ir vieno mygtuko paspaudimu pereiti į bendro tvarkaraščio redagavimo modulį. |

|  |
| --- |
| Bendro tvarkaraščio redagavimo |
| C:\Users\Lukas\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ss (2017-05-22 at 03.18.14).png |
| Šiame modulyje galime nesunkiai redaguoti paskatų išsidėstymą. Tereikia tik paspausti ant norimos paskaitos, kad ją pažymėti ir spausti ant norimos vietos, kad perkelti ten. Jei norimas perkėlimas nepažeis tvarkaraščio taisyklių jis bus įvykdomas. Atliktų operacijų baigtis visada bus spausdinama konsolėje. |

|  |
| --- |
| Konsolė |
| C:\Users\Lukas\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ss (2017-05-22 at 04.23.53).png |
| Visų atliktų operacijų baigties pranešimai randami šiame modulyje. Šio modulio paskirtis nėra vien tik spausdinti pranešimus, taip pat galima, paspaudus mygtuką „undo“, anuliuotu kažkurį iš prieš tai įvykdytų veiksmų. |

# Algoritmai

## Paskaitų laiko paskirstymo algoritmas

Algoritmas, kaip ir visi kiti mūsų algoritmai, skirtas užtikrinti duomenų bazės duomenų korektiškumą. Čia yra sukurtoms paskaitos (vartotojas jas kuria kliento dalyje), priskiriami laikai, taip kad nesikirstų nei vieno dėstytojo, grupės ar auditorijos tvarkaraščiai, t.y. vienu metu vienas iš paskaitos dalyvių turi dalyvauti tik vienoj paskaitoje. Algoritmas yra gana “bukas”, nes tvarkaraščio generavimas buvo pirminė mūsų idėja, kuri reikalavo sunkių genetinių algoritmų, ir paskaitų laikus dėlioja pagal suspaudimo principą (kuo arčiau pirmadienio 8:30), tačiau šis algoritmas yra pakankamas užtikrinti patikimą duomenų bazės įrašų kontrolę.

|  |
| --- |
| Paskaitų laiko paskirstymo metodas: public void Generate() |
| /// <summary>  /// Paskaitu laiko priskirimo metodas  /// </summary>  public void Generate()  {  //Visi galimi laikai  List<LectureTime> lecturesTimes = db.LectureTimes.ToList();  //Visos galimso dienos  List<Weekday> weekdays = db.Weekdays.ToList();  //Visos klasės su tvarkaraščiais  List<ClassRoom> classrooms = db.ClassRooms.ToList();  List<TimeSchedule> classSchedule = new List<TimeSchedule>();  foreach (ClassRoom item in classrooms)  {  TimeSchedule schedule = new TimeSchedule(item.ID,  lecturesTimes.Count);  classSchedule.Add(schedule);  }  //Visi dėstytojai su tvarkaraščiais  List<Teacher> teachers = db.Teachers.ToList();  List<TimeSchedule> teachersSchedule =  new List<TimeSchedule>();  foreach (Teacher item in teachers)  {  TimeSchedule schedule = new TimeSchedule(item.ID,  lecturesTimes.Count);  teachersSchedule.Add(schedule);  }  //Visos grupės su tvarkaraščiais  List<Group> groups = db.Groups.ToList();  List<TimeSchedule> groupsSchedule = new List<TimeSchedule>();  foreach (Group item in groups)  {  TimeSchedule schedule = new TimeSchedule(item.ID,  lecturesTimes.Count);  groupsSchedule.Add(schedule);  }  foreach (Lecture lecture in db.Lectures)  {  //Išrenkam reikiamus dalykus  Teacher currTeacher = lecture.Teacher;  Group currGroup = lecture.Group;  ClassRoom currClass = lecture.ClassRoom;  TimeSchedule currTeacherSched = teachersSchedule.  Where(x => x.ID == currTeacher.ID).  Select(x => x).FirstOrDefault();  TimeSchedule currGroupChed = groupsSchedule.  Where(x => x.ID == currGroup.ID).  Select(x => x).FirstOrDefault();  TimeSchedule currClassSched = classSchedule.  Where(x => x.ID == currClass.ID).  Select(x => x).FirstOrDefault();  //Jei nėra laiko, ieškom laisvo laiko  if (lecture.LectureTimeID == null &&  lecture.WeekdayID == null)  {  foreach (Weekday day in weekdays)  {  foreach (LectureTime time in lecturesTimes)  {  //Patikrinam ar šiuo laiku dėstytojas,  grupė ir auditorija yra laisva  if (currTeacherSched.isFree(day.ID, time.ID)  && currGroupChed.isFree(day.ID, time.ID)  && currClassSched.isFree(day.ID, time.ID))  {  lecture.LectureTimeID = time.ID;  lecture.WeekdayID = day.ID;  currTeacherSched.setTrue(day.ID,  time.ID);  currGroupChed.setTrue(day.ID, time.ID);  currClassSched.setTrue(day.ID, time.ID);  break;  }  }  //Patirkinam ar radom laiką paskaitai  if (lecture.LectureTimeID != null &&  lecture.WeekdayID != null)  {  break;  }  }  }  //Jei laikas priskirtas pasižymime į tvarkaraščius  else  {  currTeacherSched.setTrue(lecture.WeekdayID ?? 0,  lecture.LectureTimeID ?? 0);  currGroupChed.setTrue(lecture.WeekdayID ?? 0,  lecture.LectureTimeID ?? 0);  currClassSched.setTrue(lecture.WeekdayID ?? 0,  lecture.LectureTimeID ?? 0);  }  }  db.SaveChanges();  } |

Taigi, norint gauti norimą rezultatą, tenka atsižvelgti į trijų pagrindinių aktorių tvarkaraščius: dėstytojų, grupių ir auditorijų. Algoritmas suranda visiems paskaitoms dalyviams tinkamą laiką ir išsaugo rezultatus duomenų bazėje.

## Paskaitos laiko keitimo algoritmas

Algoritmas, skirtas pakeisti vienos paskaitos laikui taip, kad būtų atsižvelgta į visus jos dalyvių tvarkaraščius ir duomenys išliktų tvarkingi, t.y. niekam nesikirstų laikai tvarkaraštyje. Jei priskirta auditorija yra užimta yra ieškoma kitos auditorijos, kuri atitinka visus buvusios parametrus. Tie patys veiksmai yra atliekami ir su dėstytoju. Algoritmas išsaugo vykdymo rezultatų eigą, kurios vėliau gali matyti vartotojas.

|  |
| --- |
| Paskaitos laiko keitimo algoritmas: public void ChangeLectureTime(int lectureId, int newLectureTimeId, int newLectureDayId, ref string msg) |
| /// <summary>  /// Metodas, kuris pakeicia paskaitos laika, jei imanoma  /// </summary>  /// <param name="lectureId">ID paskaitos, kurios laika keisim  </param>  /// <param name="newLectureTimeId">Naujo laiko id</param>  /// <param name="newLectureDayId">Naujos dienos id</param>  /// <param name="msg">Pranesimas i konsole</param>  public void ChangeLectureTime(int lectureId,  int newLectureTimeId, int newLectureDayId, ref string msg)  {  Lecture first = db.Lectures.Find(lectureId);  if (first.WeekdayID != null && first.LectureTimeID != null)  {  //Paskaitos dalyvių tvarkaraščiai  TimeSchedule teacher = Schedule(first.TeacherID,  "TeacherID");  TimeSchedule classRoom = Schedule(first.ClassRoomID,  "ClassRoomID");  TimeSchedule group = Schedule(first.GroupID, "GroupID");  //Patikrinam ar laisvi norimu laiku  bool teacherIsFree = teacher.isFree(newLectureDayId,  newLectureTimeId);  bool classRoomrIsFree = classRoom.isFree(newLectureDayId,  newLectureTimeId);  bool groupIsFree = group.isFree(newLectureDayId,  newLectureTimeId);  //Jei visi laisvi, tai apkeičiam  if (teacherIsFree && classRoomrIsFree && groupIsFree)  {  first.WeekdayID = newLectureDayId;  first.LectureTimeID = newLectureTimeId;  msg = "Change was successful.";  }  else  {  if (groupIsFree)  {  Lecture second = new Lecture();  second.LectureTimeID = newLectureTimeId;  second.WeekdayID = newLectureDayId;  bool isPossibleToChangeFirstTeacher =  isPossibleToChangeTeacher(first, second);  bool isPossibleToChangeFirstClassRoom =  isPossibleToChangeClassRoom(first, second);  Teacher firstTeacher = null;  ClassRoom firstClassRoom = null;  var rand = new Random();  if (isPossibleToChangeFirstTeacher == false)  {  Teacher[] teachers = db.Teachers.  Where(x => x.Module == first.Teacher.Module  && x.ID != first.TeacherID).  Select(x => x).ToArray();  if (teachers.Length > 0)  {  Teacher[] valid =  new Teacher[teachers.Length];  int count = 0;  foreach (Teacher teacherr in teachers)  {  TimeSchedule teacherSchedule =  Schedule(teacherr.ID, "TeacherID");  If  (teacherSchedule.isFree(second.WeekdayID  ?? 0, second.LectureTimeID ?? 0))  {  valid[count] = teacherr;  count++;  }  }  if( count > 0)  {  firstTeacher =  valid.ElementAt(rand.Next(0, count));  msg = "Lecture teacher was changed  to " + firstTeacher.LastName  + " " + firstTeacher.FirstName  + ". ";  }  }  }  else  firstTeacher = first.Teacher;  if (isPossibleToChangeFirstClassRoom == false)  {  ClassRoom[] classRooms;  if (first.IsPcRequired == true)  {  classRooms = db.ClassRooms.  Where(x => x.Type == first.ClassRoom.Type  && x.ID != first.ClassRoomID && x.Type ==  first.Type && x.NumberOfPlaces >=  first.Group.StudentsCount &&  x.IsPCavailable == true).  Select(x => x).ToArray();  }  else  classRooms = db.ClassRooms.  Where(x => x.Type == first.ClassRoom.Type  && x.ID != first.ClassRoomID && x.Type ==  first.Type && x.NumberOfPlaces >=  first.Group.StudentsCount).  Select(x => x).ToArray();  if (classRooms.Length > 0)  {  ClassRoom[] valid =  new ClassRoom[classRooms.Length];  int count = 0;  foreach (ClassRoom classRoomm  in classRooms)  {  TimeSchedule classRoomSchedule =  Schedule(classRoomm.ID,  "ClassRoomID");  If  (classRoomSchedule.isFree(second.WeekdayID ??  0, second.LectureTimeID ?? 0))  {  valid[count] = classRoomm;  count++;  }  }  if(count > 0)  {  firstClassRoom =  valid.ElementAt(rand.Next(0, count));  msg += "Lecture classroom was changed  to " + firstClassRoom.Name + ". ";  }  }  }  else  firstClassRoom = first.ClassRoom;  if (firstTeacher != null &&  firstClassRoom != null)  {  first.Teacher = firstTeacher;  first.TeacherID = firstTeacher.ID;  first.ClassRoom = firstClassRoom;  first.ClassRoomID = firstClassRoom.ID;  msg += "Change was successful.";  first.WeekdayID = newLectureDayId;  first.LectureTimeID = newLectureTimeId;  }  else  msg = "This change is impossible due to  teachers/classrooms schedule.";  }  else msg = "This change is impossible due to group  schedule.";  }  }  db.SaveChanges();  } |

Algoritmas tikrina paskaitoje dalyvaujančių dalyvių tvarkaraščius tuo laiku, į kurį norima perkelti paskaitą. Esant galimybei perkelia ir išveda sėkmingo perkėlimo pranešimą. Kitu atveju ieško dėl ko negalima perkelti ir pakeičia keitimo trukdį (tik auditorija ar dėstytojas), nebent tai grupė, o tai kirstųsi su realaus gyvenimo logika. Apie eigą sekama informacija ir vartotojas mato ar paskaitos dalyviai buvo pakeisti. Blogiausiu atveju, tarkim, jei visi to modulio dėstytojai yra užimti, išvedamas pranešimas, kad keitimo atlikti negalima.

## Dviejų paskaitų keitimas vietomis

Iš esmės, panašus į algoritmą, kuris keičia paskaitos laiką, tik reikia patikrinti dvi paskaitas, tai reiškia dvigubai daugiau duomenų. Esmė, vėlgi, užtikrinti, kad atlikus keitimą (arba galima žiūrėti, kaip į du, nes keičiami dvejų paskaitų laikai), duomenys duomenų bazėje išliktų korektiški.

|  |
| --- |
| Dviejų paskaitų keitimo vietomis metodas: public void Swap2Lectures(int ID1, int ID2, ref string msg) |
| /// <summary>  /// Apkeicia dvi paskaitas vietomis, jei tai imanoma  /// </summary>  /// <param name="ID1">Pirmos paskaitos id</param>  /// <param name="ID2">Antros paskaitod id</param>  /// <param name="msg">Pranesimas konsolei</param>  public void Swap2Lectures(int ID1, int ID2, ref string msg)  {  Lecture first = db.Lectures.Where(x => x.ID == ID1).  Select(x => x).FirstOrDefault();  Lecture second = db.Lectures.Find(ID2);  if (isPossibleToChange(first, second))  {  msg = "Change was successful.";  int day = first.WeekdayID ?? 0;  int time = first.LectureTimeID ?? 0;  first.WeekdayID = second.WeekdayID;  first.LectureTimeID = second.LectureTimeID;  second.WeekdayID = day;  second.LectureTimeID = time;  }  else  {  if (isPossibleToChangeGroup(first, second) &&  isPossibleToChangeGroup(second, first))  {  Random rand = new Random();  bool isPossibleToChangeFirstTeacher =  isPossibleToChangeTeacher(first, second);  bool isPossibleToChangeSecondTeacher =  isPossibleToChangeTeacher(second, first);  bool isPossibleToChangeFirstClassRoom =  isPossibleToChangeClassRoom(first, second);  bool isPossibleToChangeSecondClassRoom =  isPossibleToChangeClassRoom(second, first);  Teacher firstTeacher = null;  Teacher secondTeacher = null;  ClassRoom firstClassRoom = null;  ClassRoom secondClassRoom = null;  if (isPossibleToChangeFirstTeacher == false)  {  Teacher[] teachers = db.Teachers.  Where(x => x.Module == first.Teacher.Module &&  x.ID != first.TeacherID).  Select(x => x).ToArray();  if (teachers.Length > 0)  {  Teacher[] valid =  new Teacher[teachers.Length];  int count = 0;  foreach (Teacher teacher in teachers)  {  TimeSchedule teacherSchedule =  Schedule(teacher.ID, "TeacherID");  if  (teacherSchedule.isFree(second.WeekdayID ??  0, second.LectureTimeID ?? 0) ||  second.TeacherID == teacher.ID)  {  valid[count] = teacher;  count++;  }  }  if (count > 0)  {  firstTeacher =  valid.ElementAt(rand.Next(0, count));  msg = "First lecture teacher was  changed to " + firstTeacher.LastName + " " +  firstTeacher.FirstName + ". ";  }  }  }  else  firstTeacher = first.Teacher;  if (isPossibleToChangeSecondTeacher == false)  {  Teacher[] teachers = db.Teachers.  Where(x => x.Module == second.Teacher.Module &&  x.ID != second.TeacherID).  Select(x => x).ToArray();  if (teachers.Length > 0)  {  Teacher[] valid =  new Teacher[teachers.Length];  int count = 0;  foreach (Teacher teacher in teachers)  {  TimeSchedule teacherSchedule =  Schedule(teacher.ID, "TeacherID");  If  (teacherSchedule.isFree(first.WeekdayID  ?? 0, first.LectureTimeID ?? 0) ||  first.TeacherID == teacher.ID)  {  valid[count] = teacher;  count++;  }  }  if (count > 0)  {  secondTeacher =  valid.ElementAt(rand.Next(0, count));  msg += "Second lecture teacher was  changed to " + secondTeacher.LastName +  " " + secondTeacher.FirstName + ". ";  }  }  }  else  secondTeacher = second.Teacher;  if (isPossibleToChangeFirstClassRoom == false)  {  ClassRoom[] classRooms;  if (first.IsPcRequired == true)  {  classRooms = db.ClassRooms.  Where(x => x.Type == first.ClassRoom.Type &&  x.ID != first.ClassRoomID && x.Type ==  first.Type && x.NumberOfPlaces >=  first.Group.StudentsCount && x.IsPCavailable  == true).Select(x => x).ToArray();  }  else  classRooms = db.ClassRooms.Where(x => x.Type  == first.ClassRoom.Type && x.ID !=  first.ClassRoomID && x.Type == first.Type &&  x.NumberOfPlaces >=  first.Group.StudentsCount).Select(x =>  x).ToArray();  if (classRooms.Length > 0)  {  ClassRoom[] valid = new  ClassRoom[classRooms.Length];  int count = 0;  foreach (ClassRoom classRoomm in classRooms)  {  TimeSchedule classRoomSchedule =  Schedule(classRoomm.ID, "ClassRoomID");  if  (classRoomSchedule.isFree(second.WeekdayID  ?? 0, second.LectureTimeID ?? 0) ||  second.ClassRoomID == classRoomm.ID)  {  valid[count] = classRoomm;  count++;  }  }  if (count > 0)  {  firstClassRoom =  valid.ElementAt(rand.Next(0, count));  msg += "First lecture classroom was  changed to " + firstClassRoom.Name + ". ";  }  }  }  else  firstClassRoom = first.ClassRoom;  if (isPossibleToChangeSecondClassRoom == false)  {  ClassRoom[] classRooms;  if (second.IsPcRequired == true)  {  classRooms = db.ClassRooms.  Where(x => x.Type == second.ClassRoom.Type &&  x.ID != second.ClassRoomID && x.Type ==  second.Type && x.NumberOfPlaces >=  second.Group.StudentsCount && x.IsPCavailable  == true).Select(x => x).ToArray();  }  else  classRooms = db.ClassRooms.  Where(x => x.Type == second.ClassRoom.Type &&  x.ID != second.ClassRoomID && x.Type ==  second.Type && x.NumberOfPlaces >=  second.Group.StudentsCount).Select(x =>  x).ToArray();  if (classRooms.Length > 0)  {  ClassRoom[] valid =  new ClassRoom[classRooms.Length];  int count = 0;  foreach (ClassRoom classRoomm in classRooms)  {  TimeSchedule classRoomSchedule =  Schedule(classRoomm.ID, "ClassRoomID");  if  (classRoomSchedule.isFree(first.WeekdayID  ?? 0, first.LectureTimeID ?? 0) ||  first.ClassRoomID == classRoomm.ID)  {  valid[count] = classRoomm;  count++;  }  }  if (count > 0)  {  secondClassRoom =  valid.ElementAt(rand.Next(0, count));  msg += "Second lecture classroom was  changed to " + firstClassRoom.Name + ". ";  }  }  }  else  secondClassRoom = second.ClassRoom;  if (firstTeacher != null && secondTeacher != null &&  firstClassRoom != null && secondClassRoom != null)  {  msg += "Change was successful.";  first.Teacher = firstTeacher;  first.TeacherID = firstTeacher.ID;  first.ClassRoom = firstClassRoom;  first.ClassRoomID = firstClassRoom.ID;  second.Teacher = secondTeacher;  second.TeacherID = secondTeacher.ID;  second.ClassRoom = secondClassRoom;  second.ClassRoomID = secondClassRoom.ID;  int day = first.WeekdayID ?? 0;  int time = first.LectureTimeID ?? 0;  first.WeekdayID = second.WeekdayID;  first.LectureTimeID = second.LectureTimeID;  second.WeekdayID = day;  second.LectureTimeID = time;  }  else  msg = "This change is impossible due to  teachers/classrooms schedule.";  }  else msg = "This change is impossible due to groups  schedule.";  }  db.SaveChanges();  } |

Algoritmas gauna dvejų paskaitų ID, pagal juos išsitraukia paskaitas iš duomenų bazės. Tada patikrina abiejų paskaitų dalyvių tvarkaraščius ir jei galima atlieka keitimą, bei atspausdina pranėšimą, apie sėkmingą baigtį. Jeigu nepavyksta, tai tada viskas vykstą, taip pat, kaip būtų keičiamas vienos paskaitos laikas, t. y.: pažiūri dėl ko negalima atlikti keitimo, patikrina ar galima gauti kitas auditorijas ar dėstytojus, pažymi eigą pranešime ir atlikus keitimą. Blogiausiu atveju keitimas nėra atliekamas ir vartotojas gauna pranešimą, kad keitimas atliktas nebuvo ir kokios to priežastys.

## Algoritmas, kuris suranda galimus keitimus

Algoritmas, į kurį užklausa ateina ne spaudant tvarkaraštį, o spaudant virš jo esančius mygtukus. Vartotojas pasirenka paskaitą, pažymi ar nori keisti dėstytoją ir auditoriją paspaudžia find, ir mato sąrašą keitimų, kurie gali būti atlikti keičiant pasirinktą paskaitą. Keitimai rodomi ir su kita paskaita, ir su perkėlimu pasirinktos paskaitos į kitą laiką. Vėl gi, svarbu užtikrinti, kad duomenų bazėje nesipjautų duomenys ir jei pažymėta, kad esant reikalui galima keisti dėstytoją ar auditoriją būtų į ką keisti.

|  |
| --- |
| Galimų keitimų suradimo algoritmas: public List<Lecture> FindPossibleChanges(int ID, bool swapTeacher, bool swapClassroom) |
| /// <summary>  /// Suranda visus galimus keitimo variantus  /// </summary>  /// <param name="ID">Lecture ID, kuriai reikai surasti  keitimus</param>  /// <returns>List galimu keitimu</returns>  public List<Lecture> FindPossibleChanges(int ID,  bool swapTeacher, bool swapClassroom)  {  //Reikia  List<Lecture> possibleChanges = new List<Lecture>();  try  {  Lecture first = db.Lectures.Find(ID);  List<Lecture> lectures = db.Lectures.ToList();  foreach (Lecture lecture in lectures)  {  if (lecture.ID != first.ID && (first.WeekdayID  != lecture.WeekdayID || first.LectureTimeID  != lecture.LectureTimeID))  {    if((isPossibleToChangeGroup(first, lecture)  && isPossibleToChangeGroup(lecture, first))  || first.GroupID == lecture.GroupID)  {  if(FullChange(first, lecture, swapTeacher,  swapClassroom) && FullChange(lecture,  first, swapTeacher, swapClassroom))  {  possibleChanges.Add(lecture);  }  }  }  }  List<Weekday> weekdays = db.Weekdays.ToList();  List<LectureTime> lectureTimes =  db.LectureTimes.ToList();  foreach (Weekday day in weekdays)  {  foreach (LectureTime time in lectureTimes)  {  Lecture artificial = new Lecture();  artificial.WeekdayID = day.ID;  artificial.LectureTimeID = time.ID;  if (isPossibleToChangeGroup(first, artificial))  {    if(FullChange(first, artificial, swapTeacher,  swapClassroom))  {  possibleChanges.Add(artificial);  }  }  }  }  return possibleChanges;  }  catch (NullReferenceException e)  {  //ViewBag.Error = "Duombazėje nėra tokių indeksų: " +  e.Source;  return null;  }  }  /// <summary>  /// Patikrina ar galima sukeisti dvi paskaitas pagal destytojus  ir klases (su vartotojo nurodytais parametrais)  /// </summary>  /// <param name="first">Nurodyta paskaita</param>  /// <param name="second">Viena is kitu paskaitu</param>  /// <param name="swapTeacher">Ar galiam keisti destytoja</param>  /// <param name="swapClassroom">Ar galima keisti  auditorija</param>  /// <returns>true or false</returns>  private bool FullChange(Lecture first, Lecture second,  bool swapTeacher, bool swapClassroom)  {    bool isTeacherFree = false;  bool isClassRoomFree = false;  if (isPossibleToChangeTeacher(first, second))  {  isTeacherFree = true;  }  else  {  if (swapTeacher)  {  Teacher[] teachers = db.Teachers.  Where(x => x.Module == first.Teacher.Module &&  x.ID != first.TeacherID).Select(x => x).ToArray();  if (teachers.Length > 0)  {  foreach (Teacher teacher in teachers)  {  TimeSchedule teacherSchedule =  Schedule(teacher.ID, "TeacherID");  if  (teacherSchedule.isFree(second.WeekdayID  ?? 0, second.LectureTimeID ?? 0))  {  isTeacherFree = true;  break;  }  }  }  }  }  if (isPossibleToChangeClassRoom(first, second))  {  isClassRoomFree = true;  }  else  {  if (swapClassroom)  {  ClassRoom[] classRooms;  if (first.IsPcRequired == true)  {  classRooms = db.ClassRooms.  Where(x => x.Type == first.ClassRoom.Type &&  x.ID != first.ClassRoomID && x.Type ==  first.Type && x.NumberOfPlaces >=  first.Group.StudentsCount &&  x.IsPCavailable == true).Select(x => x).ToArray();  }  else  classRooms = db.ClassRooms.  Where(x => x.Type == first.ClassRoom.Type &&  x.ID != first.ClassRoomID && x.Type ==  first.Type && x.NumberOfPlaces >=  first.Group.StudentsCount).  Select(x => x).ToArray();  if (classRooms.Length > 0)  {  foreach (ClassRoom classRoom in classRooms)  {  TimeSchedule classRoomSchedule =  Schedule(classRoom.ID, "ClassRoomID");  if  (classRoomSchedule.isFree(second.WeekdayID  ?? 0, second.LectureTimeID ?? 0))  {  isClassRoomFree = true;  break;  }  }  }  }  }  if (isClassRoomFree && isTeacherFree)  {  return true;  }  return false;  } |

Taigi, algoritmą sudaro du metodai. Pagrindinis, kuris ciklu eina per esamas paskaitas ir tikrina ar galima pasirinkta paskaitą apkeisti su cikle esančia ar ne. Vėliau tą patį daro su netikromis paskaitomis, kai žiūri ar paskaitą galima perkelti į kitą laiką, nekeičiant jos vietomis su kita paskaita. Jei įjungtos swapTeacher ir/ar swapClassroom vėliavėles yra ieškoma ar bus nors vienas dėstytojas/auditorija laisva tuo metu ir atitiks visus reikiamus parametrus .

## Undo algoritmas

Atšaukimo algoritmas, skirtas atšaukti keitimą. Jis turi atstatyti sistemą į prieš tai buvusią būseną. Nėra taip lengva, nes turi patikrinti ar nesusijauks duomenų bazė, dėl vėliau atliktų keitimų: pvz.: buvo atliktas keitimas nr. 45, po jo buvo atlikti dar keli keitimai, paskutinis nr. 48 ir norint atšaukti 45 keitimą, trukdo kažkas iš 46-48 keitimais pakeistų duomenų.

|  |
| --- |
| Undo metodas: public void Undo(int logItemID) |
| /// <summary>  /// Atsaukia keitima, jei yra imanoma  /// </summary>  /// <param name="logItemID"></param>  public void Undo(int logItemID)  {  Log log = db.Logs.Find(logItemID);  if (log.isUndoable)  {  if (log != null)  {  Lecture first = db.Lectures.Find(log.FirstLectureID);  if (log.SecondLectureID != null)  {  Lecture second =  db.Lectures.Find(log.SecondLectureID);  if (log.OldLectureFirstTeacherID ==  first.TeacherID && log.OldLectureFirstClassroomID  == first.ClassRoomID && log.OldLectureSecondTeacherID  == second.TeacherID &&  log.OldLectureSecondClassroomID == second.ClassRoomID)  {  if (isPossibleToChange(first, second))  {  int day = first.WeekdayID ?? 0;  int time = first.LectureTimeID ?? 0;  first.WeekdayID = second.WeekdayID;  first.LectureTimeID =  second.LectureTimeID;  second.WeekdayID = day;  second.LectureTimeID = time;  db.Logs.Remove(log);  db.SaveChanges();  }  else  {  UndoMessage(first, second, log);  db.SaveChanges();  }  }  else  {  int fTID = first.TeacherID;  int fCID = first.ClassRoomID;  int sTID = second.TeacherID;  int sCID = second.ClassRoomID;  first.TeacherID =  log.OldLectureFirstTeacherID;  first.Teacher =  db.Teachers.Find(first.TeacherID);  first.ClassRoomID =  log.OldLectureFirstClassroomID;  first.ClassRoom =  db.ClassRooms.Find(first.ClassRoomID);  second.TeacherID =  log.OldLectureSecondTeacherID ?? 0;  second.Teacher =  db.Teachers.Find(second.TeacherID);  second.ClassRoomID =  log.OldLectureSecondClassroomID ?? 0;  second.ClassRoom =  db.ClassRooms.Find(second.ClassRoomID);  if (isPossibleToChange(first, second))  {  int day = first.WeekdayID ?? 0;  int time = first.LectureTimeID ?? 0;  first.WeekdayID = second.WeekdayID;  first.LectureTimeID =  second.LectureTimeID;  second.WeekdayID = day;  second.LectureTimeID = time;  db.Logs.Remove(log);  db.SaveChanges();  }  else  {  UndoMessage(first, second, log);  first.TeacherID = fTID;  first.Teacher =  db.Teachers.Find(first.TeacherID);  first.ClassRoomID = fCID;  first.ClassRoomID =  log.OldLectureFirstClassroomID;  second.TeacherID = sTID;  second.Teacher =  db.Teachers.Find(second.TeacherID);  second.ClassRoomID = sCID;  second.ClassRoom =  db.ClassRooms.Find(second.ClassRoomID);  db.SaveChanges();  }  }  }  else  {  if (log.OldLectureFirstTeacherID ==  first.TeacherID &&  log.OldLectureFirstClassroomID ==  first.ClassRoomID)  {  //Paskaitos dalyvių tvarkaraščiai  TimeSchedule teacher =  Schedule(first.TeacherID, "TeacherID");  TimeSchedule classRoom =  Schedule(first.ClassRoomID, "ClassRoomID");  TimeSchedule group = Schedule(first.GroupID,  "GroupID");  //Patikrinam ar laisvi norimu laiku  bool teacherIsFree =  teacher.isFree(log.OldWeekdayID ?? 0,  log.OldLectureTimeID ?? 0);  bool classRoomrIsFree =  classRoom.isFree(log.OldWeekdayID ?? 0,  log.OldLectureTimeID ?? 0);  bool groupIsFree =  group.isFree(log.OldWeekdayID ?? 0,  log.OldLectureTimeID ?? 0);  //Jei visi laisvi, tai apkeičiam  if (teacherIsFree && classRoomrIsFree  && groupIsFree)  {  first.WeekdayID = log.OldWeekdayID ?? 0;  first.LectureTimeID =  log.OldLectureTimeID ?? 0;  db.Logs.Remove(log);  db.SaveChanges();  }  else  {  UndoMessage(first, log);  db.SaveChanges();  }  }  else  {  int fTID = first.TeacherID;  int fCID = first.ClassRoomID;  first.TeacherID =  log.OldLectureFirstTeacherID;  first.ClassRoomID =  log.OldLectureFirstClassroomID;  first.Teacher =  db.Teachers.Find(first.TeacherID);  first.ClassRoom =  db.ClassRooms.Find(first.ClassRoomID);  //Paskaitos dalyvių tvarkaraščiai  TimeSchedule teacher =  Schedule(first.TeacherID, "TeacherID");  TimeSchedule classRoom =  Schedule(first.ClassRoomID, "ClassRoomID");  TimeSchedule group = Schedule(first.GroupID,  "GroupID");  //Patikrinam ar laisvi norimu laiku  bool teacherIsFree =  teacher.isFree(log.OldWeekdayID ?? 0,  log.OldLectureTimeID ?? 0);  bool classRoomrIsFree =  classRoom.isFree(log.OldWeekdayID ?? 0,  log.OldLectureTimeID ?? 0);  bool groupIsFree =  group.isFree(log.OldWeekdayID ?? 0,  log.OldLectureTimeID ?? 0);  //Jei visi laisvi, tai apkeičiam  if (teacherIsFree && classRoomrIsFree  && groupIsFree)  {  first.WeekdayID = log.OldWeekdayID ?? 0;  first.LectureTimeID =  log.OldLectureTimeID ?? 0;  db.Logs.Remove(log);  db.SaveChanges();  }  else  {  UndoMessage(first, log);  first.TeacherID = fTID;  first.ClassRoomID = fCID;  first.Teacher =  db.Teachers.Find(first.TeacherID);  first.ClassRoom =  db.ClassRooms.Find(first.ClassRoomID);  db.SaveChanges();  }  }  }  }  }  } |

Algoritmas priimą logo ID, išsitraukia jį iš duomenų bazės. Atstato sistemą į prieš tai buvusią būseną ir tikrina pagal paskaitos/-ų dalyvių tvarkaraščius tikrina ar galima atšaukti keitimą. Jei galima išsaugo sistemos būseną ir ištrina logą, o jei ne, tai tada suranda ir atspausdina pranešimą, kas trukdo atšaukimui (UndoMessage metodai);

# Testai

Testavimo metu išbandėme tiek kasdienius scenarijus tiek stresines situacijas.

Kasdienio naudojimo testai davė patenkinamus rezultatus, tačiau stresinis testavimas buvo katastrofiškas, supratome kad neveikia didžioji dalis informacijos validavimo mechanizmų.

Detalesnis testavimo apibendrinimas:

|  |
| --- |
| 1. Įrašo, kuriame yra iš labai daug raidžių susidaranti frazė, įrašymas per vartotojo sąsają į lentelę.  Tikėtasi: validacija neleis įvykdyti šio veiksmo.  Gauta: įrašas buvo įrašytas.  Sprendimas: reikės validacijoje realizuoti modulį, kuris reguliuotų maksimalų leistinos frazės ilgį |
| 2.Klaidingai įvesto (perilgo) įrašo vaizdavimas tvarkaraštyje.  Tikėtasi: kad įrašas bus perkeltas į naujas eilutės.  Gauta: įrašas išėjo iš konteinerio ribų.  Sprendimas: redaguoti konteinerio css kodą. |
| 3.Dviejų paskaitų apkeitimas tarpusavyje.  Tikėtasi: paskaitos bus sukeistos vietomis.  Gauta: paskaitos buvo apkeistos. |
| 4.Paskaitos perkėlimas greta kitos paskaitos (t.y. galimybė perkelti paskaitas į ta patį laiką, jei jos vyksta skirtingose auditorijose).  Tikėtasi: paskaitos bus perkeltos.  Gauta: paskaita buvo neperkelta, nes įvykių registravimo metodas pelytes spaudimo prie paskaitos neatpažįsta.  Sprendimas: redaguoti pelės įvykių aptikimo logiką. |
| 5.Paskaitos perkėlimas į laiką, kuriame nėra įrašų.  Tikėtasi: paskaita bus perkelta.  Gauta: paskaita buvo perkelta. |
| 6.Įrašų vedimas su \n simboliu.  Tikėtasi: \n nepadarys string kintamojo sudaryto iš kelių eilučių.  Gauta: vienos eilutės string kintamasis. |
| 7.Sukurti paskaitą, kurioje studentų daugiau nei vietų auditorijoje.  Tikėtasi: validacija neleis įvykti šiai klaidai.  Gauta: paskaitą sukurti pavyko, tačiau generuojant tvarkaraštį ši paskaita automatiškai pasišalina.  Sprendimas: patobulinti validacijos metodą, kad šis neleistų net sukurti tokio tipo paskaitų. |
| 8.Trinti naudojamus žemesnio hierarchinio lygio įrašus duomenų bazėje.  Tikėtasi: neleis trinti, nes tai pakenktų įrašams į kuriuos įeina šis žemesnio lygmens įrašas.  Gauta: įrašas buvo pašalintas, kartu su juo ir aukštesnio lygmens įrašas į kurį įėjo šis įrašas.  Sprendimas: įvesti griežtą privačių raktų kontrolę. |
| 9.Paskaitą, kuriai reikia kompiuterių dėti į auditoriją kurioje jų nėra.  Tikėtasi: validacija neleis įvykti šiai klaidai.  Gauta: paskaitą sukurti pavyko, tačiau generuojant tvarkaraštį ši paskaita automatiškai pasišalina.  Sprendimas: patobulinti validacijos metodą, kad šis neleistų net sukurti tokio tipo paskaitų. |
| 10.Iš duomenų bazės trinti dieną, kurioje yra paskaitų.  Tikėtasi: apsauginiai metodai neleis tam įvykti.  Gauta: programinė išimtis.  Sprendimas: toje vietoje realizuoti išimčių apdorojimo mechanizmą. |

# Šaltiniai

1. ASP.NET MVC gidas: <https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/>
2. Repositorijų pamoka: <https://www.youtube.com/watch?v=0fKg7e37bQE>
3. Genetinis tvarkaraščio generavimo algoritmas: <http://zemris.fer.hr/~golub/clanci/iti2003.pdf>
4. Dar viena implementacija: <https://codereview.stackexchange.com/questions/110872/genetic-algorithm-to-fill-timetable-for-school>
5. Tvarkaraščių generavimo problemos analizė: <https://stackoverflow.com/questions/2177836/algorithm-for-creating-a-school-timetable>
6. Unitime open-source repositorija: <https://github.com/UniTime/unitime>

# Išvados

Šio projekto metu visi patyrėme tą jausmą, kai nepilnai įsigilinęs, kažką nuvertini. Pasirinkome užduotį, kuri mums pasirodė ne itin sunki, tačiau realybėje ši užduotis priskiriama neišsprendžiamų problemų kategorijai. Šio klaidingo pasekmės nebuvo katastrofiškos, tiesiog teko kelis kartus keisti savo tikslą, o tai sugaišo nemažai laiko. Bet galu gale susikoncentravus į paskutinę savo užduotį, savo tikslų pasiekėme.

Dirbdami prie šio projekto dalis komandos narių pagilino savo žinias jau pažįstamose technologijose, dalis pabandė kažko visiškai naujo. Visi pramokome dirbti repositorijoje talpinamo kodo aplinkoje. Tai yra gerai, nes praktikoje tokios žinios yra labai svarbios, kadangi darbas prie didesnio masto projektų, kompanijose, vyksta būtent tokiu principu. Taip pat pagilinome savo žinias darbo su duomenų bazėmis srityje. Sukūrėme lokalią duomenų bazę, sąsają per kuria ją gali valdyti SQL neišmanantis vartotojas. Dirbome ir su ASP.NET MVC technologija. Tai davė daug naudos, nes suteikė galimybę pasikartoti kaip reikia dirbti su C# programavimo kalba, susipažindino su itin plačiai naudojamu MVC programinės įrangos architektūros modeliu. Galu gale reikėtų paminėti, jog patobulėjome darbe su technologijomis, be kurių neįmanomas joks šiuolaikinis web projektas – HTML5 ir CSS3.