Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий

Работа допущена к защите
Руководитель ОП
_____ А.В. Щукин
«_____» _____ 2021 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
РАБОТА БАКАЛАВРА
РАЗРАБОТКА СИСТЕМЫ СОСТАВЛЕНИЯ ПРЕДВАРИТЕЛЬНОГО РАСПИСАНИЯ СЕССИИ

по направлению подготовки 09.03.03 Прикладная информатика
Направленность (профиль) 09.03.03_03 Прикладная информатика в области информационных ресурсов

Выполнил
студент гр. 3530903/70302 Д.В. Сухова

Руководитель
доцент ВШИСиСТ,
к.т.н. А.В. Щукин

Консультант
ассистент ВШИСиСТ В.А. Пархоменко

Санкт-Петербург
2021

САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО

Институт компьютерных наук и технологий

УТВЕРЖДАЮ
Руководитель ОП
_____ А.В. Щукин
«_____» _____ 2021г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы

студенту Суховой Дарье Викторовне гр. 3530903/70302

1. Тема работы: Разработка системы составления предварительного расписания сессии.
2. Срок сдачи студентом законченной работы: 19.05.2021.
3. Исходные данные по работе: таблица аудиторного фонда, данные сайта ruz.spbstu.ru, таблицы с учебными планами.
4. Содержание работы (перечень подлежащих разработке вопросов):
   4.1. Анализ существующих систем составления расписания.
   4.2. Исследование алгоритмов для составления расписания сессии и оптимизации этого процесса.
   4.3. Написание сервиса для получения данных о расписании с использованием API ruz.spbstu.ru
   4.4. Разработка алгоритма составления предварительного расписания сессии.
   4.5. Тестирование и апробация системы составления расписания сессии.
5. Перечень графического материала (с указанием обязательных чертежей):
   5.1. Схема хранения данных.
   5.2. Архитектура разработанной системы.
6. Консультанты по работе:
   6.1. Ассистент ВШИСиСТ, В.А. Пархоменко (содержание и нормоконтроль).
7. Дата выдачи задания: 01.02.2021.

Руководитель ВКР _____ А.В. Щукин

Консультант _____ В.А. Пархоменко

Задание принял к исполнению 01.02.2021

Студент _____ Д.В. Сухова

РЕФЕРАТ

На 68 с., 17 рисунков, 7 таблиц, 13 приложений

КЛЮЧЕВЫЕ СЛОВА: СОСТАВЛЕНИЕ РАСПИСАНИЯ, РАСПИСАНИЕ СЕССИИ, JAVA, SPRING, АРХИТЕКТУРА СИСТЕМЫ.

Тема выпускной квалификационной работы: «Разработка системы составления предварительного расписания сессии».

Целью данной работы является исследование алгоритмов для составления расписания сессии и проектирование сервиса составления предварительного расписания сессии СПбПУ. В ней сравниваются существующие российские и зарубежные системы составления расписания, рассматриваются алгоритмы составления и оптимизации расписания, а также описывается проектирование и реализация системы составления предварительного расписания сессии. За основу алгоритма составления расписания был взят поиск в глубину с проверкой возможности размещения определённой аттестации в конкретной аудитории в заданное время. Система разрабатывалась на языке Java с помощью фреймворка Spring. Для получения информации об установленном расписании университета, без создан модуль обращения к официальному сайту расписания ВУЗа. Для сбора пожеланий преподавателей о проведении сессии с помощью Google API создан модуль для генерации таблиц и форм и чтения информации из них. Система апробирована на летней сессии студентов заочной формы обучения.

ABSTRACT

68 pages, 17 figures, 7 tables, 13 appendices

KEYWORDS: SCHEDULING, JAVA, SPRING, SESSION SCHEDULE, SYSTEM ARCHITECTURE.

The subject of the graduate qualification work is «Development of the system for preliminary scheduling of the session».

The purpose of this work is to study algorithms for scheduling a session and designing a service for scheduling a preliminary schedule of a Polytech session. It compares existing Russian and foreign scheduling systems, examines scheduling and optimization algorithms, and describes the design and implementation of a preliminary session scheduling system. The scheduling algorithm was based on a depth-first search

checking the possibility of placing a certain certification in a specific audience at a given time. The system was developed in Java using the Spring framework. To obtain information about the established timetable of the university, without creating a module for accessing the official website of the timetable of the university. To collect the wishes of teachers about holding a session using the Google API, a module has been created for generating tables and forms and reading information from them. The system was tested at the summer session of correspondence students.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Составление расписания экзаменационных сессий является сложным и кропотливым процессом, который требует автоматизации. Создание программы, которая поможет сотрудникам университета с предварительным распределением экзаменов по датам, времени и аудиториям с учётом пожеланий преподавателей и университетского расписания, сможет значительно ускорить процесс составления расписания. Диспетчер не всегда имеет возможность учесть предпочтения преподавателей. Сложность составления расписания сессии заключается в переборе вариантов расстановки аттестаций и сверкой их с семестровым расписанием во избежание накладок с занятостью кабинетов и преподавателей.

Таким образом, актуальность этой работы обуславливается необходимостью

- формализовать сбор требований профессорско-преподавательского состава к проведению сессии;
- учитывать утверждённое расписание университета, опубликованное на сайте «ruz.spbstu.ru»;
- упростить процесс генерации предварительного расписания путём его автоматизации.

Целью данной работы является исследование алгоритмов для составления расписания сессии и проектирование сервиса составления предварительного расписания сессии СПбПУ. Для достижения этой цели, в необходимо будет решить следующие **задачи**:

- Провести анализ существующих систем составления расписания.
- Рассмотреть задачу составления расписания сессии с алгоритмической точки зрения, исследовать алгоритмы для составления расписания сессии и оптимизации этого процесса.
- Написать сервис для получения данных о расписании с использованием API «ruz.spbstu.ru».
- Реализовать алгоритм составления предварительного расписания сессии.
- Протестировать систему составления расписания.

Практическая значимость работы состоит в автоматизации труда диспетчеров СПбПУ по предварительной компоновке расписания для его последующего размещения на сайте.

Степень разработанности проблемы. Проблема является NP-полной и является широко известной, однако технические особенности реализации для

конкретной организации ранее рассмотрены не были (работа с API сайта расписания СПбПУ для учета существующего расписания).

В главе 1 приведен подробный обзор существующих программных систем по составлению расписания. Глава 2 посвящена обзору алгоритмов, которые можно применять для составления расписания сессии и оптимизации этого процесса. В главе 3 речь идёт о реализации разработанной системы, а в главе 4 описываются методы и результаты её тестирования и апробации.

ГЛАВА 1. АНАЛИЗ СИСТЕМ СОСТАВЛЕНИЯ РАСПИСАНИЯ В ВУЗАХ

Проблема составления расписания не нова, и существует множество средств, призванных упростить её решение. В интернете можно найти онлайн-календари с возможностью совместного редактирования, системы управления бизнес-процессами и программы генерации расписания для разного рода предприятий. Но далеко не все из них могут быть использованы в качестве полноценной системы составления расписания сессии для университета, поэтому в параграфе 1.1 рассматриваются требования к системе составления расписания сессии СПбПУ.

На рынке существует отдельная ниша систем составления расписания для ВУЗов, и в параграфе 1.2 рассматриваются её русскоязычные представители, а в параграфе 1.3 - зарубежные. Далее приведено сравнение таких систем и анализ их преимуществ и недостатков для решения конкретной задачи - составления предварительного расписания сессии в университете.

1.1. Требования к системе составления расписания сессии

1.1.1. Учёт семестрового расписания СПбПУ

Важным аспектом системы составления расписания сессии СПбПУ является учёт занятости аудиторий и преподавателей. Бывают ситуации, когда сессия для некоторых учебных групп начинается в тот момент, когда у других групп всё ещё проводятся семестровые занятия. Чтобы иметь возможность составлять расписание сессии, которое не ставит проведение аттестаций в занятые по семестровому расписанию аудитории, необходимо обращаться с помощью API к официальному расписанию занятий [14].

1.1.2. Роли пользователей системы

Одним из требований к системе сбора сведений является выделение двух ролей пользователей:

- Администратор - пользователь, который сообщает системе сведения о сессии, аудиториях и о плане экзаменов, а также инициирует процесс генерации расписания.
- Преподаватель - пользователь, который сообщает системе о собственных пожеланиях к проведению своих экзаменов.

Рассмотрим возможности этих типов пользователей подробнее. На диаграмме на рисунке 1.1 показаны возможности преподавателя. Преподаватель может участвовать в составлении расписания, указав даты, дни недели и время, когда он доступен или не доступен для проведения аттестаций, необходимые типы аудиторий, своих ассистентов и пожелания по компоновки групп по дням.

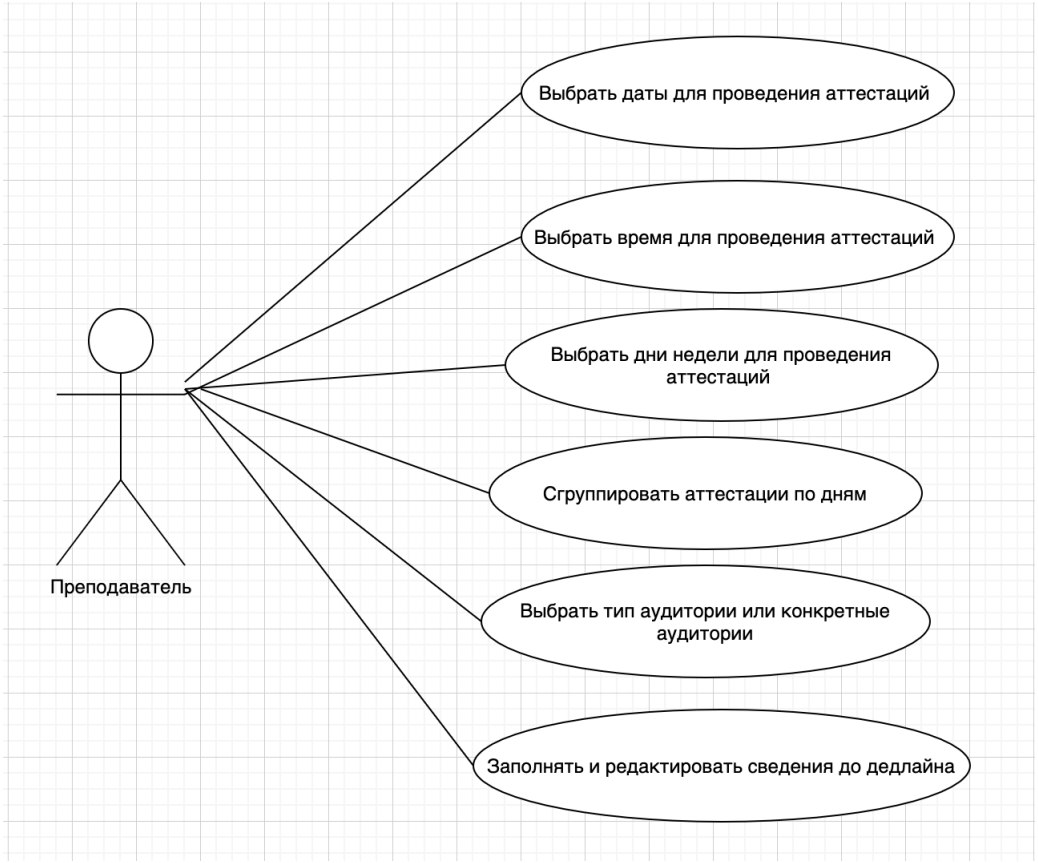

Рис.1.1. Use case диаграмма ППС

Возможности администратора показаны на диаграмме на рисунке 1.2. Администратор заполняет общие сведения о сессии, такие как даты, время, доступные аудитории с указанием, сколько в них мест и есть ли там проектор и компьютеры и предстоящие экзамены по группам и преподавателям. Также, могут быть указаны аттестации, для которых уже определено время и место проведение. Это актуально, например, для событий, проводимых другими подразделениями университета. В возможности администратора помимо вышесказанного входит генерация формы сбора предпочтений преподавателей.

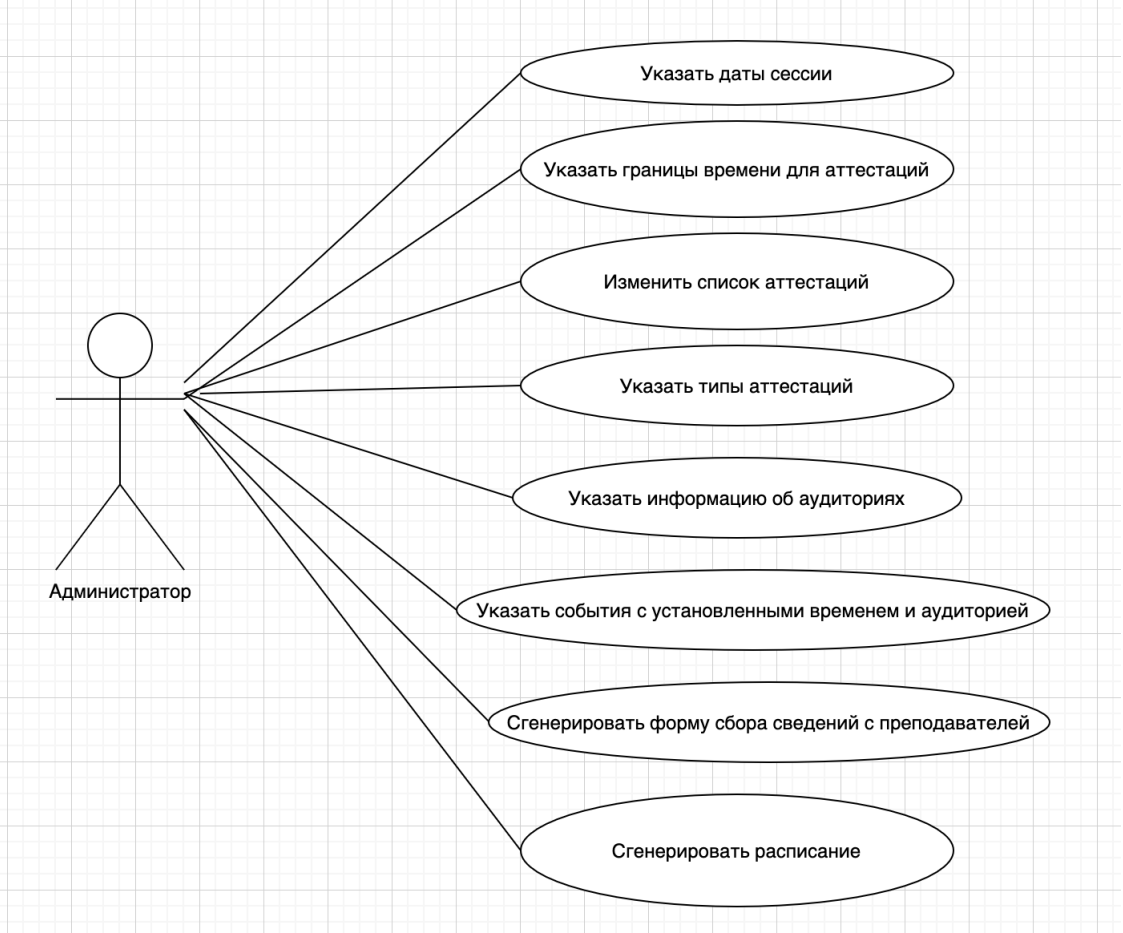

Рис.1.2. Use case диаграмма администратора

Таким образом, когда администратор заполнит все вышеперечисленные поля формы, в системе уже будет минимальный набор данных, необходимый для составления расписания. Далее, преподаватели могут заполнять свои формы с пожеланиями к своему расписанию. Незаполненная преподавателем форма не будет являться проблемой для системы. Пустая форма по умолчанию приравнивается к готовности преподавателя проводить экзамены и зачёты в любой день, в любое время и в любой аудитории.

1.2. Обзор российских систем составления расписания

1.2.1. 1С: ХроноГраф Расписание

Фирма «1С», занимающаяся разработкой ПО для бизнеса и образования, в качестве системы для автоматизации учебного планирования и составления расписания в разного рода организациях предлагает свою программу «1С: ХроноГраф Расписание» [1]. «1С: ХроноГраф Расписание» позволяет:

- составлять понедельное расписание организации или отдельных её подразделений;
- задавать периоды обучения с учётом нерабочих дней, каникул и разбиением на четные и нечётные недели;
- создавать черновое расписание, используя функцию «Предварительный расчёт».

Основной проблемой данной программы является несовместимость с другими платформами. «1С: ХроноГраф Расписание» - однопользовательская программа, и нельзя интегрировать её с web-приложением для возможности сбора данных напрямую от пользователей. Сложность составления расписания сессии в этой системе обуславливается также её ориентированностью на составление расписания по неделям без учёта специфики проведения аттестаций.

1.2.2. Avtor

Программа Avtor («АВТОРасписание») [12] имеет несколько версий для различных учебных заведений: общеобразовательных школ, колледжей, техникумов, профессиональных училищ и ВУЗов. Это позволяет в подстроиться под специфику расписания конкретного типа образовательного учреждения, что является одним из её конкурентных преимуществ.

«АВТОРасписание» имеет достаточно широкий спектр применений. Этот программный продукт позволяет

- составлять понедельное расписание для учебных групп с минимальным количеством окон;
- составлять расписание преподавателей с минимальным количеством окон;
- оптимально размещать занятия по аудиториям, учитывая их вместимость и оснащённость необходимым оборудованием;
- учитывать пожелания сотрудников к своему расписанию;
- разделять учебные группы на подгруппы;
- вносить ручные корректировки в расписание.

Преимуществом этой программы помимо прочего является возможность публиковать расписание обучающихся и преподавателей из самой системы «Автор» на сайте, внутреннем портале или на мультимедийных стендах образовательной организации. Но при этом импорт данных всё ещё производится вручную диспетче-

ром, что не очень удобно для учебного заведения с большим штабом сотрудников, которые сами могли бы вносить свои пожелания в систему.

1.2.3. Галактика Расписание учебных занятий

«Галактика Расписание учебных занятий» - часть системы управления ВУЗом организации Галактика [13]. Этот программный продукт позволяет составлять расписание в ВУЗе, а также:

- вычислять несколько десятков показателей эффективности расписаний;
- оптимально размещать занятия по аудиториям, учитывая их вместимость и оснащённость необходимым оборудованием;
- учитывать приоритет преподавателей, учебных групп и дисциплин;
- контролировать пересечение расписаний для преподавателей, учебных групп и подгрупп во избежание «накладок»;
- контролировать длительность занятий;
- вручную бронировать аудиторный фонд;
- учитывать план изучения дисциплин для выстраивания их в правильном порядке.

«Галактика Расписание учебных занятий» - серьёзный инструмент для формирования расписания в высших учебных заведениях, учитывающий множество факторов при его составлении и имеющий удобную систему отчётности. На данный момент эта программа наиболее полно решает проблему автоматической генерации расписания российских ВУЗов, но и она не имеет интерфейса для прямого импорта пожеланий преподавателей прямо в систему. Компания «Галактика» помимо прочего предлагает техническое сопровождение своего ПО, но это учитывается при расчёте стоимости лицензии на использование программы.

Составление расписания в СПбПУ производится при активном использовании данной программы.

1.3. Обзор зарубежных систем составления расписания

1.3.1. Apereo UniTime

UniTime от компании Apereo [2] - система автоматического создания расписания западных высших учебных заведений. Она учитывает, что студенты

могут выбирать себе индивидуальный набор курсов, и составляет индивидуальное расписание именно для студаентов, а не для учебных групп.

UniTime даёт возможность:

- автоматически генерировать расписание курсов и экзаменов;
- минимизировать конфликты студенческих курсов;
- вносить ручные корректировки в расписание.

Эта программа имеет понятный web-интерфейс и может быть интегрирована в другую систему, но она не позволяет преподавателям вносить данные о своей занятости, чтобы учесть их при составлении расписания. Неприспособленность программы под составление расписания для групп, а не для конкретных студентов делает её менее удобной, чем российские аналоги.

1.3.2. Lantiv Scheduling Studio

Программа «Scheduling Studio» [7] от компании Lantiv представляет собой систему совместной работы над расписанием и реализует следующие задачи:

- совместный доступ к редактированию расписания ВУЗа;
- оффлайн редактирование с возможностью синхронизации после появления в сети;
- цветовое выделение накладок расписания;
- составление расписания на различные временные периоды: неделя, семестр, четверть, год;
- копирование составленных элементов расписания на другие периоды.

Данный программный продукт имеет приятный и понятный интерфейс, но не имеет модуля автоматической генерации расписания, из-за чего основная часть работы всё ещё ложится на плечи диспетчеров. «Scheduling Studio» удобно использовать для составления нетривиального расписания, которое меняется от недели к неделе и плохо вписывается в шаблон школьного расписания или расписания учебных занятий ВУЗа, например. Но для составления расписания сессии требуется большая степень автоматизации, чем предлагается этим ПО.

1.4. Выводы

Сведения о возможностях каждого из описанного в параграфах 1.2 и 1.3 сведём в таблицу 1.1.

Таблица 1.1

Сравнение систем составления расписания

	Учитывает пожелания ППС	Интегрируется с сайтами ВУЗов	Имеет возможность задавать нетривиальное расписание	Плата за использование	Генерация предварительного расписания	Открытый исходный код
1С: ХроноГраф Расписание	+	-	-	+	+	-
Avtor	+	-	+	+	+	-
Галактика Расписание учебных занятий	+	+	+	+	+	-
Apereo UniTime	-	+	+	-	+	+
Lantiv Scheduling Studio	-	-	+	+	-	-

Видно, что среди систем составления расписания для составления предварительного расписания сессии лучше всего могла бы подойти программа «Галактика Расписание учебных занятий», так как она удовлетворяет большинству требований, но при этом она, как и почти всё представленное в таблице ПО, требует оплату за использование. Также среди представленных программных продуктов все, кроме одного, не предоставляют открытый доступ к исходному коду. Рассматривался вариант с доработкой кода Apereo UniTime, но слишком многое в нём было завязано на индивидуальные графики учёбы студентов, а наиболее подходящим расписанием считалось то, которое минимизирует пересечение курсов одного учащегося. В рамках данной работы необходимо было разработать программу, удовлетворяющую всем перечисленным в таблице критериям.

ГЛАВА 2. АЛГОРИТМЫ СОСТАВЛЕНИЯ РАСПИСАНИЯ СЕССИИ

Глава посвящена обзору алгоритмов, которые можно применять для составления расписания сессии, а также для оптимизации этого процесса.

В параграфе 2.1 приведена математическая постановка задачи составления расписания сессии, далее в параграфе 2.2 рассмотрены два возможных режима

составления расписания, а в параграфах 2.3 и 2.4 - алгоритмы, используемые для решения данной задачи.

2.1. Постановка задачи составления расписания сессии

Для составления расписания учебной сессии в университете необходимо каждой запланированной аттестации, запланированной у преподавателя, подобрать день, время и аудиторию проведения. Известны доступные аудитории и множество аттестаций, которые необходимо провести в рамках сессии. Для каждой аудитории известно время ее доступности. Для каждого типа аттестации определена длительность, максимальное количество в день и количество дней отдыха до и после её проведения. Нужно составить расписание сессии так, чтобы оно соответствовало некоторым критериям качества. Далее введём математические обозначения для названных выше параметров, формализуем ограничения и определим, какое расписание будет являться оптимальным.

2.1.1. Обозначения

Составим расписание сессии для каждой из $\overline{1,g}$ учебных групп, которые характеризуются номером и количеством учащихся в ней студентов.

Множество номеров учебных групп:

$$Gr = \{gr_1,...,gr_g\}. \tag{2.1}$$

Количество студентов в соответствующей учебной группе:

$$S = \{s_1,...,s_g\}. \tag{2.2}$$

В данной задаче необходимо учитывать пожелания преподавателей, поэтому каждый преподаватель, помимо ФИО, должен характеризоваться множеством удобных для него дней и часов проведения аттестаций. Также преподавателям необходимо прописывать приоритет, чтобы в случаях, когда нельзя удовлетворить пожеланиям всех преподавателей, в первую очередь будут учитываться пожеланиям именно преподавателей с большим приоритетом.

Множество, состоящее из p имён преподавателей ВУЗа:

$$T = \{t_1,...,t_p\}. \tag{2.3}$$

Приоритеты соответствующих преподавателей:

$$Pr = \{pr_1,..,pr_p\}. \quad (2.4)$$

Множество из q дней, которые преподаватель t выбрал возможными для проведения им аттестаций:

$$Td_t = \{td_{t1},...,td_{tq}\}. \quad (2.5)$$

Множество из c часов, которые преподаватель t выбрал возможными для проведения им аттестаций:

$$Tt_t = \{tt_{t1},...,tt_{tc}\}. \quad (2.6)$$

Важно также учитывать правила проведения сессий, ограничивающие количество аттестаций в день и определяющие, сколько дней отдыха нужно оставить группе до и после аттестации.

Определим множество из y типов аттестаций (экзамен, зачёт и т.п.):

$$A = \{a_1,...,a_y\}. \quad (2.7)$$

Количество дней отдыха перед (Pb) и после (Pa) проведением аттестации каждого типа:

$$Pb = \{pb_1,...,pb_y\} \quad (2.8)$$

$$Pa = \{pa_1,...,pa_y\}. \quad (2.9)$$

Длительность каждого типа аттестации в часах:

$$Ad = \{ad_1,...,ad_y\}. \quad (2.10)$$

Максимальное количество аттестаций каждого типа в день:

$$Ac = \{ac_1,...,ac_y\}. \quad (2.11)$$

При выборе аудиторий для проведения аттестаций необходимо полагаться на их размер и техническую оснащённость. Определим множество из u номеров аудиторий:

$$R = \{r_1,...,r_u\}. \quad (2.12)$$

Типы соответствующих аудиторий (компьютерный класс, имеет проектор и т.п.):

$$Rt = \{rt_1,...,rt_y\}. \tag{2.13}$$

Количество мест в соответствующих аудиториях:

$$Rs = \{rs_1,..,rs_y\}. \tag{2.14}$$

Аттестации, которые необходимо провести в течение сессии описываются учебной группой, дисциплиной и преподавателями, которые должны провести данную аттестацию. Так же для каждой аттестации необходимо указать её тип и тип аудитории, в которой она должна проводиться.

Множество аттестаций:

$$E = \{e_1,...,e_k\}. \tag{2.15}$$

Множества групп (Eg) и дисциплин (Ec) для каждой из k аттестаций:

$$Eg = \{eg_i \in Gr, \forall i \in \{1,..,k\}\} \tag{2.16}$$

$$Ec = \{ec_1,...,ec_k\}. \tag{2.17}$$

Множества типов аттестаций и типов аудиторий для каждой из k аттестаций:

$$Ea = \{ea_i \in A, \forall i \in \{1,..,k\}\} \tag{2.18}$$

$$Er = \{er_1,...,er_k\}. \tag{2.19}$$

Множество, состоящее из преподавателей, проводящих аттестацию e:

$$Et_e = \{et_{ei} \in T, \forall i\}. \tag{2.20}$$

Каждой аттестации необходимо сопоставить дату, часы и аудиторию для проведения. Декартово произведение всех возможных дат, часов и свободных кабинетов представляет собой множество потенциальных окон для проведения аттестаций.

Множество окон:

$$W = \{w_1,..,w_n\}, \tag{2.21}$$

для которых $wd_i \in D$ - календарный день, соответствующий окну i, $wh_i \in H$ - время, соответствующее окну i, $wc_i \in R, \forall i \in \{1,...,n\}$ - аудитория, соответствующая окну i.

Введём множество Solutions, состоящее из функций булевых переменных, которые принимают значение 1, если за i-ей аттестацией бронируется j-е окно:

$$S(i,j) = \begin{cases} 1 & \text{the attestation } e_i \text{ is held in window } w_j \\ 0 & \text{the attestation } e_i \text{ is not held in window } w_j. \end{cases} \quad (2.22)$$

Таким образом, необходимо найти значения функции $S \in Solutions$ для $\forall i \in \{1,..,k\}$ и $\forall j \in \{1,..,n\}$ при условии соблюдения ряда ограничений (см. пункт 2.1.2) и оптимизации ряда критериев качества: длительности сессии для каждого преподавателя/учебной группы, суммарной длительности минимального отдыха между событиями по группам и суммарного количества рабочих дней преподавателей в период сессии (см. пункт 2.4.1). Множество пар <i,j>, для которых S(i,j) = 1, представляют собой расписание сессии - соответствие аттестации дню, времени и аудитории.

Программно описанные в этом параграфе множестве реализованы в качестве классов, представленных в приложении 2. Далее подробно рассмотрим ограничения решаемой оптимизационной задачи.

2.1.2. Ограничения

Задача составления расписания сессии имеет ряд физических ограничений, а также ограничений, обусловленных правилами проведения сессии. Соблюдение этих ограничений позволит составить корректное расписание сессии:

ПА: Иногда не совсем понятны обозначения ограничений.

Во-первых, в одной аудитории в одно время может проводиться максимум одна аттестация. То есть для любого из n окон справедливо, что среди всех k аттестаций ему можно выставить не более одной аттестации:

$$\forall j \in \{1,..,n\} \sum_{i=1}^{k} S(i,j) \leqslant 1. \quad (2.23)$$

Каждая аттестация в период сессии должна проводиться единожды, так как в данном случае не рассматривается расписание дополнительной сессии:

$$\forall i \in \{1,..,k\} \sum_{j=1}^{n} S(i,j) = Ad_e a_i, \quad (2.24)$$

где $Ad_e a_i$ - длительность i-ей аттестации.

Преподаватель не должен отрабатывать в определённый день больше некоторого количества часов x:

$$\forall b \in \{1,..,p\} : \sum_{\forall i \in \{1,..,k\}, et_{ia}=t_b} \sum_{\forall j \in \{1,..,n\}} \sum_{\forall u \in d_u \in D, wd_j=d_u} S(i,j) \leqslant x. \tag{2.25}$$

Любая группа не может сдавать аттестации любого типа в день больше, чем максимальное число, обусловленное типом аттестации:

$$\forall b \in \{1,..,g\}, \forall j \in \{1,..,n\}, \forall m \in \{1,..,y\} : \sum_{\forall i \in \{1,..,k\}, et_i=a_m, eg_i=gr_b} \sum_{\forall d \in D, wd_j=d} S(i,j) \leqslant ac_m * ad_m, \tag{2.26}$$

где $ac_m * ad_m$ - общее число часов, разрешённое на проведение у одной учебной занятий m-ого типа аттестации.

Любая группа перед любой своей аттестацией не должна иметь аттестаций в окно отдыха перед аттестацией этого типа:

$$\forall b \in \{1,..,g\}, \forall i \in \{1,..,k\} eg_i = gr_b : \max_{\forall j \in \{1,..,n\}, ed_j < ed_i} (ed_j) < ed_i - pb_{et_i}. \tag{2.27}$$

Любая группа после любой своей аттестацией не должна иметь аттестаций в окно отдыха после аттестацией этого типа:

$$\forall b \in \{1,..,g\}, \forall i \in \{1,..,k\} eg_i = gr_b : \min \forall j \in \{1,..,n\}, ed_j > ed_i(ed_j) > ed_i - pa_{et_i}. \tag{2.28}$$

Каждая аттестация должна проводиться в аудитории равной и или превосходящей по вместимости размеру группы. То есть для любого окна справедливо, что из факта, что размер группы превышает размер аудитории, следует, что нельзя проводить данное занятие в это окно:

$$\forall i \in \{1,...,k\}, \forall j \in \{1,..,n\} : s_{eg_i} > rs_{wc_j} \rightarrow S(i,j) = 0. \tag{2.29}$$

Каждая аттестация не может проводиться в аудитории с оснащённостью меньшей ожидаемой:

$$\forall i \in \{1,...,k\}, \forall j \in \{1,..,n\} : er_i < rt_{wc_j} \rightarrow S(i,j) = 0. \tag{2.30}$$

Чтобы составить расписание, комфортное для преподавательского состава, учтём некоторые ограничения связанные с предпочтениями преподавателей. Эти

ограничения уже не столь критичны, как перечисленные выше, так как их нарушение не влечёт за собой нарушение правил или законов физики, но их наличие делает расписание более удобным для сотрудников.

Каждый преподаватель волен указать список календарных дней, в которые он готов принимать аттестации. Идеальное расписание предполагает, что любой преподаватель проводит все свои аттестации только в дни из этого списка:

$$\forall t \in T, \forall d \in D : d \notin Td_t \rightarrow S(i,j) = 0. \tag{2.31}$$

Также, преподаватель может указать удобные для него часы проведения аттестаций. Идеальное расписание учитывает это и располагает аттестации так, чтобы они затрагивали только выбранные часы каждого преподавателя:

$$\forall t \in T, \forall h \in H : h \notin Tt_h \rightarrow S(i,j) = 0. \tag{2.32}$$

Данная задача относится к классу NP-полных задач, а значит в худшем исходе придётся перебрать все возможные k аттестаций и n окон в качестве аргументов функции S(i,j), где выполняются все описанные выше ограничения. Обобщим их для функции S:

$$S(i,j) = \begin{cases} \forall j \in \{1,..,n\} \sum_{i=1}^{k} S(i,j) = 1; \\ \forall i \in \{1,..,k\} \sum_{j=1}^{n} S(i,j) = Ad_e a_i; \\ \forall b \in \{1,..,p\} : \\ \sum_{\forall i \in \{1,..,k\}, et_{ia}=t_b} \sum_{\forall j \in \{1,..,n\}} \sum_{\forall u \in d_u \in D, wd_j=d_u} S(i,j) \leqslant x; \\ \forall b \in \{1,..,g\}, \forall j \in \{1,..,n\}, \forall m \in \{1,..,y\} : \\ \sum_{\forall i \in \{1,..,k\}, et_i=a_m, eg_i=gr_b} \sum_{\forall d \in D, wd_j=d} S(i,j) \leqslant ac_m * ad_m; \\ \forall b \in \{1,..,g\}, \forall i \in \{1,..,k\} eg_i = gr_b : \\ \max\limits_{\forall j \in \{1,..,n\}, ed_j < ed_i} (ed_j) < ed_i - pb_{et_i}; \\ \forall b \in \{1,..,g\}, \forall i \in \{1,..,k\} eg_i = gr_b : \\ \min\limits_{\forall j \in \{1,..,n\}, ed_j > ed_i} (ed_j) > ed_i - pa_{et_i}; \\ \forall i \in \{1,...,k\}, \forall j \in \{1,..,n\} : s_{eg_i} > rs_{wc_j} \rightarrow S(i,j) = 0; \\ \forall i \in \{1,...,k\}, \forall j \in \{1,..,n\} : er_i < rt_{wc_j} \rightarrow S(i,j) = 0 \end{cases} \tag{2.33}$$

Оптимальным расписанием будет такое, где минимальное количество пожеланий преподавателей игнорируется. Таким образом, сведём задачу к поиску такого расписания S, на котором выполняется следующий минимум:

$$\min_{\forall S \in Solutions} \Big(\sum_{\forall i \in \{1,\ldots,k\}, \forall j \in \{1,..,n\}, wd_j \notin Td_{et_i}, wh_j \notin Th_{et_i}} (S(i,j) * pr_{et_i}) \Big). \qquad (2.34)$$

Видно, что идеальными расписаниями будут те, где учитываются пожелания всех преподавателей, так как в этом случае точно достигается наименьшее значение минимума формулы (2.34) = 0.

Для решения данной задачи рассмотрим алгоритмы динамического и линейного программирования, алгоритм на графах, эволюционный алгоритм и метод численной оптимизации.

2.2. Режимы алгоритмов составления расписания

Алгоритмы составления расписания можно использовать в двух режимах: инкрементальном и пакетном (batch-режим). В данном разделе рассмотрим, что из себя представляет каждый из этих режимов и почему пакетный режим больше подходит для решения задачи составления расписания сессии.

2.2.1. Инкрементальный режим

Инкрементальным режимом назовём выполнение алгоритма многократно при появлении новых данных. В контексте данной задачи это означает, что каждый раз, когда новый преподаватель будет добавлять информацию о своих предпочтениях, алгоритм будет просчитывать возможные варианты расписания с учётом:

– уже имеющихся данных, которые имеют приоритет, перед новыми;

– внесённых преподавателем изменений.

Так, при каждом новом пересчёте уже составленные расписания преподавателей считаются утверждёнными, что даёт возможность не считать их заново, тем самым ускорив работу алгоритма.

Из преимуществ данного подхода можно выделить возможность преподавателя, не дожидаясь других, увидеть своё расписание. Также, с точки зрения организации, плюсом можно считать стимул преподавателей как можно раньше заполнить форму сбора информации, чтобы иметь приоритет перед другими.

Таким образом появляется возможность раньше закончить процесс составление расписания.

Недостатки данного режима более существенны на практике, чем его преимущества, так как основным недостатком является закрепление наиболее удобного расписания за теми, кто заполнил форму сбора раньше остальных, что может повлечь коллизии. Можно рассмотреть случай, когда первый преподаватель, заполнивший форму сбора, выбирает большое окно для проведения экзаменов и алгоритм бронирует за ним несколько определённых случайных дат, а остальные оставляет свободными. Тогда второй преподаватель, уезжающий на конференцию в свободные даты и готовый провести экзамены в даты, занятые первым, уже не сможет это сделать, потому что у первого преподавателя приоритет, и его уже нельзя подвинуть на сводные даты.

2.2.2. Пакетный режим

Пакетным режимом в данном случае будет единоразовая обработка всех полученных сведений. Для этого необходимо определить дедлайн для сбора пожеланий преподаватель, при наступлении которого составить предварительное расписание для всех за один вызов алгоритма.

Преимуществом такого подхода является возможность установить приоритеты преподавателей и групп, на основе которых можно решать коллизии в случае невозможности нахождения идеального решения, учитывающего пожелания каждого.

Недостатком такого подхода можно назвать невозможность до дедлайна получить расписание, но в реальности это не будет глобальным минусом, потому что составленное таким образом предварительное расписание, всё равно, в последствии может быть скорректировано вручную.

Таким образом, из двух рассмотренных режимов для решения задачи составления расписания больше подходит именно пакетный, потому что его преимущества более значимы, а единственный недостаток не играет роли на практике, в отличие от описанных недостатков инкрементального режима.

2.3. Обзор алгоритмов составления расписания сессии

2.3.1. Обход графа в глубину для поиска идеальных решений

Задача составления расписания сессии является NP-полной, как, например, задача обхода графа, для которой существует множество классических алгоритмов, решающих её. Алгоритм обхода графа в глубину, описанный в книге «The Art of Computer Programming» [6], является одним из них. Далее Но чтобы применить этот алгоритм для составления расписания, необходимо свести эту задачу к задаче построения графа, у которого в качестве вершин выступают элементы расписания - аттестации и временные окна для их проведения. Наличие ребра обуславливается выполнением двух следующих условий:

- ребро из вершины с аттестацией i из отсортированного списка аттестаций E, может быть соединено только с аттестацией i+1 из E;
- ребро из вершины <i,j> с аттестацией i и окном j в вершину <i+1, q> может существовать, если выполняются условия формул (2.33), (2.31) и (2.32) для S(i+1, q) при n=i+1.

Тогда решением будет являться связный граф состоящий из k вершин - пар аттестаций и временных окон, отведённых для этих аттестаций.

Основная идея обхода в глубину – когда возможные пути по ребрам, выходящим из вершин, разветвляются, нужно сначала полностью исследовать одну ветку и только потом переходить к другим веткам. Сложность классического обхода в глубину с матричным заданием графа равна $O(m^2)$, где $m = n * k$ - количество вершин. Но так как известно, что доступность рёбер для каждой из вершин с аттестацией i ограничена n вершинами с аттестацией i+1, сложность снизится до $O(n^2 * k)$. Но так как для проверки остальных условий необходимо проверять i предыдущих вершин графа, сложность останется $O(n^2 * k * \log(k))$.

В процессе решения значениями функции S(i,j) будет заполняться булева матрица. Важно помнить, что значение 1 в ячейке может быть проставлено, только при соблюдении ограничений системы (2.33) , и стоит заметить, что выполнение ограничения, что в одной аудитории в одно время может проводиться только одна аттестация, влечёт за собой наличие в одном столбце матрицы не более одного значения 1, из-за чего матрицу можно заменить на целочисленный массив, где индексы соответствуют индексу возможного окна, а значения - индексу события, которое будет там проведено.

Будем считать, что идеальное решение найдено, если в каждой строке матрицы есть значение 1 или каждое число от 0 до k встречается в массиве решения, что означает, что каждую аттестацию можно сопоставить какому-то окну w_j с учётом перечисленных выше ограничений. Такое решение добавляется в список всех идеальных решений. Далее представлен псевдокод процедуры FindSolutions 2.1. , которая реализует алгоритм обхода графа в глубину. Заметим, что в списке доступных окон W все элементы отсортированы по дням, кабинетам и часам, чтобы легко можно было оценивать занятость одного кабинета несколько часов подряд.

В процедуре FindNextStartTime 2.4 проводится проверка условий из формулы (2.33), чтобы подобрать следующее подходящее ребро, исходящее из указанной вершины. В случае, если для вершины не существует исходящего ребра, которое удовлетворяет всем условиям, данная процедура вернёт значение -1, в противном случае вернёт следующее ребро. В данной функции также происходит проверка условий из формул (2.31) и (2.32), чтобы обеспечить поиск идеального решения, удовлетворяющего пожеланиям всех преподавателей.

Algorithm FindSolutions

Input: $\mathbb{E}$, $\mathbb{W}$, $k = |\mathbb{E}|$, $n = |\mathbb{W}|$

Output: $\mathbb{S}olutions$

```
1.  Solutions ← {}, S ← [], i ← 0
2.  while i < k do
3.      time = NextTime(i)
4.      for ∀u ∈ 0,...,k do
5.          if S[u] = i then
6.              S[u] ← −1
7.      nextTime = FindNextStartTime(i,time)
8.      if nextTime = −1 then
9.          if i = 0 then
10.             return Solutions
11.         else
12.             i ← i − 1
13.     else
14.         duration = ed_i
15.         for ∀b ∈ {nextTime,...,nextTime + duration} do
16.             S[b] ← i
17.         i ← i + 1
18.     if i = k then
19.         Solutions ← Solutions ⋃ S
20.         i ← i − 1
21. return Solutions
```

Рис.2.1. Псевдокод алгоритма DFS для составления расписания

```
Function FindNextStartTime
    Input: event, time, 𝔼, 𝕎, 𝕊, k = |𝔼|, n = |𝕎|
    Output: t
1.  if time < 0 then
2.      return-1
3.  for i ∈ {time,...,k} do
4.      w ← W[i]
5.      if S[i] = −1 ∧ rt_w = er_event ∧ i + ad_event < n ∧ s_eg_event ⩽ er_w ∧ er_w =
          er_W[i+ad_event] ∧ ed_w = ed_W[i+ad_event] ∧ wd ∈ Td_event ∧ wt ∈
          Tt_event ∧ wt + ed_w ∈ Tt_event then
6.          teacherTime ← 0, countPerDay ← 0, j ← 0
7.          while i < k do
8.              if S[j] = −1 then
9.                  j ← j + 1
10.                 continue
11.             if w = S[j] ∧ wt ∈ et_solution[j] then
12.                 break
13.             if et_S[j] = et_event then
14.                 teacherTime = teacherTime + 1
15.                 if teacherTime > maxTimePerDay − ed_event then
16.                     break
17.             if eg_S[j] = eg_event then
18.                 if (ea_S[j]=ea_event) ∧ (wd = ed_S[j]) then
19.                     countPerDay ← countPerDay + 1
20.                     if (ac_et_event < countPerDay) ∨ (wt − pb_ea_event S[j] ∧
                          wt + pa_ea_event >= wd_S[j]) then
21.                         break
22.                 k ← k + 1
23.         if j = n then
24.             return i
25.     return-1
```

Рис.2.2. Проверка условий формул (2.33), (2.31) и (2.32)

2.3.2. Алгоритм обхода графа в глубину с учётом приоритетов преподавателей

На практике возникают ситуации, когда недостаток аудиторий и накладки в личных расписаниях преподавателей не позволяют найти такое расписание, которое удовлетворит всем пожеланиям. По этой причине приходится учитывать приоритеты ограничений и минимизировать потери по формуле (2.34).

Существует ряд ограничений, поступиться с которыми нельзя. В их число входят правила проведения аттестаций и физические условия, связанные с проведением экзаменов в аудиториях, опписанные в формуле (2.33). Но есть и менее жёсткие факторы - это пожелания преподавателей к датам и времени экзаменов.

Каждый преподаватель указывает дни и время, когда ему было бы удобно присутствовать на аттестации. В некоторых случаях удобство эквивалентно тому, что в остальные дни преподаватель в принципе не может принимать экзамены. Приоритет таких ограничений должен быть выше, чем у случаев, когда преподавателю просто комфортнее было бы присутствовать на экзаменах в определённые дни.

Для этого было введено множество приоритетов Pr, где больший приоритет соответствует большему влиянию учёта пожеланий преподавателя на потери функции (2.34). Система приоритетов в случае появления коллизий при составлении расписания позволит в первую очередь пытаться нарушить только пожелания с наименьшим приоритетом и только, если это необходимо, с более высокими.

Модернизируем алгоритм поиска «идеального» решения:

Для этого нужно хранить массив ignoreWishes, в котором будут фиксироваться те аттестации, для которых приходится искать решения без учёта предпочтений преподавателей.

Также, в алгоритме 2.3 на строке 2 необходимо отсортировать массив аттестаций по приоритетам преподавателей, чтобы искать решения без учёта высокоприоритетных преподавателей лишь в последнюю очередь.

Метод поиска решений FindSolutions 2.3 теперь содержит условный оператор, который в случае определения того, что идеального решения найти не удаётся, переходит в режим игнорирования пожеланий преподавателя для конкретного события. Для этого в массиве ignoreWishes выставляется true по индексу, соответствующему этому событию.

Algorithm FindSolutions

Input: $\mathbb{P}, \mathbb{E}, \mathbb{W}, k = |\mathbb{E}|, n = |\mathbb{W}|$

Output: $\mathbb{S}$

1. $ignoreWishes \leftarrow [], S \leftarrow [], i \leftarrow 0$
2. $Sort(E)$
3. **while** $i < k$ **do**
4. $\quad time = NextTime(i)$
5. $\quad$ **for** $\forall u \in 0,...,k$ **do**
6. $\quad\quad$ **if** $S[u] = i$ **then**
7. $\quad\quad\quad S[u] \leftarrow -1$
8. $\quad nextTime = FindNextStartTime(i,time)$
9. $\quad$ **if** $ignoreWishes[i] = False \wedge nextTime = -1$ **then**
10. $\quad\quad ignoreWishes[i] = True$
11. $\quad\quad nextTime = FindNextStartTime(i,0)$
12. $\quad$ **if** $nextTime = -1$ **then**
13. $\quad\quad$ **if** $i = 0$ **then**
14. $\quad\quad\quad$ **return** $null$
15. $\quad\quad$ **else**
16. $\quad\quad\quad ignoreWishes[i] = False$
17. $\quad\quad\quad i \leftarrow i - 1$
18. $\quad$ **else**
19. $\quad\quad duration = ed_i$
20. $\quad\quad$ **for** $\forall b \in \{nextTime,...,nextTime + duration\}$ **do**
21. $\quad\quad\quad S[b] \leftarrow i$
22. $\quad\quad i \leftarrow i + 1$
23. **return** S

Рис.2.3. Псевдокод алгоритма DFS для составления расписания с учётом приоритетов преподавателей

Метод поиска следующего подходящего времени и аудитории теперь может работать в двух режимах. В режими игнорирования рассматриваются только те окна, которые хотя бы в какой-то мере не удовлетворяют пожеланиям преподавателя. Для этого в алгоритме 2.4 в строке 5 используется логический оператор XOR между результатом проверки соблюдения пожеланий преподавателя и значением в массиве игнорирования.

Function FindNextStartTime

Input: $event, time, \mathbb{E}, \mathbb{W}, \mathbb{S}, k = |\mathbb{E}|, n = |\mathbb{W}|, ignoreWishes$

Output: t

1. **if** $time < 0$ **then**
2. **return**-1
3. **for** $i \in \{time,...,k\}$ **do**
4. $w \leftarrow W[i]$
5. $teacherWish \leftarrow (wd \in Td_{event} \wedge wt \in Tt_{event} \wedge wt + ed_w \in Tt_{event}) \nrightarrow ignoreWishes[i]$
6. **if** $S[i] = -1 \wedge rt_w = er_{event} \wedge i + ad_{event} < n \wedge s_{eg_{event}} \leqslant er_w \wedge er_w = er_{W[i+ad_{event}]} \wedge ed_w = ed_{W[i+ad_{event}]} \wedge teacherWish$ **then**
7. ...
8. **if** $j = n$ **then**
9. **return** i
10. **return**-1

Рис.2.4. Проверка условий формулы (2.33) с учётом режима игнорирования

Асимптотическая сложность этой интерпретации алгоритма тоже равна $O(n^2 * k)$, но требует дополнительную память на хранение бинарного массива ignoreWishes.

2.3.3. Генетический алгоритм

По данным книги «Evolutionary Multiobjective Optimization. Theoretical Advances and Applications» [15], генетический алгоритм имитирует естественный отбор и состоит из нескольких этапов:

- создание популяции;
- размножение путём обмена генами у двух особей;

- мутации путём проведения определённых изменений генов некоторых особей;
- расчёт метрики приспособленности для особей и отбор лучших для перехода в новое поколение.

Определим основные понятия генетического алгоритма для простого случая составления расписания. Ген будет представлять собой объект с полями: Группа, Преподаватель, Предмет, Тип аттестации, Дата, Время, Аудитория. Популяция - все возможные комбинации расположения аттестаций по аудиториям во времени. Особь - некоторый набор генов (пример расписания). Такая задача сводится к нахождению наилучшей особи - расписания.

Фитнес-функция для этого алгоритма, в которой задаётся способ расчёта метрики качества расписания, должна учитывать ограничения (2.33), чтобы с каждым новым поколениям расписания имели всё меньше и меньше накладок.

Из преимуществ генетического алгоритма можно выделить:

- Адаптивность времени выполнения;
- Возможность ограничить количество поколений алгоритма, по истечении которых алгоритм оставит наиболее подходящее расписание;
- Адаптивность качества;
- Если задать условием остановки алгоритма - достижение некоторого счёта в фитнес-функции, можно завершить работу с приемлемыми потерями, полученными, например, от пренебрежения пожеланиями преподавателями.

Недостатками генетического алгоритма для задачи составления расписания будут:

- Массивные входные данные, так как для формирования популяции необходимо брать декартово произведение нескольких датасетов, а после манипулировать большой начальной популяцией;
- Нелинейная зависимость качества от длительности выполнения, так как часто в эволюционных алгоритмах последующее поколение бывает менее приспособлено, чем предыдущее, а значит, заранее предугадать количество итераций при заданном минимальном пороге качества нельзя;
- Сложность подбора параметров фитнес-функции;
- Длительная отладка.

2.3.4. Жадный алгоритм

В данном пункте рассматриваю концепцию жадного алгоритма из книги «Совершенный алгоритм. Жадные алгоритмы и динамическое программирование» [3]: жадный алгоритм - алгоритм, который на каждом шаге выбирает локально оптимальное решение, ожидая, что итоговое решение тоже будет оптимальным. В какой-то мере жадным алгоритмом пользуются диспетчеры при составлении расписания сессии: сначала выставляются даты для аттестаций проводимых высокоприоритетными преподавателями-совместителями, а далее в свободные окна расставляются менее приоритетные.

Но программная реализация жадного алгоритма при составлении расписания слишком часто не будет сходиться к решению, которое удовлетворяет всем ограничениям. Во многих задачах допустимо искать не глобальный оптимум решения, чтобы уменьшить время выполнения, но при составлении расписания сессии нельзя пренебрегать рябом ограничений, а значит если на какой-то итерации алгоритма некоторую аттестацию остаётся расположить в занятой аудитории, применение такой оптимизации совершенно теряет смысл. Можно повторять жадный поиск решения несколько, если сразу не удаётся найти подходящее расписание, но так процесс подбора решения потеряет свою "жадность"и выродится к полному перебору.

2.4. Методы оптимизации поиска лучших решений

2.4.1. Метрики оценки качества расписания

Для оценки предложенных расписаний подсчитаем несколько метрик:

Длительность сессии для каждого преподавателя с учётом его приоритета:

$$DurationT = \sum_{\forall t \in T} (pr_t * (\max_{i \in \{0,\dots,n\}, et_{S[i]}=t} (wd_i) - \min_{i \in \{0,\dots,n\}, et_{S[i]}=t} (wd_i))) \qquad (2.35)$$

Считаем, что расписание преподавателя должно быть максимально компактным по датам, а значит разницу между первым и последним днём проведения экзаменов нужно минимизировать.

Суммарная длительность минимального отдыха между экзаменами по группам:

$$Pause = \sum_{\forall gr \in Gr} (\min_{i,j \in \{0,\ldots,n\}, eg_{S[i]}=gr, wd_i \neq wd_j, i>j} (wd_i - wd_j)) \tag{2.36}$$

Считаем, что студентам желательно не иметь маленьких перерывов между экзаменами, поэтому стремимся максимизировать Pause для расписания.

Суммарная длительность сессии по группам:

$$DurationG = \sum_{\forall gr \in Gr} \max_{i \in \{0,\ldots,n\}, eg_{S[i]}=gr} (wd_i) \tag{2.37}$$

Чем раньше закончится сессия, тем раньше у иногородних студентов появится возможность уехать домой, а значит порядковый номер последнего дня сессии для группы нужно пытаться минимизировать.

Суммарное кол-во рабочих дней для преподавателей с учётом их приоритетов:

$$Work_t = \{wd_i, \forall i \; in 1,\ldots,n, et_{S[i]} = t\}; \tag{2.38}$$

$$WDays = \sum_{\forall t \in T} (pr_t * \|Work_t\|) \tag{2.39}$$

Считаем, что преподавателю удобно иметь максимальное количество дней, свободных от проведения экзаменов, а значит метрику WDays стараемся минимизировать. На практике это означает, что лучше провести несколько зачётов в один день, чем заставлять человека несколько раз в неделю приезжать в университет.

Все эти метрики имеют разные шкалы, какие-то мы пытаемся максимизировать, какие-то минимизировать, одни измеряются несколькими неделями, а другие несколькими днями. Поэтому каждую метрику для всех рассматриваемых решений необходимо нормализовать. Это приведёт данные метрики к единой шкале и улучшит их интерпретируемость.

Ко всем метрикам качества для выбранных для сравнения расписаний применим минимаксную нормализацию [16], которая рассчитывается по формуле:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{2.40}$$

Далее сделаем так, чтобы большему значению каждой метрики соответствовало более предпочтительное расписание. Для этого те метрики, для которых лучшее решение имеет меньшее значение - вычтем из 1. Тем самым соотношение всех значений метрики среди расписаний останется таким же, но качество метрики будет расти с качеством решения.

2.4.2. Метод ветвей и границ

Метод ветвей и границ, описанный в «An automatic method of solving discrete programming problems» [10], принадлежит к группе алгоритмов целочисленного линейного программирования и применяется для оптимизации задач полного перебора. Он как нельзя кстати придётся для модернизации алгоритма полного обхода графа в глубину. Суть этого метода состоит в том, чтобы в процессе обхода графа, отбрасывать заведомо менее оптимальные решения, чем у же найденные.

Такой подход потребует выделить память под хранение метрик качества. В памяти необходимо держать качество текущего рассчитываемого расписания и предыдущего лучшего расписания. В качестве метрики качества возьмём разброс между первым и последним днём проведения сессии для каждого преподавателя. Чем он меньше, тем лучше считается расписание. Естественно, можно выбрать и другие критерии качества расписания, но алгоритм при этом поменяется незначительно. Выбор такой метрики вынуждает хранить массив с количеством дней разброса для каждого преподавателя, поэтому выделим память под массивы metrics[] и bestMetrics[] размером p (число преподавателей).

Модернизируем функцию поиска решений методом обхода графа в глубину, добавив в качестве условия перехода к следующей аттестации проверку на то, является ли расписание на данном этапе, худшим, чем предыдущее полностью составленное расписание. Эта проверка добавляется в строке 18 алгоритма 2.5.

Algorithm FindSolutions

Input: $\mathbb{E}$, $\mathbb{W}$, $k = |\mathbb{E}|, n = |\mathbb{W}|$

Output: $\mathbb{S}olutions$

1. $metrics \leftarrow [], bestMetrics \leftarrow [], Solutions \leftarrow \{\}, S \leftarrow [], i \leftarrow 0$
2. **while** $i < k$ **do**
3. $time = NextTime(i)$
4. **for** $\forall u \in 0,...,k$ **do**
5. **if** $S[u] = i$ **then**
6. $S[u] \leftarrow -1$
7. $nextTime = FindNextStartTime(i,time)$
8. **if** $nextTime = -1$ **then**
9. **if** $i = 0$ **then**
10. **return** $Solutions$
11. **else**
12. $i \leftarrow i - 1$
13. **else**
14. $duration = ed_i$
15. **for** $\forall b \in \{nextTime,...,nextTime + duration\}$ **do**
16. $S[b] \leftarrow i$
17. $i \leftarrow i + 1$
18. **if** $i = 0 \vee \neg isItWorse(i)$ **then**
19. $i \leftarrow i + 1$
20. **else**
21. **return** $Solutions$
22. **if** $i = k$ **then**
23. $Solutions \leftarrow Solutions \bigcup S$
24. $bestMetrics \leftarrow metrics$
25. $i \leftarrow i - 1$
26. **return** $Solutions$

Рис.2.5. Псевдокод алгоритма DFS с использованием метода ветвей и границ

Функция isItWorse 2.6 пересчитывает для рассматриваемого преподавателя длительность его сессии и проверяет, не является ли оно худшим, чем у того же преподавателя в предыдущем полном расписании. Для этого в массиве текущего решения находится максимальный и минимальный день, для рассматриваемого преподавателя и возвращается их разница.

Function isItWorse

Input: $event, time, \mathbb{E}, \mathbb{W}, \mathbb{S}, k = |\mathbb{E}|, n = |\mathbb{W}|, ignoreWishes$

Output: t

1. max,$min \leftarrow wd_{n-1}, i \leftarrow event - 1$
2. **while** $i >= 0 \wedge et_i = et_{event}$ **do**
3. $\quad j \leftarrow \min_{j \in 0,k-1, S[j]=i}(j)$
4. $\quad$ **if** $wd_j > max$ **then**
5. $\quad\quad max = wd_j$
6. $\quad$ **else if** $wd_j < min$ **then**
7. $\quad\quad min = wd_j$
8. $\quad i \leftarrow i - 1$
9. $metrics[event] \leftarrow max - min$
10. **return** $bestMetrics[0] \neq -1 \wedge bestMetrics[et_{event}] < metrics[et_{event}]$

Рис.2.6. Функция сравнения метрики качества для преподавателя с предыдущим лучшим решением

2.4.3. Алгоритм имитации отжига

Для выбора лучшего из расписаний и расчёта всех метрик необходимо будет дополнительно обойти x расписаний, состоящих из n возможных временных окон. А потом просуммировать их по p преподавателям и g учебным группам. Если входные данные были такими, что было найдено 5-10 возможных расписаний, то не составит труда просчитать их все, так как относительно n - числа временных окон, оно вносит минимальный вклад в сложность вычислений.

Но рассмотрим случай, когда методом полного перебора было найдено большое количество возможных расписаний, и число x достаточно велико и даже сравнимо с n. В такой ситуации было бы полезно находить более оптимальные решения без расчёта метрик для каждого расписания, жертвуя некоторой погрешностью.

Для этого можно применить оптимизационный алгоритм имитации отжига, описанный в статье «On simulated annealing phase transitions in phylogeny reconstruction» [9]. Он основан на имитации физического процесса отжига металлов - при постепенно понижающейся температуре переход атома из одной ячейки кристаллической решётки в другую происходит с некоторой вероятностью, которая понижается с понижением температуры.

При помощи моделирования этого процесса ищется такая точка или множество точек, на котором достигается минимум некоторой функции F(S). Для задачи выбора лучшего расписания функция F(S) - функция, вычисляющая качество расписания, например, по метрике DurationG, где S - некоторое из полученных расписаний:

$$F(S) = \sum_{\forall gr \in Gr} \max_{i \in \{0,\ldots,x\}, eg_{S[i]}=gr} (wd_i) \tag{2.41}$$

Решение ищется последовательным вычислением точек $S_0,\ldots,S_f$ из множества Solutions(для задачи выбора оптимального расписания - возможные варианты расписаний), где f - количество итераций понижения температуры. Каждая последующая точка претендует на то, чтобы лучше предыдущих приближать решение. В качестве исходных данных возьмём точку S_0. На каждом шаге алгоритм вычисляет новую точку и понижает значение счётчика - температуры Q_i , и когда он достигает нуля, алгоритм останавливается в этой точке.

Температурой для данной задачи возьмём некоторую константу, достаточно большую, чтобы хватило итераций для нахождения примерного оптимума, но при этом меньшую, чем x. Пусть это будет константа $Q_0 = \frac{n}{10}$.

К точке x_i на каждом шаге применяется оператор A, который случайным образом модифицирует её, в результате чего получается новая точка x^*. Он возвращает следующее расписание для расчёта метрики качества на следующей итерации.

$$A(S_i) = S_{Y(0,x-i)}, \text{ Y - random number on a segment [0,x-i]} \tag{2.42}$$

Но перейдёт ли S^* в S_{i+1} произойдёт с вероятностью, которая вычисляется в соответствии с распределением Гиббса:

$$P(x^* \to S_{i+1} \mid S_i) = \begin{cases} 1, & F(S^*) - F(S_i) < 0 \\ \exp\left(-\dfrac{F(S^*) - F(S_i)}{Q_i}\right), & F(S^*) - F(S_i) \geqslant 0 \end{cases} \tag{2.43}$$

В псевдокоде 2.7 представлено создание цикла понижения температуры, который возвращает расписание, которое будет выбрано, когда температура станет нулевой.

Algorithm SimulatedAnnealing

Input: $\mathbb{S}olutions, \mathbb{E}, \mathbb{W}, x = |\mathbb{S}olutions|, k = |\mathbb{E}|, n = |\mathbb{W}|$
Output: $\mathbb{S}$
1. $i \leftarrow 0$
2. $metrics[i] = F(Solutions[i])$
3. **for** $q \in n/10,...,0$ **do**
4. $\quad next = A(Solutions[i])$
5. $\quad metrics[next] = F(Solutions[next])$
6. $\quad pr = P(S_{next}) \mid S_i)$
7. $\quad$ **if** $Y(0,1 < p)$ **then**
8. $\quad\quad i \leftarrow next$
9. **return** $Solutions[i]$

Рис.2.7. Псевдокод алгоритма имитации отжига для определения оптимального расписания из предложенных

Из преимуществ этого метода оптимизации можно выделить его скорость, потому что расчёт метрик качества расписания осуществляется только для установленного числа решений. Но при этом это снижает точность выбора наиболее удачного расписания, так как присутствует фактор случайности при выборе следующего решения на каждом шаге алгоритма.

2.5. Выводы

Система составления расписания сессии СПбПу реализует алгоритм обхода в глубину с учётом приоритетов преподавателей, одним из входных параметров которого будет максимальное количество рассчитанных расписаний. Этот параметр необходим для ускорения работы алгоритма, если не стоит задачи выбирать самое лучшее из всех возможных расписаний. Реализация этого алгоритма представлена в приложении 1.

Если выбрана метрика определения наиболее оптимального расписания, можно применить метод ветвей и границ, чтобы ещё ускорить вычисления и

вернуть меньшее количество наиболее качественных расписаний сессии. Метод имитации отжига плохо применим к оптимизации выбора самого удачного расписания на практике, так как он нацелен на относительно большое число входных вариантов расписания, что во-первых редкость, потому что зачастую предпочтения преподавателей и загруженность помещений не даёт большой выбор расписаний, а во-вторых требует длительного времени на расчёт большого числа расписаний, среди которых будет выбираться лучшее.

ГЛАВА 3. РЕАЛИЗАЦИЯ СИСТЕМЫ СОСТАВЛЕНИЯ ПРЕДВАРИТЕЛЬНОГО РАСПИСАНИЯ

В данной главе речь идёт о ПО, рассматривавшемся в качестве средства разработки системы составления расписания сессии для СПбПУ. В параграфах 3.1 и 3.2 приводится обоснование выбора средств реализации системы таких как язык программирования, фреймворков и СУБД. В параграфе 3.3 приводится архитектура реализованной системы .

3.1. Выбор языка программирования

Так как система составления расписания сессии должна иметь веб-модуль, необходимо подобрать подходящий для этой цели язык программирования. Среди высокоуровневых языков, которые потенциально могли бы сделать процесс разработки более быстрым, а результат качественным рассматривались следующие претенденты:

- Java;
- JavaScript;
- C#;
- C++.

3.1.1. Java

Java - один из самых популярных объектно-ориентированных языков программирования, что всегда является преимуществом, так как количество и качество документации, множество фреймворков и библиотек под любые нужды делает

процесс разработки быстрее и эффективнее. Технология Garbage Collection, реализованная в Java оптимизирует управление памятью программ, что очень важно для работы с большим количеством входных данных, как в задаче составления расписания сессии. Также плюсом данного языка является его кроссплатформенность, а значит код на нём может быть запущен везде, где установлена Java Virtual Machine.

3.1.2. JavaScript

JavaScript - язык программирования, который изначально использовался для разработки фронтенда и с помощью скриптов встраивал в HTML страницы исполняемый код. Сейчас JavaScript можно использовать и в качестве основного языка для написания бэкенда, так как с появлением среды Node.js программы на JavaScript можно запускать и на сервере. Преимуществом JavaScript всё ещё является его ориентированность на работу с веб страницами, а также его простота. Скрипты на этом языке поддерживаются всеми современными браузерами и не требуют установки дополнительного программного обеспечения. В качестве языка для бэкенда недостатком служит отсутствие явной типизации, а так же разработку усложняет тот факт, что нет возможности узнать об ошибке на этапе компиляции, а значит о том, что где-то в коде появилось сложение строки с числом, разработчик узнает только тогда, когда программа выполнится до этого места и не раньше.

3.1.3. C#

C#, как и Java, является кроссплатформенным языком программирования, который разработан для создания приложений среды NET Framework. С точки зрения разработки C# удобен обилием синтаксического сахара, за счёт которого упраздняются громоздкие конструкции, делающие написание кода более быстрым, а также повышающие его читаемость. Это в какой-то мере сказывается на производительности вычислений, но окупается удобством разработки. Помимо этого для C# тоже существует множество фреймворков, что также предоставляет возможность переиспользовать уже готовые технологии.

3.1.4. C++

C++ - объектно-ориентированный язык программирования с возможностью низкоуровневой работы с данными, что даёт возможность писать на нём даже микроконтроллеры. Но сложность синтаксиса и небогатая стандартная библиотека

затрудняет высокоуровневую разработку. C++ достаточно популярен и имеет большое количество документации, но даже с ней не всегда легко правильно самостоятельно обеспечить грамотную работу с памятью. Ещё одним недостатком этого языка является зависимость от платформы и этот фактор перевешивает даже отличную производительность этого языка.

3.1.5. Определение языка программирования

Для реализации сервиса составления предварительного расписания сессии к языку программирования предъявлялись следующие требования:

- наличие подробной документации;
- наличие фреймворков для упрощения работы с базой данных и веб-сервисом программы;
- кроссплатформенность;
- удобство и скорость разработки.

Язык Java удовлетворяет всем перечисленным критериям благодаря своей популярности среди программистов. Это существенно упрощает процесс разработки, так как для Java существует множество пособий, а способы устранения возникающих ошибок уже в большинстве собой описаны на различных интернет-ресурсах. Далее выбор фреймворков будет рассматриваться именно для Java, благо для данного языка их существует достаточно. Но стоит заметить, что язык JavaScript, хоть и не очень хорошо подходит для основного языка в этом проекте, но будет использоваться для создания веб-интерфейса, так как Java не очень хорошо приспособлена для этой задачи.

3.2. Выбор фреймворка для построения веб-приложения

3.2.1. Spring

Spring - Java-фреймворк, состоящий из множества компонентов таких как Inversion of Control для управления объектами, MVC для расширения Servlet API и прозрачного разделения между слоями модель, представление, контроллер и даже компонента доступа к базам данных [8]. Все эти и другие компоненты могут добавляться в проект независимо, что делает разработку более гибкой. Наличие этих модулей избавляет разработчика от низкоуровневой работы с запросами, налаживания взаимодействия с базами данных и прочих моментов имеющих чисто

технический характер и не имеющих связи с бизнес-задачей программы. Ещё одним преимуществом Spring для разработчика является его популярность и качественная документация, чего часто не хватает молодым и мало известным проектам.

3.2.2. Dropwizard

Dropwizard - это фреймворк для создания веб-сервисов RESTful [4]. Он по умолчанию поставляется с такими библиотеками, как Jetty server, Jackson, Metrics, Hibernate Validator и Guava. Dropwizard не имеет русскоязычной документации, что является недостатком. Так же у Dropwizard не так много модулей, как у Spring, но всё ещё просто расширяется с использованием файлов конфигураций.

3.2.3. Vert.x

Vert.x - это асинхронный, событийно ориентированный фреймворк, создатели которого во многом вдохновлялись фреймворком node.js [11]. Он позволяет оптимизировать параллельную отправку и получение запросов, обрабатывая их большее количество с меньшими ресурсами, нежели фреймворки, основанные на блокирующем асинхронном вводе-выводе. Vert.x позволяет работать с действительно нагруженными системами, но не имеет такого же количества модулей для разных целей, как Dropwizard и тем более Spring.

3.2.4. Определение фреймворка для построения веб-приложения

Для написания системы составления расписания сессии СПбПу из рассмотренных фреймворков был выбран именно Spring, потому что обладает всеми необходимыми для данной задачи модулями. Огромное количество библиотек, включённых в Spring может помочь при дальнейшем расширении сервиса без переписывания его на другой фреймворк. Также при поддержании работы системы другими разработчиками не возникнет проблемы с изучением Spring, потому что для него существует множество справочных материалов в том числе на русском языке.

3.2.5. Хранение данных

Spring Framework имеет поддержку ORM Hibernate [5]. ORM или Object–Relational Mapping - это технология, позволяющая сопоставлять объекты объекно–ориентрованной модели, на которой зиждится Java, с сущностями базы данных. Таким образом, не нужно вручную создавать таблицы с полями, соответствующими структуре Java - объекта, ведь можно просто добавить к нему аннотацию, а Hibernate сделает всё автоматически, а самое главное автоматически преобразует полученные результаты select-запроса в объект, что не всегда просто делать самостоятельно. Также, Hibernate имеет предопределённые запросы к базе данных, из-за чего можно не прописывать монотонные sql-запросы на поиск и удаления элемента по id или добавления его в таблицу.

Hibernate поддерживает диалекты и MySQL, и Oracle, и Microsoft SQL Server, и PostgreSQL, поэтому в качестве СУБД была выбрана PostgreSQL, так как она полностью бесплатна и проста в установке на любой платформе.

3.2.6. Выводы

Таким образом, в качестве основного языка программирования был выбран Java, который во фронтенд части дополняется JavaScript, более приспособленным для реализации интерфейсов. Среди фреймворков для работы с веб-приложениями для Java выбор пал на Spring Framework, имеющий множество разным модулей, в том числе Hibernate, с помощью которого осуществляется взаимодействие с базой данной PostgreSQL.

3.3. Архитектура системы

На схеме 3.1 представлена архитектура система составления предварительного расписания сессии СПбПУ.

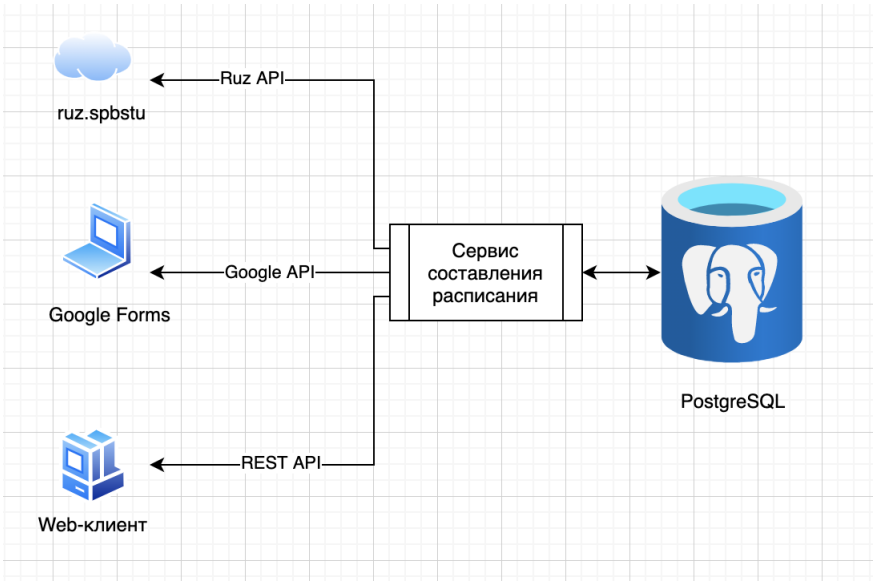

Рис.3.1. Схема архитектуры системы

Администратор имеет доступ к веб-интерфейсу приложения. С его помощью сведения о том, для каких аттестаций, групп, на какие даты и время должно быть составлено расписание. Spring сервер сохраняет эти данные в базе данных. Полученные сведения дополняются информацией с сайта расписания университета по API. На основе имеющихся данных сервер с помощью Google API формирует форму для сбора сведений преподавательского состава. После получения с помощью Google API информации о пожеланиях преподавателей на сервере происходит генерация предварительного расписания и выгрузка его в google-таблицу.

Далее рассмотрим каждое из данных звеньев системы подробнее.

3.3.1. Веб-клиент и UI

Веб-клиент позволяет администратору работать с системой составления расписания. "Общение"пользователя с сервером осуществляется с помощью REST–запросов на сервер, написанных на JavaScript. Этот код представлен в приложении 9.

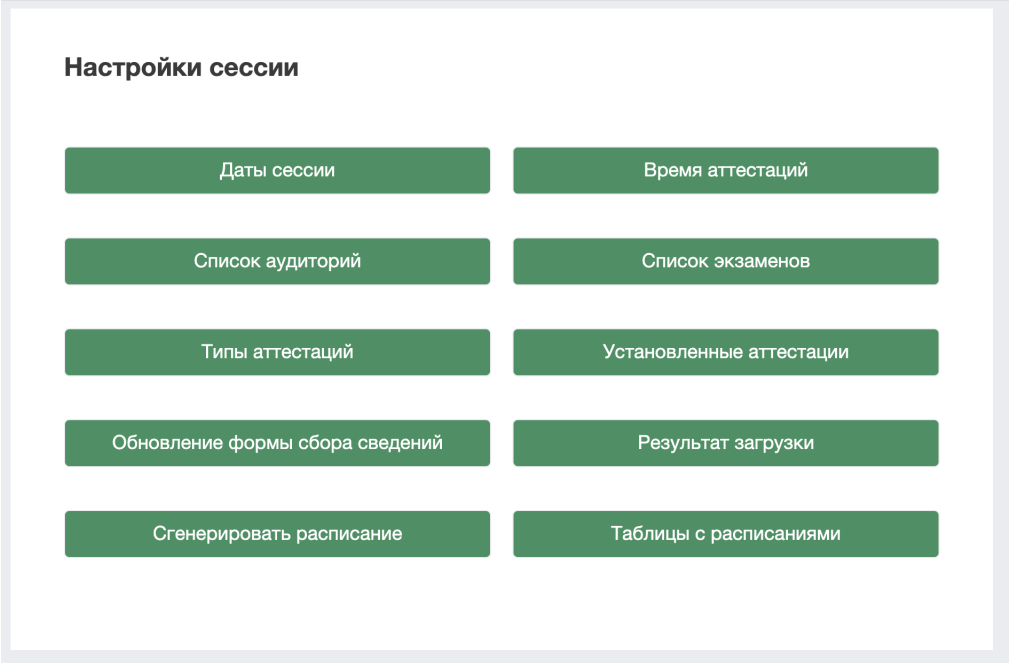

Рис.3.2. Главная страница интерфейса администратора

На рисунке 3.2 представлен перечень действий, который может выполнить администратор. Они реализуются рядом POST-запросов.

POST-запрос к /session/update_date с параметром days позволяет выбрать дни сессии. Форма для выбора дат изображена на скриншоте 3.3.

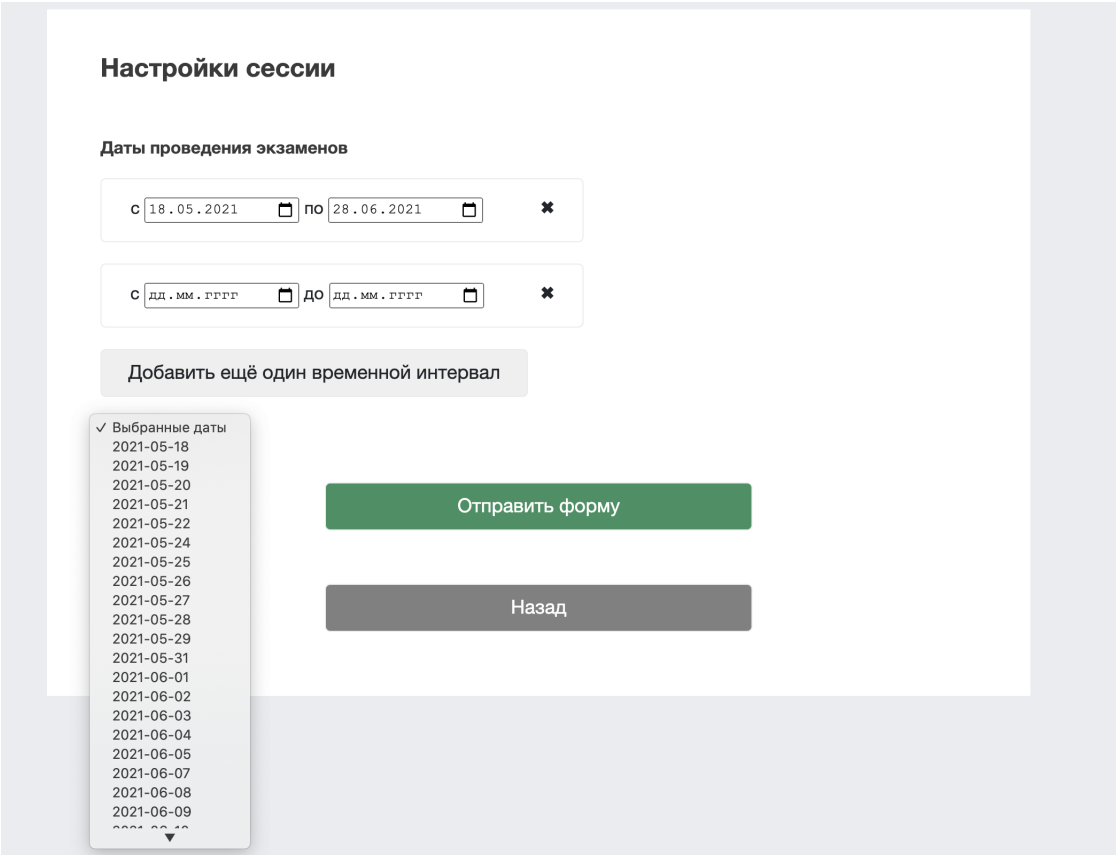

Рис.3.3. Форма выбора дней

POST-запрос к /session/update_class с параметром classroomFile позволяет добавить файл с данными об аудиториях, а запрос к /session/update_att с параметром attestationFile позволяет загрузить файл с данными о типах аттестаций.

POST-запрос к /session/update_time с параметрами time1 и time2 позволяет выбрать время проведения аттестаций. Форма для выбора времени изображена на скриншоте 3.4.

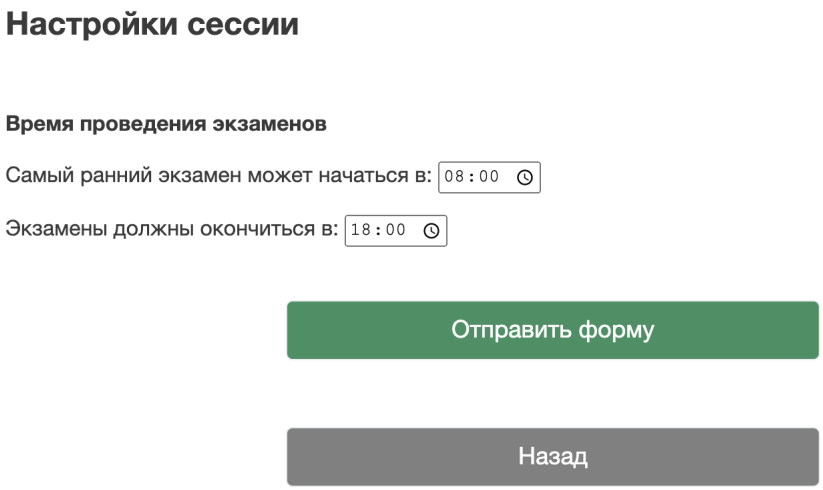

Рис.3.4. Форма выбора времени

POST-запросы к /session/update_exam_teacher и /session/update_exam с параметрами spec и examFile позволяет для конкретных специальностей загружать файлы со списком экзаменов. Форма для загрузки учебных планов представлена на скриншоте 3.5.

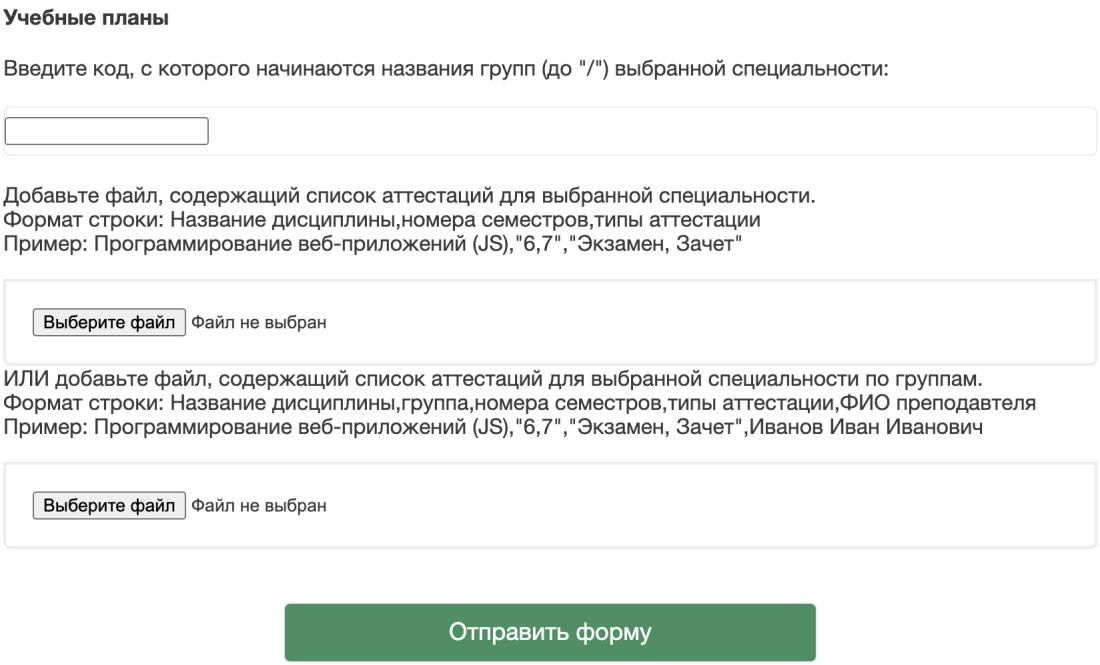

Рис.3.5. Форма загрузки учебных планов

POST-запрос к /session/update_external с параметром externalId позволяет добавить идентификатор google-таблицы, в которой содержатся аттестации с известными датами, временем и аудиториями, а запрос к /session/teachers/update с параметром spec даёт возможность обновить google-таблицу с данными для формирования google-формы сбора предпочтений ППС к расписанию сессии.

С помощью POST-запроса к /session/result/create с параметрами spec и levels для выбранных специальностей и курсов генерируется предварительное расписание. Страница с выбором специальностей и курсов для создания расписания представлена на рисунке 3.6.

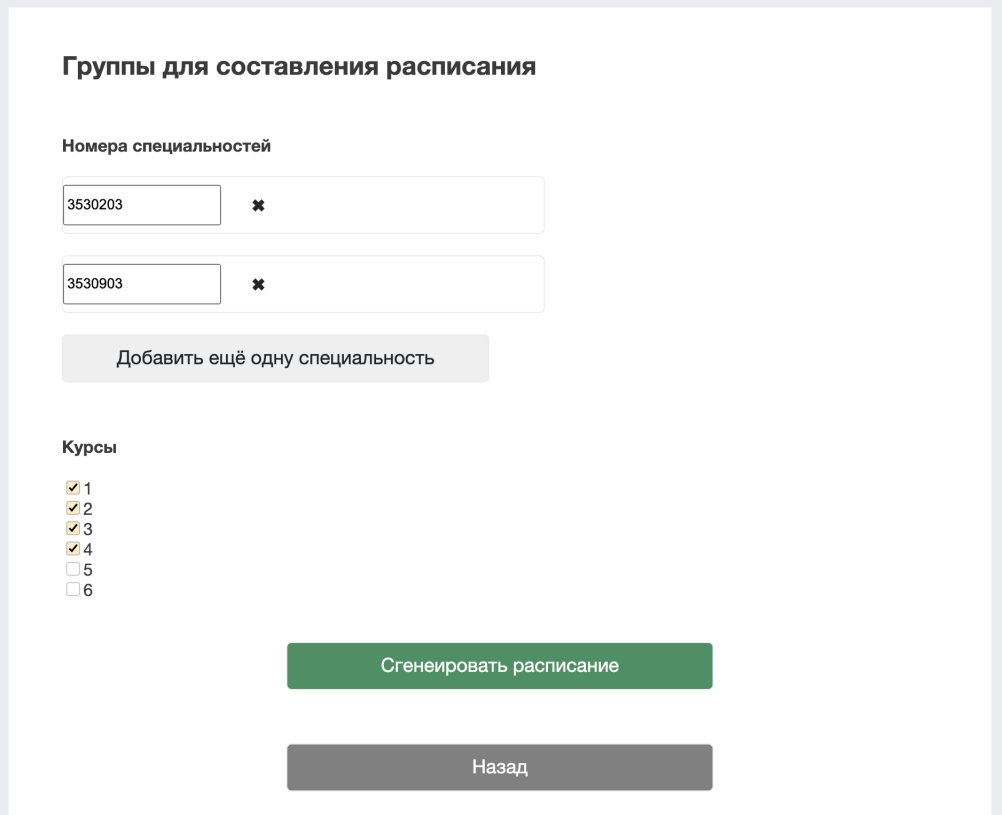

Рис.3.6. Форма загрузки учебных планов

3.3.2. Spring-сервер

Сервер имеет три основных класса-контроллера: SessionService 5, ApiRuz 3 и GoogleFormService 6. Они осуществляют бизнес-логику системы и обеспечивают связь между пользователями, данными и системой.

SessionService класс отвечает за загрузку данных о сессии от администратора. Он обрабатывает запросы, описанные в предыдущем параграфе и сохраняет данные в базе данных.

GoogleFormService класс отвечает за генерацию google-формы с предпочтениями преподавателей, загрузку данных из неё, а также из таблицы с известными аттестациями.

Чтобы создать форму для сбора предпочтений ППС, сначала нужно выгрузить данные о преподавательских аттестациях в google-таблицу. Администратор может самостоятельно подредактировать в ней данные, например, убрав аттестации, даты и время которых определяются дирекциями других институтов. Далее администратор может создать google-форму для сбора предпочтений. Форма генерируется автоматически с помощью AppScripts скрипта 10 по данным из таблицы.

Форма для сбора предпочтений ППС формирует для каждого преподавателя персональный опросник, в котором он определяет предпочтения только для своих аттетстаций. Фрагмент такой формы представден на рисунке 3.7.

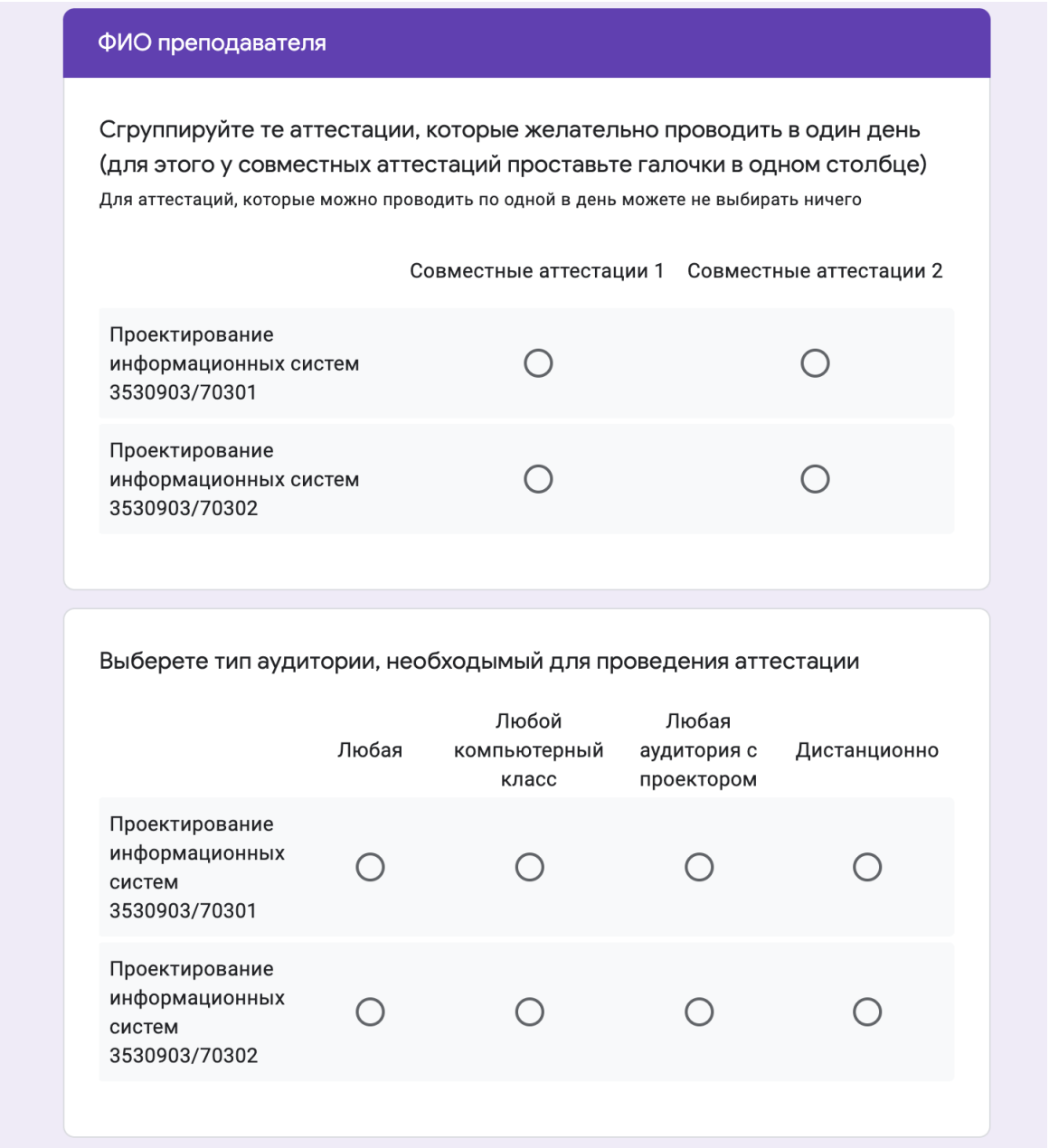

Рис.3.7. Фрагмент формы для сбора предпочтений ППС

Когда предпочтения преподавателей собраны, администратор может составить расписание. Для этого сервер при помощи Google API запрашивает данные из формы и на их основе составляет предварительное расписание. Полученное расписание сервер так же записывает в таблицу.

Класс ApiRuz отвечает за получение данных с официального сайта расписания СПбПУ для формирования списка аттестаций и учёта занятости аудиторий занятиями.

Метод getDateTimesByRoomAndDate(String room, LocalDate localDate) получает даты и время, когда указанная аудитория занята.

Метод initGroups(Iterable<String> spec) получает данные о группах выбранных специальностей - их номера и курс.

Метод initRasp() для полученных раннее групп находит преподавателей, которые читают им дисциплины.

Непосредственно алгоритм генерации предварительного расписания реализован в классе ScheduleAllSolutions и подробно описан в главе 2.

3.3.3. Хранилище данных

Исходя из данных, которые необходимо учитывать при составлении расписания сессии, была спроектирована модель данных, соответствующая схеме базы данных для их хранения. Она представлена на схеме 3.8.

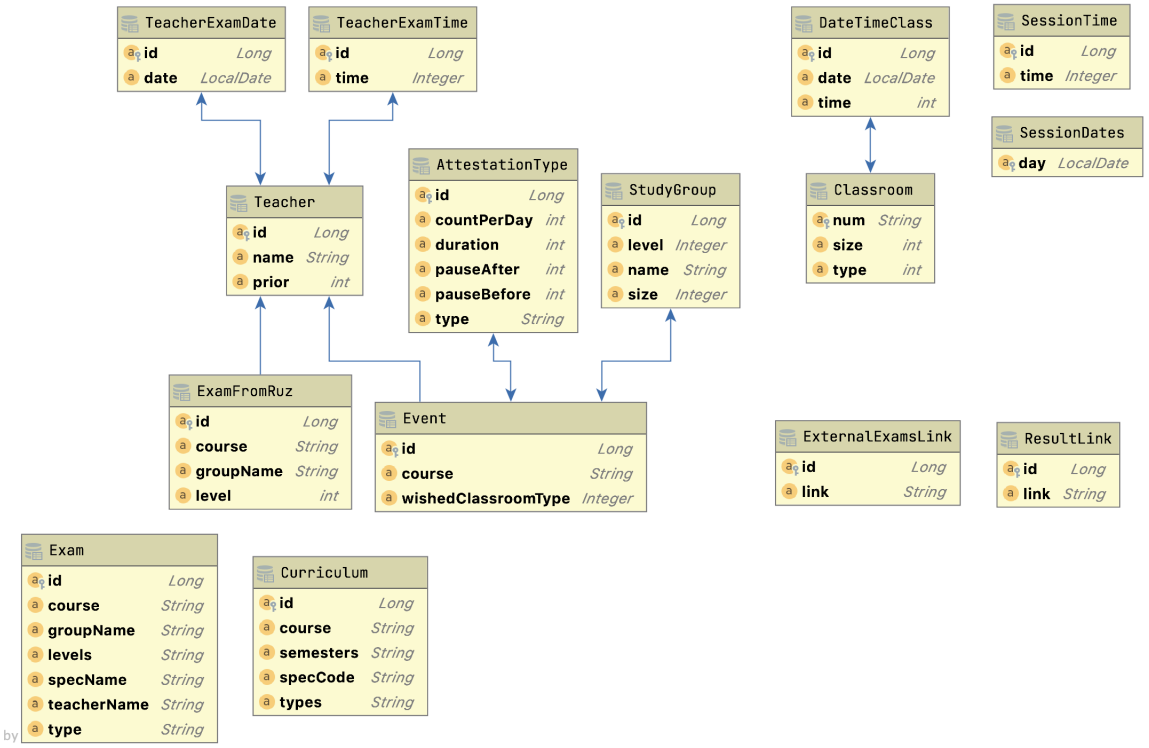

Рис.3.8. Схема базы данных

Сущности схемы:

- Teacher - таблица преподавателей.
  - id - уникальный идентификатор
  - name - имя
  - prior - приоритет
- TeacherExamDate - таблица дат, которые преподаватель указывает, как доступные для проведения аттестаций.
  - id - уникальный идентификатор

    - date - дата
    - teacherId - id преподавателя
- TeacherExamTime - таблица времени, которое преподаватель указывает, как доступное для проведения аттестаций.
    - id - уникальный идентификатор
    - time - время
    - teacherId - id преподавателя
- StudyGroup - таблица учебных групп.
    - id - уникальный идентификатор
    - level - курс
    - name - номер
    - size - кол-во студентов
- AttestationType - таблица типов аттестаций.
    - id - уникальный идентификатор
    - countPerDay - количество в день
    - duration - длительность в часах
    - pauseAfter -пауза в днях после аттестации
    - pauseBefore - пауза в днях до аттестации
    - type - название типа
- Classroom - таблица аудиторий с указанием есть ли там компьютеры или проектор и количеством мест.
    - id - уникальный идентификатор
    - classroom - номер кабинета
    - computers - наличие компьютеров
    - projector - наличие проектора
    - sitscount - кол-во мест в аудитории
- SessionDate - таблица дат, в которые проводится сессия. Такой способ хранения данных о датах сессии был выбран вместо хранения даты начала и окончания, потому что зимняя сессия может состоять из нескольких интервалов, прерванных например новогодними праздниками.
    - id - уникальный идентификатор
    - date - дата
- SessionTime - таблица часов для проведения аттестаций.
    - id - уникальный идентификатор
    - time - время

- ExternalExamLink - ссылки на google-таблицы, с аттестациями, для которых назначено время и место проведения. Они могут использоваться, например, для учёта аттестаций, которые назначаются другими подразделениями университета
    - id - уникальный идентификатор
    - link - ссылка
- Curriculum - таблица для хранения учебного плана.
    - id - уникальный идентификатор
    - cource - название дисциплины
    - semester - семестр, в который проводится аттестации
    - specCode - специальность
    - type - тип аттестации
- Exam - таблица аттестаций, данные о которых вводятся вместе с данными о преподавателях. Используется при формировании расписании сессии заочников, когда по данным из расписания ruz, нельзя определить, кто должен проводить аттестации.
    - id - уникальный идентификатор
    - course - название дисциплины
    - groupName - группа
    - levels - семестр
    - specName - специальность
    - teacherName - преподаватель
    - type - тип аттестации

Описанные выше сущности соотносятся с данными, которые необходимо получать от пользователей. В процессе работы алгоритма составления расписания в персистентное хранилище записываются также следующие сущности:

- ResultLink - ссылки на google-таблицы, со сгенерированными ранее вариантами расписаний
    - id - уникальный идентификатор
    - link - ссылка
- DateTimeClass - таблица доступных окон - дата+время+аудитория. В ней хранятся окна для проведения аттестаций, из которых исключены варианты, занятые другими занятиями.
    - id - уникальный идентификатор
    - date - дата

    - time - время
    - classroomId - id аудитории
- Event - таблица аттестаций, которые необходимо провести.
    - id - уникальный идентификатор
    - cource - название дисциплины
    - wishedClassroomType - необходимый тип аудитории
    - teacherId - id преподавателя
    - groupId - id учебной группы

Фреймворк Hibernate Spring-сервера позволяет сопоставлять java-объекты сущностям базы данных, поэтому сущности базы реализованы одноимёнными классами, помеченными аннотациями @Entity и @Data, на поля которых поставлены аннотации @Column, @ManyToOne и @OneToMany, обозначающие связи между колонками таблиц. Их код представлен в приложении 2.

Для осуществления запросов к базе созданы классы-наследники интерфейса CrudRepository. Через них функции класса SessionDao 7, помеченные аннотацией @Transactional, обращаются к данным.

3.4. Выводы

В результате была реализована система составления предварительного расписания сессии СПбПУ, включающая модуль веб-клиента для администратора, модуль взаимодействия с расписанием университета, модуль связи с google-таблицами для получения данных от преподавателей и модуль для генерации предварительного расписания.

Данная система может быть использована не только для составления расписания сессии, но и для составления расписания любых событий, занимающих непрерывное время в течение одного учебного дня. Это могут быть, например, установочные занятия для студентов заочной формы обучения, экзамены дополнительной сессии, консультации по практикам или ВКР. Такое расширение применимости системы возможно благодаря аналогичному принципу генерации расписания для неповторяющихся мероприятий, я формы сбора сведений о сессии и предпочтениях преподавательского состава достаточно гибкие для того, чтобы получать сведения о мероприятиях других типов.

ГЛАВА 4. ТЕСТИРОВАНИЕ И АПРОБАЦИЯ

В этой главе в параграфах 4.1, 4.2 и 4.3 речь пойдёт о проводимом тестировании системы составления предварительного расписания сессии СПбПУ, а в параграфе 4.4 о её апробации в период весенней сессии 20-21 учебного года.

4.1. Функциональное тестирование

Для проверки того, что разработанная система выполняет все требуемые функции, было проведено ручное функциональное тестирование. Для этого были созданы несколько искусственных вариантов входных файлов, содержащих данные о типах аттестаций, учебных планах направлений и список аудиторий. С их помощью было проверены следующие функциональности:

- при загрузке файла с аудиториями данные из него корректно сохраняются в базе данных и пользователю отображается новый список;
- при загрузке файла с учебным планом для конкретной специальности, в базе данных обновляется информация только по выбранной специальности;
- при загрузке файла с типами аттестаций информация из него сохраняется в базе данных, перетирая предыдущие данные о типах мероприятий, и отображается пользователю.

Также на тестовой google-таблице с данными о событиях с установленными датой, временем и аудиторией была проверена возможность учитывать эти данные, как константу при составлении остального расписания.

При выборе дат и времени в формах редактирования сессии, была проверена корректность обновления этих данных в базе данных и отображение их пользователю. Входные данные подбирались таким образом, чтобы было видно, что данные с сайта с расписанием учебных занятий [14] учитываются при генерации нового расписания.

Для проектов, разрабатываемых длительное время командой программистов, вместо ручного тестирования следовало бы использовать автоматизированное, но для данного проекта разработка самих автотестов заняла бы больше времени, чем ручное их прохождение. Проводимые тесты воспроизводили сценарии работы с системой составления предварительного расписания сессии, и показали, что с точки зрения пользователя она выполняет все заявленные функции.

4.2. Unit-тестирование

Для отладки и финальной проверки правильности работы отдельных методов программы использовалось модульное тестирование. Написание тестов позволяет на ранних этапах разработки отсекать ошибки, что повышает производительность и ускоряет процесс создания системы, так как корректность отдельных программных блоков проверяется сразу же, исключая ситуацию с переписыванием большого количества кода, вызванную чередой вовремя не замеченных багов.

Для написания модульных тестов использовался java-фреймворк jUnit. Был создан тестовый класс SchedulerTest, код которого находится в приложении 8. Данный класс содержит методы, помеченные аннотацией @Test, в которых выполнение некоторого условия проверяется с мощью методов класса Assert. Класс Assert фреймворка jUnit имеет множество методов, удобных для сверки ожидаемого результата проверяемой функции с реальным. Так, assertArrayEquals позволяет сравнивать содержимое массивов, assertNull - равенство null, assertEquals - любое равенство и так далее. При невыполнении условий такие тесты выбрасывают исключение, что даёт разработчику сигнал о том, что изменения кода внесли ошибки в старый код, и прежде чем приступать к следующим, нужно доработать эти. Таким образом, с помощью модульного тестирования была проверена корректность работы алгоритма составления предварительного расписания при разных наборах входных данных, что подтвердило его работоспособность и в случаях, когда расписание можно составить с учётом всех пожеланий, и когда приходится игнорировать некоторые пожелания, и когда расписание невозможно составить в принципе.

4.3. Тестирование производительности

Для того чтобы удостовериться, что выбранный алгоритм составления расписания наиболее оптимален, необходимо было провести ряд измерений, связанных со **скоростью работы** и **занимаемой памятью**. В сравнении с алгоритмом обхода в ширину, оптимизированного по времени применением мемоизации, выбранный алгоритм обхода в глубину с учётом приоритетов преподавателей заметно выигрывает по памяти. В таблице 4.1 приведено время и память, затраченное каждым из алгоритмов при составлении расписания на заданное количество мероприятий с определённым количеством свободных дней и аудиторий для их проведения. Для

более точных результатов было принято решение генерировать входные датасеты с аттестациями для составления расписания автоматически, чтобы при увеличении их числа сохранять одинаковым среднее количество событий на преподавателя и на группу, среднее свободное время на преподавателя и среднюю продолжительность мероприятий. Для этого генерировался небольшой датасет на несколько событий, который расширялся его копированием с переименованием всех дисциплин, преподавателей и групп. Код класса, в котором проводятся замеры времени и генерация входных данных, находится в приложении 12.

В случае с обходом в ширину объём входных данных для замеров времени и памяти был небольшим, так как для реализации этого алгоритма необходимо хранить объёмные массивы рассчитанных ранее частей расписания, из-за чего на более крупных входных данных вычисления прерывались ошибкой недостатка памяти. В таблице видна тенденция алгоритма 13 к быстрому росту используемой памяти с увеличением объёма входных датасетов, которая не свойственна выбранному алгоритму поиска в глубину.

Также, был проверен алгоритм обхода в глубину с использованием многопоточности. В таблице 4.2 приведено сравнение выбранного алгоритма с его многопоточной версией. Из неё видно, что время на выполнение составления расписания с использованием параллельных вычислений больше, чем без них, хотя предполагалось обратное. Такой эффект объясняется затратой времени на создание потоков, переключение между ними и использованием потокобезопасных блоков. Код класса с данным алгоритмом находится в приложении 11.

По результатам тестов как по памяти, так и по времени наиболее хорошо себя показал выбранный в итоге алгоритм обхода в глубину с учётом приоритетов преподавтелей. На втором месте по обоим показателям многопоточная версия этого алгоритма, а вот алгоритм обхода в ширину с большим отрывом отстаёт, что особенно заметно при составлении расписания для 8 мероприятий на 7 дней и 7 аудиторий, ведь ему потребовалось в 67 раз больше времени и в 18 раз больше памяти, чем победившему алгоритму.

Таблица 4.1

Сравнение алгоритма DFS и BFS

Кол-во окон	Алгоритм		Кол-во мероприятий			
			1	2	4	8
160 (4 дн., 4 ауд., 10 ч.)	DFS	время (мс)	49	54	58	57
		память (байт)	11 177 746	11 743 432	11 743 432	11 743 432
	BFS	время (мс)	38	87	182	386
		память (байт)	11 115 598	11 117 289	11 125 344	12 116 086
250 (5 дн., 5 ауд., 10 ч.)	DFS	время (мс)	46	55	56	60
		память (байт)	11 312 944	12 104 340	12 104 624	12 104 624
	BFS	время (мс)	38	82	173	506
		память (байт)	11 116 673	11 119 104	11 144 900	20 581 732
360 (6 дн., 6 ауд., 10 ч.)	DFS	время (мс)	45	57	61	63
		память (байт)	11 558 148	12 555 800	12 546 126	12 546 568
	BFS	время (мс)	35	80	178	1 188
		память (байт)	11 117 368	11 120 984	11 181 744	64 185 568
490 (7 дн., 7 ауд., 10 ч.)	DFS	время (мс)	49	62	63	64
		память (байт)	11 960 845	13 067 872	13 067 872	13 067 872
	BFS	время (мс)	40	83	181	4 307
		память (байт)	11 118 024	11 123 056	11 248 352	238 954 399

Таблица 4.2

Сравнение алгоритма DFS и DFS с применением многопоточности

Кол-во окон	Алго-ритм		Кол-во мероприятий			
			15	30	60	120
2000 (20 дн., 10 ауд., 10 ч.)	DFS	время (мс)	142	171	257	731
		память (байт)	20 859 008	20 859 085	20 859 288	20 859 584
	Parallel	время (мс)	161	200	298	875
		память (байт)	112 779 515	112 779 624	112 779 624	112 779 624
4500 (30 дн., 15 ауд., 10 ч.)	DFS	время (мс)	239	272	416	1316
		память (байт)	30 879 064	30 879 136	30 847 120	30 833 680
	Parallel	время (мс)	352	393	583	1543
		память (байт)	112 779 624	112 779 694	112 779 800	112 779 800
12500 (50 дн., 25 ауд., 10 ч.)	DFS	время (мс)	873	579	935	2942
		память (байт)	62 897 160	62 897 232	62 897 384	62 897 680
	Parallel	время (мс)	895	965	1331	3500
		память (байт)	112 779 800	112 779 800	112 779 800	112 779 908
25000 (50 дн., 50 ауд., 10 ч.)	DFS	время (мс)	1030	1132	2181	8763
		память (байт)	112 997 160	112 997 232	112 997 416	112 997 744
	Parallel	время (мс)	1770	1861	2952	9943
		память (байт)	112 779 936	112 779 936	112 780 016	112 780 136

4.4. Апробация

Апробация системы составления предварительного расписания проводилась на весенней сессии 2020-2021 учебного года.

Первый этап апробации заключался в составлении расписания экзаменов для студентов 4-ых курсов направлений 02.03.03 и 09.03.03. Форма для сбора пожеланий преподавателей была выполнена с помощью сервиса Microsoft Forms, так как это позволяло автоматически определять адреса электронных почт и имена респондентов. На этом этапе были выявлены несколько её недостатков:

- форма одинакова для всех преподавателей, и для того чтобы указать необходимые типы аудиторий преподавателю приходилось выбирать из огромного списка только те группы и дисциплины, у которых он проводит аттестации;
- имена ассистентов, введённые в поле респондентами не всегда соответствовали именам преподавателей с сайта расписания учебных занятий, что затруднило их маппинг;
- одному респонденту приходилось отправлять форму несколько раз для разных аттестаций;
- Microsoft Forms не имеет API для чтения собранных результатов;
- ручное создание формы заняло много времени.

Для решения этих проблем было принято решение делать формы с помощью сервиса Google Forms. Его преимуществом перед Microsoft является наличие API, позволяющего полностью автоматизировать как чтение результатов, так и создание самой формы. Теперь каждый преподаватель мог выбрать своё имя, а далее заполнять форму только по своим аттестациям. Также, в новой версии формы был минимизирован ручной ввод информации респондентами. Это решило проблему опечаток и различий введённых имён от имён из расписания. Новая версия формы не требовала повторной отправки, так как все необходимые данные преподаватель может ввести за одну итерацию заполнения формы.

Новый вариант google-формы был опробован при составлении предварительного расписания летней сессии студентов заочной формы обучения специальности Прикладная информатика. а также расписания их установочных занятий. Таким образом, были получены две таблицы с расписаниями, сгенерированными системой, что продемонстрировало принципиальную применимость разработанной системы

к составлению расписания любого мероприятия, занимающего непрерывное время в течение одного учебного дня.

В качестве входных данных системе подавался список возможных типов аттестаций, включающий их длительность, максимальное кол-во в день и сведения о количестве дней отдыха до и после. Список представлен в таблице 4.3.

Таблица 4.3

Типы аттестаций

Тип аттестации	Отдых до	Отдых после	Длительность	Макс. кол-во в день
Эк	0	0	5	1
Зч	0	0	3	2
ЗчО	0	0	3	2
Лек2	0	0	2	10
Лаб2	0	0	2	10
Пр2	0	0	2	10
Лек4	0	0	4	10
Лаб4	0	0	4	10
Пр4	0	0	4	10
Кр	0	0	3	10
Кпр	0	0	4	10

Для определения того, какие события необходимо учесть в расписании, на вход была подала таблица с учебным планом на летнюю сессию 2021, в которой перечислены учебные группы, дисциплины, типы аттестации и преподаватели, проводящие для них аттестации. Система также может принимать на вход сведения об учебных планах без указания преподавателей. В таком случае сопоставление события и его проводящего происходило бы с помощью данных из расписания «ruz.spbstu.ru». Но в данном случае преподаватели были известны, и загруженные данные представлены в таблице 4.4.

Таблица 4.4

План экзаменов летней сессии студентов заочной формы обучения

Дисциплина	Курс	Группа	Тип	ППС
Дифференциальные уравнения	1	з3530903/00001	Эк	ФИО1
Дифференциальные уравнения	1	з3530903/00002	Эк	ФИО1
Дополнительные главы программирования (C#)	1	з3530903/00001	Эк	ФИО2
Дополнительные главы программирования (C#)	1	з3530903/00002	Эк	ФИО2
Базы данных	2	з3530903/90001	Эк	ФИО3
Базы данных	2	з3530903/90002	Эк	ФИО3
Структуры и алгоритмы обработки данных	2	з3530903/90001	Эк	ФИО6
Структуры и алгоритмы обработки данных	2	з3530903/90002	Эк	ФИО6
Дискретная математика (дополнительные главы)	2	з3530903/90002	ЗчО	ФИО6
Дискретная математика (дополнительные главы)	2	з3530903/90001	ЗчО	ФИО6
Учебная практика/Ознакомительная практика	2	з3530903/90001	Зч	ФИО7
Учебная практика/Ознакомительная практика	2	з3530903/90002	Зч	ФИО7
Проектирование информационных систем	4	з3530903/70301	Эк	ФИО7
Проектная деятельность бакалавра	4	з3530903/70301	Зч	ФИО8
Использование и разработка web-сервисов	4	з3530903/70301	Эк	ФИО9
Использование и разработка web-сервисов	4	з3530903/70301	Зч	ФИО9
Интеллектуальный анализ данных	4	з3530903/70301	Эк	ФИО10
Интеллектуальный анализ данных	4	з3530903/70301	Зч	ФИО10
Производственная практика/Научно-исследовательская работа	4	з3530903/70301	ЗчО	ФИО2
Администрирование информационных систем (на английском языке)	3	з3530903/80301	Эк	ФИО11
Теория вероятностей и математическая статистика	3	з3530903/80301	Эк	ФИО1
Архитектура вычислительных систем	3	з3530903/80301	Эк	ФИО12
Технологии разработки программного обеспечения (C#)	3	з3530903/80301	Эк	ФИО7
Программирование веб-приложений (JS)	3	з3530903/80301	Зч	ФИО13
Производственная практика/Технологическая (проектно-технологическая) практика	3	з3530903/80301	Зч	ФИО5

Далее профессорско-преподавательский состав, участвующий в летней сессии для заочников заполнял google-формы, в которых называли предпочтительный для себя интервал дат для проведения сессии, интервал дат своего отсутствия, удобные часы для проведения занятий и неподходящие дни недели. Также, им предлагалось ввести ссылки, по которым планируется проводить дистанционные аттестации. В данном случае эти ссылки - аналог аудиторий. Данные формы сохраняются в связанной с ней таблице, которая имела вид таблицы 4.5.

Таблица 4.5

Пожелания преподавателей к проведению сессии

Совм.	Дата с	Дата до	Имя	Ассист.	Отсутсвие с	Отсутствие до	Дни недели	час 1	час 2
Нет			ФИО1	ФИО13	15.06.2021	18.06.2021		9	18
Нет			ФИО7		08.06.2021	18.06.2021	Суббота	10	20
Да	14.06.2021	19.06.2021	ФИО10					10	18
Да	10.06.2021	30.06.2021	ФИО13		17.06.2021	18.06.2021	Пятница, Суббота	9	20
Нет			ФИО12					10	20
Да	23.06.2021	07.07.2021	ФИО11				Пятница, Суббота	10	17
Нет			ФИО6				Суббота	10	20
Нет			ФИО9					11	17
Да	07.06.2021	28.06.2021	ФИО3				Четверг, Суббота	10	20

Далее были выбраны даты проведения сессии, и сгенерировано расписание, выгруженное в google-таблицу. Оно представлено в таблице 4.6. Данное расписание учитывает пожелания преподавателей, ограничения, связанные с типами аттестаций и временными рамками.

Таблица 4.6

Сгенерированное расписание сессии

Группа	ППС	Дисциплина	Тип	Дата	Час	Teams
з3530903/70301	[ФИО8]	Проектная деятельность бакалавра	Зч	2021-06-17	12	...
з3530903/70301	[ФИО10]	Интеллектуальный анализ данных	Зч	2021-06-17	10	...
з3530903/70301	[ФИО10]	Интеллектуальный анализ данных	Эк	2021-06-18	10	...
з3530903/00001	[ФИО2]	Дополнительные главы программирования (C#)	Эк	2021-06-21	11	...
з3530903/90001	[ФИО6]	Дискретная математика (дополнительные главы)	Зч	2021-06-21	10	...
з3530903/90002	[ФИО6]	Дискретная математика (дополнительные главы)	Зч	2021-06-21	12	...
з3530903/80301	[ФИО13]	Программирование веб-приложений (JS)	Зч	2021-06-21	9	...
з3530903/70301	[ФИО9]	Использование и разработка web-сервисов	Эк	2021-06-21	11	...
з3530903/90002	[ФИО7]	Учебная практика/Ознакомительная практика	Зч	2021-06-21	12	...
з3530903/90001	[ФИО7]	Учебная практика/Ознакомительная практика	Зч	2021-06-21	14	...
з3530903/00002	[ФИО2]	Дополнительные главы программирования (C#)	Эк	2021-06-22	8	...
з3530903/80301	[ФИО12]	Архитектура вычислительных систем	Эк	2021-06-22	10	...
з3530903/70301	[ФИО9]	Использование и разработка web-сервисов	Зч	2021-06-23	8	...
з3530903/80301	[ФИО11]	Администрирование информационных систем (на английском языке)	Эк	2021-06-23	10	...
з3530903/70301	[ФИО2]	Производственная практика/Научно–исследовательская работа	Зч	2021-06-24	8	...
з3530903/80301	[ФИО7]	Технологии разработки программного обеспечения (C#)	Эк	2021-06-24	10	...
з3530903/90001	[ФИО3]	Базы данных	Эк	2021-06-25	8	...
з3530903/90002	[ФИО3]	Базы данных	Эк	2021-06-25	13	...
з3530903/00001	[ФИО13, ФИО1]	Дифференциальные уравнения	Эк	2021-06-25	8	...
з3530903/00002	[ФИО13, ФИО1]	Дифференциальные уравнения	Эк	2021-06-25	13	...
з3530903/80301	[ФИО13, ФИО1]	Теория вероятностей и математическая статистика	Эк	2021-06-25	12	...
з3530903/70301	[ФИО7]	Проектирование информационных систем	Эк	2021-06-25	10	...
з3530903/90002	[ФИО6]	Структуры и алгоритмы обработки данных	Эк	2021-06-28	10	...
з3530903/90001	[ФИО6]	Структуры и алгоритмы обработки данных	Эк	2021-06-28	12	...

4.5. Выводы

Система составления предварительного расписания сессии была успешно применена в СПбПУ при составлении расписания сессии и установочных занятий студентов заочной формы обучения, чем доказала, что может быть использована для составления расписаний любых разовых мероприятий, занимающих непрерывное время в течение дня. Это делает её очень полезной для составления расписания дополнительной сессии, так как ей занимается напрямую кафедра без участия дирекции института, а значит, в этом случае предварительное расписание можно рассматривать как основанное.

Как и было заявлено в требованиях, система автоматизирует сбор сведений от профессорско-преподавательского состава, а так же сама получает и учитывает данные из официального расписания СПбПУ. Ручное редактирование расписания доступно в нескольких форматах - финальные ручные правки администратором в сгенерированной google-таблице, либо внесение установленных аттестаций в таблицу учёта, и генерация нового расписания на её основе.

Из потенциальных улучшений, которые в будущем можно добавить в систему, можно назвать:

- создание более надёжного модуля аутентификации;
- рассылка преподавателям напоминаний о том, что они не заполнили форму сбора сведений;
- открытие и рассылка готового расписания преподавателям;
- создание модуля навигации между сгенерированными таблицами и форма сбора сведений.

Добавление этих и других модулей не будет вынуждать разработчика переписывать много существующего кода, так как всё это отдельные блоки, которые можно легко встраивать в текущую систему.

ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы мною были сравнены существующие российские и зарубежные системы составления расписания, изучены алгоритмы составления и оптимизации расписания и подходы к этому процессу, спроектирована и реализована система составления предварительного расписания сессии СПбПУ.

Созданный программный продукт автоматизирует сбор требований преподавателей к проведению аттестаций, получает данные о проведении занятий и занятости аудиторий с официального сайта расписания СПбПУ и генерирует предварительное расписание с учётом физических и нормативных ограничений, а также с учётом полученных пожеланий профессорско-преподавательского состава. Помимо прочего, он позволяет редактировать расписание с использованием google-таблиц и учитывать занятия с уже установленным временем и аудиторией, что даёт возможность экспериментировать с результатами.

Система была неоднократно протестирована и на этапе апробирования показала, что может быть использована не только для составления расписания аттестаций, но и для других не повторяющихся из недели в неделю событий, как например консультации ВКР, установочные занятия студентов заочной формы обучения или экзамены дополнительной сессии, что демонстрирует её широкий спектр применений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. 1С: ХроноГраф Расписание. — URL: https://1c.ru/rus/products/1c/predpr/compat/catalog/solution.jsp?SolutionID=15710 (дата обращения: 11.03.2021).

2. Apereo UniTime. — URL: https://www.unitime.org/ (дата обращения: 11.03.2021).

3. *Doig L. A. H.* Совершенный алгоритм. Жадные алгоритмы и динамическое программирование. — 2020.

4. Dropwizard Docs. — URL: https://www.dropwizard.io/en/latest/getting-started.html (дата обращения: 17.03.2021).

5. Hibernate Docs. — URL: https://hibernate.org/orm/documentation/5.4/ (дата обращения: 02.04.2021).

6. *Knuth D. E.* The Art of Computer Programming Vol 1. 3rd ed. — 1997. — URL: https://www-cs-faculty.stanford.edu/~knuth/taocp.html (дата обращения: 17.02.2021).

7. Lantiv Scheduling Studio. — URL: https://scheduling-studio.lantiv.com/ (дата обращения: 11.03.2021).

8. Spring Framework documentation. — URL: https://spring.io/projects/spring-framework (дата обращения: 17.03.2021).

9. *Strobl M.A.R.; Barker D.* On simulated annealing phase transitions in phylogeny reconstruction. — 2016. — URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4912009/ (дата обращения: 19.02.2021).

10. *T. P.* An automatic method of solving discrete programming problems. — 1960.

11. Vert.x Docs. — URL: https://vertx.io/docs/ (дата обращения: 17.03.2021).

12. АВТОРасписание. — URL: http://avtor.bravosoft.org/ (дата обращения: 11.03.2021).

13. Галактика Расписание учебных занятий. — URL: https://galaktika-it.ru (дата обращения: 11.03.2021).

14. Расписание СПбПУ. — URL: http://ruz.spbstu.ru (дата обращения: 11.03.2021).

15. *Abraham A. Jain L. G. R.* Evolutionary Multiobjective Optimization. Theoretical Advances and Applications. — 2005.

16. *Freedman D. Pisani R. P. R.* Statistics: Fourth International Student Edition. — 2007.

Приложение 1

Поиск расписаний с учётом приоритетов преподавателей

```
ipackage ru.technopolis.scheduler;

import ru.technopolis.model.*;

import java.time.LocalDate;
import java.util.*;

public class SchedulePriorAlgorithm {
  public List<int[]> solutions;
  List<Event> events;
  List<Solution> external;
  List<DateTimeClass> dtc;
  Map<Event, DateTimeClass> externalMap = new HashMap<>();
  List<DateTimeClass> externalDtc = new ArrayList<>();
  int[] solution;
  int eventSize;
  int dtcSize;
  int maxTimePerDay;
  int solutionsMaxSize;
  boolean[] ignoreWishes; //приходится ли игнорировать пожелан
     ия преподавателя для этого события

  public SchedulePriorAlgorithm(List<Event> events, List<
     DateTimeClass> dtc, List<Solution> externalExams, int
     maxTimePerDay, int solutionsMaxSize, List<Solution>
     externalSolutions) {
    this.events = new ArrayList<>();
    this.dtc = new ArrayList<>();

    this.events.addAll(events);
    Collections.sort(this.events);
    Set<DateTimeClass> dtcSet = new HashSet<>(dtc);
    //сначала те события, для которых определено время
    if (externalExams != null && externalExams.size() > 0) {
      this.external = externalExams;
      for (Solution sol : this.external) {
        Event e = sol.event;
        System.out.println(e);
        externalMap.put(e, sol.dtc);
        for (int i = 0; i < e.type.duration; i++) {
```

```
          DateTimeClass dtc2 = new DateTimeClass(sol.dtc.date,
              sol.dtc.time + i, sol.dtc.classroom);
          externalDtc.add(dtc2);
        }
        int i = this.events.size() - 1;
        while (i >= 0) {
          Event ev = this.events.get(i);
          if (e.course.equals(ev.course) && e.group.name.
             equals(ev.group.name)) {
            this.events.remove(ev);
            break;
          }
          i--;
        }
      }
      dtcSet.addAll(externalDtc);
    }
    Collections.sort(this.events);
    this.events.addAll(0, externalMap.keySet());
    this.dtc.addAll(dtcSet);
    Collections.sort(this.dtc);
    this.eventSize = this.events.size();
    this.dtcSize = this.dtc.size();
    this.maxTimePerDay = maxTimePerDay;
    this.external = externalSolutions;
    this.solutionsMaxSize = solutionsMaxSize;
    this.solution = new int[dtcSize];
    for (int i = 0; i < dtcSize; i++) {
      solution[i] = -1;
    }
    this.solutions = new ArrayList<>();
    for (Event event : externalMap.keySet()) {
      for (int i = 0; i < dtcSize; i++) {
        if (this.dtc.get(i).equals(this.externalMap.get(event)
           )) {
          this.submitDateTimeClass(this.events.indexOf(event),
              i);
          break;
        }
      }
    }

    this.ignoreWishes = new boolean[eventSize];
  }
```

```
public SchedulePriorAlgorithm findSolutions() {
  System.out.println("START FIND SOL");
  int externalSize = externalMap.keySet().size();
  int event = externalSize;
  if (externalSize == eventSize) {
    solutions.add(solution);
    return this;
  }
  while (event < eventSize) {
    //            if (event < externalSize) {
      //                event++;
      //            } else {
      int time = this.getTime(event);
      this.cleanEvent(event);
      int nextTime = this.findNextStartTime(event, time);

      //переходим в режим игнорирования пожеланий преподавателя, если не смогли найти идеального решения
      if (!ignoreWishes[event] && nextTime == -1) {
        ignoreWishes[event] = true;
        nextTime = this.findNextStartTime(event, 0);
      }

      if (nextTime == -1) { //если не удалось найти другого подходящего DateTimeClass для этого события
        if (event == externalSize) {
          System.out.println("END FIND SOL :(");
          return this; //если речь о первом событии, то уже были перебраны все остальные варинты даже без учёта пожеланий преподавателя
        } else {
          ignoreWishes[event] = false;
          event--; // иначе возвращаемся к предыдущему событию и пробуем изменить для него DateTimeClass
        }
      } else {
        this.submitDateTimeClass(event, nextTime); //бронируем за событием время и аудиторию
        event++; // переходим к следующему событию
      }
      // если решение найдено, добавляем его в список решений и продолжаем искать решения
      if (event == eventSize) {
        solutions.add(solution.clone());
```

```
            if (solutions.size() >= solutionsMaxSize) return
                this;
            event--;
          }
          // }
      }
      System.out.println("END FIND SOL :)");
      return this;
    }

    boolean findSolution() {
      int event = 0;
      while (event < eventSize) {
        int time = this.getTime(event);
        int nextTime;
        this.cleanEvent(event);
        nextTime = this.findNextStartTime(event, time);

        //переходим в режим игнорирования пожеланий преподавател
            я, если не смогли найти идеального решения
        if (!ignoreWishes[event] && nextTime == -1) {
          ignoreWishes[event] = true;
          nextTime = this.findNextStartTime(event, 0);
        }

        if (nextTime == -1) { //если не удалось найти другого по
            дходящего DateTimeClass для этого события
          if (event == 0) {
            return false; //если речь о первом событии, то уже б
                ыли перебраны все остальные варинты, и решения не
                т
          } else {
            ignoreWishes[event] = false;
            event--; // иначе возвращаемся к предыдущему событию
                и пробуем изменить для него DateTimeClass
          }
        } else {
          this.submitDateTimeClass(event, nextTime); //бронируем
              за событием время и аудиторию
          event++; // переходим к следующему событию
        }
      }
      return true;
    }
```

```
  public List<List<Solution>> getSolutions() {
155   System.out.println("GET SOL");
    if (solutions.size() == 0) return new ArrayList<>();
    List<List<Solution>> res = new ArrayList<>();
    for (int[] ints : solutions) {
      List<Solution> sh = new ArrayList<>();
160   for (int j = 0; j < ints.length; j++) {
        if (ints[j] != -1) {
          sh.add(new Solution(events.get(ints[j]), dtc.get(j))
             );
        }
      }
165   res.add(sh);
    }
    return res;
  }

170 private int findNextStartTime(int event, int time) {
    if (time < 0) return -1;
    Event ev = events.get(event);
    StudyGroup group = ev.group;
    Set<Teacher> teachers = ev.teacher;
175 int duration = ev.type.duration - 1;
    LocalDate dateGroupEvent = null;
    Integer timeGroupEvent = null;
    if (ev.eventGroup != -1 && event != 0) {
      for (int i = 0; i < solution.length; i++) {
180     if (solution[i] != -1 && solution[i] != event &&
           events.get(solution[i]).eventGroup.equals(ev.
           eventGroup)) {
          dateGroupEvent = dtc.get(i).date;
          if (ev.teacher.size() > 1) {
            timeGroupEvent = dtc.get(i).time;
          }
185       break;
        }
      }
    }
    for (int i = time; i < dtcSize; i++) {
190   DateTimeClass location = dtc.get(i);
      // если для события установлено, что оно должно проводит
         ься в один день с другим
      if ((dateGroupEvent != null && !dateGroupEvent.equals(
         location.date)) ||
```

```
        (timeGroupEvent != null && !timeGroupEvent.equals(
           location.time))) {
          continue;
195     }
        boolean teacherWishes = teachers.stream().mapToInt(
           teacher -> {
          if (
          teacher.date.stream().anyMatch(dt -> dt.equals(
             location.date))
          && teacher.time.stream().anyMatch(t -> t.equals(
             location.time))
200       && teacher.time.stream().anyMatch(t -> t.equals(
             location.time + duration))) return 0;
          else return 1;
        }).sum() == 0;
        // в режиме игнорирования учитываем только события НЕ уд
           овлетворяющие пожелания преподавателя
        if (ignoreWishes[event]) {
205       teacherWishes = !teacherWishes;
        }

        boolean classRoom;
        if (ev.link != null && !ev.link.equals("")) {
210       classRoom = location.classroom.num.equals(ev.link);
        } else if (ev.wishedClassroomNums != null && !ev.
           wishedClassroomNums.equals("")) {
          classRoom = ev.wishedClassroomNums.contains(location.
             classroom.num);
        } else {
          classRoom = ev.wishedClassroomType <= location.
             classroom.type;
215     }

        if (classRoom //есть ли в аудитории нужное оборудование
        && teacherWishes // подходит ли дата и время преподавате
           лю
        && solution[i] == -1 // аудитория в это время свободна
220     && group.size <= location.classroom.size //влезает ли гр
           уппа в аудиторию
        // не слишком ли сейчас поздно для начала проведения дли
           тельного события
        && i + duration < dtcSize
        && location.classroom.num.equals(dtc.get(i + duration).
           classroom.num)
```

```
        && location.time + duration == dtc.get(i + duration).
           time // в массиве времени нет разрывов
225     && location.date.equals(dtc.get(i + duration).date)
        ) {
          Map<String, Integer> teacherTime = new HashMap<>();
          int countPerDay = 0;
          int k = 0;
230       while (k < dtcSize) {
            // если время-место свободны, они не повлияют на огр
               аничения
            if (solution[k] == -1) {
              k++;
              continue;
235         }
            Event currentEvent = events.get(solution[k]);
            DateTimeClass currentLocation = dtc.get(k);
            // если проверяемая аудитория в это время занята
            if (location.date.equals(currentLocation.date) &&
               currentLocation.time >= location.time &&
               currentLocation.time < location.time + ev.type.
               duration && location.classroom.num.equals(
               currentLocation.classroom.num)) {
240           break;
            }
            boolean success = false;
            // если у преподавателя в этот день уже были занятия
            if (currentLocation.date.equals(location.date)
245         && !(dateGroupEvent != null && ev.eventGroup.equals(
               currentEvent.eventGroup) && ev.teacher.size() >
               1)
            ) {
              for (Teacher curT : currentEvent.teacher) {
                for (Teacher teacher : teachers) {
                  if (curT.name.equals(teacher.name)) {
250                 if (!teacherTime.containsKey(teacher.name))
                       {
                      teacherTime.put(teacher.name, 1);
                    } else {
                      teacherTime.compute(teacher.name, (key, v)
                          -> v + 1);
                    }
255                 // если преподаватель в это время ведёт друг
                       ую аттестацию
                    if (currentLocation.time >= location.time
```

```
                    && currentLocation.time <= location.time +
                       duration) {
                      success = true;
                      break;
                    }
                    // если преподаватель уже достаточно отработ
                       ал в этот день.
                    if (teacherTime.get(teacher.name) >
                       maxTimePerDay - duration) {
                      success = true;
                      break;
                    }
                  }
                }
                if (success) break;
              }
              if (success) {
                break;
              }
            }
            if (currentEvent.group.name.equals(group.name)) {
              //если в этот день уже были занятия у этой группы
              if (currentEvent.type.type.equals(ev.type.type) &&
                  location.date.equals(currentLocation.date)) {
                countPerDay++;
                if (ev.type.countPerDay < countPerDay) {
                  break;
                }
              }
              //если до или после события есть другое событие, д
                 о которого меньше "отдыха", чем нужно
              if (location.date.minusDays(Math.max(ev.type.
                 pauseBefore, currentEvent.type.pauseAfter)).
                 compareTo(currentLocation.date) <= 0
              && location.date.plusDays(Math.max(ev.type.
                 pauseAfter, currentEvent.type.pauseBefore)).
                 compareTo(currentLocation.date) >= 0) {
                break;
              }
            }
            k++;
          }
          if (k == dtcSize) {
            return i;
          }
```

```
      }
    }
    return -1;
  }

  private void cleanEvent(int event) {
    for (int i = 0; i < dtcSize; i++) {
      if (solution[i] == event) solution[i] = -1;
    }
  }

  // бронинирование за событием помещения и времени
  private void submitDateTimeClass(int event, int nextTime) {
    int duration = events.get(event).type.duration;
    for (int i = nextTime; i < nextTime + duration; i++) {
      solution[i] = event;
    }
  }

  // возвращает следующий свободный индекс DateTimeClass для у
     казанного события
  private int getTime(int event) {
    int min = -1;
    boolean contains = false;
    for (int i = 0; i < dtcSize - 1; i++) {
      // если аудитория в это время ещё никем не забронирована
      if (solution[i] == -1) {
        if (contains) {
          return i;
        } else if (min == -1) {
          min = i;
        }
      }
      // если это событие уже бронировало какое-то время и мес
         то
      if (solution[i] == event) {
        if (contains) return i;
        contains = true;
      }
    }
    // если последняя ячейка была занята текущим событием, то
       следующей для него нет
    if (contains) return -1;
    return min;
  }
```

```
335 }
```

Приложение 2

Классы для хрранения входных данных

```
@Data
@Accessors(chain = true)
@Entity
@Audited
public class AttestationType {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  public Long id;
  @Column(nullable = false)
  public String type;
  @Column(nullable = false)
  public int pauseBefore;
  @Column(nullable = false)
  public int pauseAfter;
  @Column(nullable = false)
  public int duration;
  @Column(nullable = false)
  public int countPerDay;

  @OneToMany(mappedBy = "type", fetch = FetchType.LAZY,
     cascade = CascadeType.ALL)
  @JsonManagedReference
  private List<Event> event;

  public AttestationType(String type, int pauseBefore, int
     pauseAfter, int duration, int countPerDay) {
    this.type = type;
    this.pauseBefore = pauseBefore;
    this.pauseAfter = pauseAfter;
    this.duration = duration;
    this.countPerDay = countPerDay;
  }

  public AttestationType(){
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
```

```
    if (o == null || getClass() != o.getClass()) return false;
    AttestationType that = (AttestationType) o;
    return Objects.equals(type, that.type);
  }

  @Override
  public int hashCode() {
    return Objects.hash(type);
  }

  @Override
  public String toString() {
    return "AttestationType{" +
      "id="+ id +
      ", type='" + type + '\'' +
      ", pauseBefore=" + pauseBefore +
      ", pauseAfter=" + pauseAfter +
      ", duration=" + duration +
      ", countPerDay=" + countPerDay +
      '}';
  }
}

@Data
@Accessors(chain = true)
@Entity
@Audited
public class Curriculum {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  public Long id;
  @Column(nullable = false)
  public String specCode;
  @Column(nullable = false)
  public String course;
  @Column(nullable = false)
  public String semesters;
  @Column(nullable = false)
  public String types;

  public Curriculum(String specCode, String course, String
     semesters, String types) {
    this.specCode = specCode;
    this.course = course;
    this.semesters = semesters;
```

```
    this.types = types;
  }

  public Curriculum() {
  }

  @Override
  public String toString() {
    return "Curriculum{" +
      "specCode='" + specCode + '\'' +
      ", cource='" + course + '\'' +
      ", semesters='" + semesters + '\'' +
      ", types='" + types + '\'' +
      '}';
  }
}

@Data
@Accessors(chain = true)
@Entity
@Audited
public class DateTimeClass implements Comparable<DateTimeClass
   > {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  public Long id;
  @Column(nullable = false)
  public LocalDate date;

  @Column(nullable = false)
  public int time;

  @ManyToOne(fetch = FetchType.EAGER, cascade = {CascadeType.
     PERSIST, CascadeType.MERGE, CascadeType.REFRESH,
     CascadeType.DETACH})
  @PrimaryKeyJoinColumn
  public Classroom classroom;

  public DateTimeClass(LocalDate date, int time, Classroom
     classroom) {
    this.date = date;
    this.time = time;
    this.classroom = classroom;
  }
```

```
125   public DateTimeClass() {

      }

      @Override
130   public String toString() {
        return "DateTimeClass{" +
          "date=" + date.toString() +
          ", time=" + time +
          ", classroom=" + classroom.num +
135       '}';
      }

      @Override
      public boolean equals(Object o) {
140     if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        DateTimeClass that = (DateTimeClass) o;
        return  Objects.equals(date, that.date) && time == that.
           time && classroom.num.equals(that.classroom.num);
      }
145
      @Override
      public int hashCode() {
        return Objects.hash(date, time, classroom);
      }
150   @Override
      public int compareTo(DateTimeClass dtc) {
        int d = this.date.compareTo(dtc.date);
        int cl = this.classroom.num.compareTo(dtc.classroom.num);
        int t = Integer.compare(this.time, dtc.time);
155     return d != 0 ? d : (cl != 0 ? cl : t);
      }
    }

    @Data
160 @Accessors(chain = true)
    @Entity
    @Audited
    public class Event implements Comparable<Event> {
      @Id
165   @GeneratedValue(strategy = GenerationType.AUTO)
      public Long id;
```

```
    @ManyToOne(fetch = FetchType.EAGER, cascade = {CascadeType.
       PERSIST, CascadeType.MERGE, CascadeType.REFRESH,
       CascadeType.DETACH})
    @PrimaryKeyJoinColumn
    public StudyGroup group;

    @OneToMany(fetch = FetchType.LAZY, cascade = CascadeType.ALL
       )
    @PrimaryKeyJoinColumn
    public Set<Teacher> teacher;

    @ManyToOne(fetch = FetchType.EAGER, cascade = {CascadeType.
       PERSIST, CascadeType.MERGE, CascadeType.REFRESH,
       CascadeType.DETACH})
    @PrimaryKeyJoinColumn
    public AttestationType type;

    @Column(nullable = true)
    public String course;
    @Column(nullable = true)
    public Integer wishedClassroomType;
    @Transient
    public String wishedClassroomNums;
    @Transient
    public String link;
    @Transient
    public Integer eventGroup = -1; // группа аттестаций, которы
       е желательно проводить в один день

    public Event(StudyGroup group, Set<Teacher> teacher,
       AttestationType type, String course, int
       wishedClassroomType) {
      this.group = group;
      this.teacher = teacher;
      this.type = type;
      this.course = course;
      this.wishedClassroomType = wishedClassroomType;
      this.wishedClassroomNums = "";
    }
    public Event(StudyGroup group, Set<Teacher> teacher,
       AttestationType type, String course) {
      this.group = group;
      this.teacher = teacher;
      this.type = type;
      this.course = course;
```

```
    this.wishedClassroomType = 1;
    this.wishedClassroomNums = "";
}
public Event(StudyGroup group, String course,
    AttestationType type) {
    this.group = group;
    this.teacher = new HashSet<>();
    this.course = course;
    this.type = type;
    this.wishedClassroomType = 1;
    this.wishedClassroomNums = "";
}

public boolean isGroupSpecEnabled(List<String> spec){
    if (spec.isEmpty()) return true;
    for (String s: spec) {
        if (this.group.name.startsWith(s)){
            return true;
        }
    }
    return false;
}
public boolean isGroupLevelEnabled(List<Integer> levels){
    return levels.isEmpty() || levels.contains(group.level);
}

public Event() {
}

// чтобы при сортировке сначала был больший приоритет
@Override
public int compareTo(Event event) {
    int prior = Integer.compare(Teacher.maxPrior(event.teacher
        ), Teacher.maxPrior(this.teacher));
    if (prior != 0) return prior;
    int dates = Integer.compare(Teacher.minDatesLength(event.
        teacher), Teacher.minDatesLength(this.teacher));
    if (dates != 0) return dates;
    int size = Integer.compare(event.teacher.size(), this.
        teacher.size());
    if (size != 0) return size;
    int t = this.teacher.toString().compareTo(event.teacher.
        toString());
    if (t != 0) return t;
```

```
    int evGr = Integer.compare(this.eventGroup, event.
       eventGroup);
    return evGr;
  }

  @Override
  public String toString() {
    return "Event{" +
      "group=" + group.toString() +
      ", teacher=" + teacher +
      ", type=" + type +
      ", course='" + course + '\'' +
      ", wishedClassroomType=" + wishedClassroomType +
      ", wishedClassroomNums=" + wishedClassroomNums +
      ", wishedClassroomNums=" + link +
      ", eventGroup= " + eventGroup +
      '}';
  }

  public StudyGroup getGroup() {
    return group;
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Event event = (Event) o;
    return Objects.equals(group.name, event.group.name) &&
       Objects.equals(teacher, event.teacher) && Objects.
       equals(type.type, event.type.type) && Objects.equals(
       course, event.course);
  }

  @Override
  public int hashCode() {
    return Objects.hash(group, teacher, type, course);
  }
}

@Data
@Accessors(chain = true)
@Entity
@Audited
public class Exam {
```

```
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public Long id;
    @Column(nullable = false)
    public String groupName;
    @Column(nullable = false)
    public String specName;
    @Column(nullable = false)
    public String course;
    @Column(nullable = false)
    public String type;
    @Column(nullable = false)
    public String levels;
    @Column(nullable = false)
    public String teacherName;

    public Exam(String specName, String groupName, String course
        , String type, String level, String teacherName) {
      this.specName = specName;
      this.groupName = groupName;
      this.course = course;
      this.type = type;
      this.levels = level;
      this.teacherName = teacherName;
    }

    public Exam() {
    }

}

@Data
@Accessors(chain = true)
@Entity
@Audited
public class ExamFromRuz {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public Long id;
    @Column(nullable = false)
    public int level;
    @Column(nullable = false)
    public String groupName;
    @Column(nullable = false)
    public String course;
```

```
    @OneToMany(fetch = FetchType.LAZY, cascade = CascadeType.ALL
       )
    @PrimaryKeyJoinColumn
330 public Set<Teacher> teacherExams;

    @Transient
    public Set<String> teacherName;

335 public ExamFromRuz() {

    }

    public Set<String> getTeacherName() {
340   if (teacherExams != null) {
        return teacherExams.stream().map(x -> x.name).collect(
          Collectors.toSet());
      } else {
        return new HashSet<>();
      }
345 }

    public ExamFromRuz(String groupName, String course, int
       level) {
      this.groupName = groupName;
      this.course = course;
350   this.level = level;
    }

    public ExamFromRuz(Set<String> teacherName, String groupName
       , String course, int level) {
      this.teacherName = teacherName;
355   Set<Teacher> set = new HashSet<>();
      teacherName.forEach(x -> set.add(new Teacher(x)));
      this.teacherExams = set;
      this.groupName = groupName;
      this.course = course;
360   this.level = level;
    }

    public ExamFromRuz(String groupName, Set<Teacher> teachers,
       String course, int level) {
      this.teacherName = teachers.stream().map(x -> x.name).
        collect(Collectors.toSet());
365   this.teacherExams = teachers;
      this.groupName = groupName;
```

```
    this.course = course;
    this.level = level;
  }

  @Override
  public int hashCode() {
    return groupName.hashCode() + course.hashCode();
  }

  @Override
  public String toString() {
    return "Rasp{" +
      "teacherName='" + teacherName + '\'' +
      ", groupName='" + groupName + '\'' +
      ", course='" + course + '\'' +
      ", level='" + level + '\'' +
      '}';
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    ExamFromRuz examFromRuz = (ExamFromRuz) o;
    return Objects.equals(groupName, examFromRuz.groupName) &&
        Objects.equals(course, examFromRuz.course);
  }
}

@Data
@Accessors(chain = true)
@Entity
@Audited
public class ExternalExamsLink{
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  public Long id;

  @Column(nullable = false)
  public String link;

  public ExternalExamsLink(String link){
    this.link = link;
  }

```

```
  public ExternalExamsLink(){
  }
}

@Data
@Accessors(chain = true)
@Entity
@Audited
public class ResultLink{
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  public Long id;

  @Column(nullable = false)
  public String link;

  public ResultLink(String link){
    this.link = link;
  }

  public ResultLink(){
  }
}

@Data
@Accessors(chain = true)
@Entity
@Audited
public class SessionDates {

  @Id
  @Column(nullable = false)
  public LocalDate day;

  public SessionDates(LocalDate day) {
    this.day = day;
  }

  public SessionDates() {
  }

  @Override
  public String toString() {
    return day.toString();
  }
```

```
}

@Data
@Accessors(chain = true)
@Entity
@Audited
public class SessionTime {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  public Long id;

  public SessionTime() {
  }

  @Column(nullable = false)
  public  Integer time;

  public SessionTime(Integer time) {
    this.time = time;
  }
}

@Data
@Accessors(chain = true)
@Entity
@Audited
public class StudyGroup implements Comparable<Object> {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  public Long id;
  @Column(nullable = false)
  public String name;
  @Column(nullable = false)
  public Integer size;
  @Column(nullable = false)
  public Integer level;

  @OneToMany(mappedBy = "group", fetch = FetchType.LAZY,
     cascade = CascadeType.ALL)
  @JsonManagedReference
  private List<Event> event;

  public StudyGroup(String name, int size, int level) {
    this.name = name;
    this.size = size;
```

```
500     this.level = level;
    }

    public StudyGroup() {
    }
505
    @Override
    public int compareTo(Object o) {
      if (!(o instanceof StudyGroup) || this.name == null)
         return 0;
      StudyGroup g = (StudyGroup) o;
510   if (g.name == null) return 0;
      return this.name.compareTo(g.name);
    }

    @Override
515 public int hashCode() {
      return Objects.hash(name);
    }

    @Override
520 public String toString() {
      return "Group{" +
        "id=" + id +
        ", name='" + name + '\'' +
        ", size=" + size +
525     ", level=" + level +
        '}';
    }
}

530 @Data
@Accessors(chain = true)
@Entity
@Audited
public class Teacher {
535   @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public Long id;

    @Column(nullable = false)
540 public String name;

    @OneToMany(mappedBy = "teacher", fetch = FetchType.LAZY,
       cascade = CascadeType.ALL)
```

```
    @JsonManagedReference
    public List<TeacherExamDate> teacherDate;

    @Transient
    public List<LocalDate> date;

    @OneToMany(mappedBy = "teacher", fetch = FetchType.LAZY,
       cascade = CascadeType.ALL)
    @JsonManagedReference
    public List<TeacherExamTime> teacherTime;

    @Transient
    public List<Integer> time;

    @Column(nullable = false)
    public int prior;

    public Teacher(String name, List<LocalDate> date, List<
       Integer> time, int prior) {
      this.name = name;
      this.date = date;
      this.time = time;
      this.prior = prior;
    }
    public Teacher(String name, Event event) {
      this.name = name;
      this.prior = 1;
      //this.event = event;
    }

    public Teacher(String name) {
      this.name = name;
      this.prior = 1;
    }

    public Teacher() {
    }

    public static int maxPrior(Set<Teacher> set){
      return set.stream().mapToInt(x -> x.prior).max().orElse(0)
         ;
    }
    public static int minDatesLength(Set<Teacher> set){
      return set.stream().mapToInt(x -> x.date.size()).min().
         orElse(0);
```

```
  }

  @Override
  public String toString() {
    return name;
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Teacher teacher = (Teacher) o;
    return Objects.equals(name, teacher.name);
  }

  @Override
  public int hashCode() {
    return Objects.hash(name);
  }

  public int compareTo(Teacher teacher) {
    return Integer.compare(prior, teacher.prior);
  }
}

@Data
@Accessors(chain = true)
@Entity
@Audited
public class TeacherExamDate{
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  public Long id;
  @Column(nullable = false)
  public LocalDate date;

  @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL
     )
  @JsonBackReference
  private Teacher teacher;

  public TeacherExamDate() {
  }
}
```

```
@Data
@Accessors(chain = true)
@Entity
@Audited
public class TeacherExamTime{
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  public Long id;
  @Column(nullable = false)
  public Integer time;

  @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL
     )
  @JsonBackReference
  private Teacher teacher;

  public TeacherExamTime() {
  }
}
```

Приложение 3

Модуль взаимодействия с API ruz.spbstu.ru

```
@Controller
public class ApiRuz {
  public Set<ExamFromRuz> rasp;
  private Map<String, Set<Group>> specMap;
  private Map<String, Set<ExamFromRuz>> raspMap;
  private List<Integer> enabledLevels = List.of(1, 2, 3, 4, 5,
      6);
  private List<String> allGroupsNames;

  private List<StudyGroup> allGroups;
  private Set<ExamFromRuz> allExamFromRuz = new HashSet<>();

  public static List<DateTime> getDateTimesByRoomAndDate(
     String room, LocalDate localDate) {
    List<com.fleshka4.spbstu.ruz.api.models.Room> rooms =
       RuzSpbStu.getAuditoriesByBuildingId(25).getRooms();
    List<DateTime> dt = new ArrayList<>();
    int id = -1;
    for (int i = 0; i < rooms.size(); i++) {
      if (rooms.get(i).getName().equals(room)) {
        id = rooms.get(i).getId();
        break;
      }
    }
    if (id != -1) {
      List<Day> days = RuzSpbStu.
         getScheduleByAuditoryIdAndDate(id, localDate).getDays
         ();
      for (Day d : days) {
        LocalDate date = LocalDate.parse(d.getDate());
        List<Lesson> lessons = d.getLessons();
        for (Lesson l : lessons) {
          int start = l.getTimeStart().getHour();
          int end = l.getTimeEnd().getHour();
          for (int i = start; i <= end; i++) {
            dt.add(new DateTime(date, i));
          }
        }
      }
    }
```

```
    return dt;
  }

  public ApiRuz() {
  }

  public ApiRuz(Set<ExamFromRuz> exams, Iterable<String> spec)
      {
    this.allExamFromRuz = exams;
    raspMap = new HashMap<>();
    for (String code : spec) {
      raspMap.put(code, new HashSet<>());
      for (ExamFromRuz ex : exams) {
        if (ex.groupName.startsWith(code)) {
          raspMap.get(code).add(ex);
        }
      }
    }
  }

  public ApiRuz(List<Integer> levels, Iterable<String>
     specNames) {
    enabledLevels = levels;
    initGroups(specNames);
    initRasp();
  }

  private int getRealLevel(Group g) {
    int level = g.getLevel();
    if (g.getKind() == 1) {
      level += 4;
    }
    return level;
  }

  private void initGroups(Iterable<String> spec) {
    specMap = new HashMap<>();
    RuzSpbStu.getGroupsbyFacultyId(95).forEach(x -> {
      for (String specCode : spec) {
        if (x.getNameGroup().startsWith(specCode)) {
          if (enabledLevels.contains(getRealLevel(x))) {
            if (!specMap.containsKey(specCode)) {
              Set<Group> set = new HashSet<>();
              set.add(x);
              specMap.put(specCode, set);
```

```
            }
            specMap.get(specCode).add(x);
          }
        }
      }
    });
    initAllGroups();
  }

  private void initAllGroups() {
    allGroupsNames = new ArrayList<>();
    allGroups = new ArrayList<>();
    for (Set<Group> set : specMap.values()) {
      allGroupsNames.addAll(set.stream().map(Group::
         getNameGroup).collect(Collectors.toList()));
      allGroups.addAll(set.stream()
      .map(x -> new StudyGroup(x.getNameGroup(), 10, x.
         getLevel() + 4 * x.getKind()))
      .collect(Collectors.toList()));
    }
    allGroupsNames.sort(String::compareTo);
  }

  private void initAllRasp() {
    this.allExamFromRuz = new HashSet<>();
    for (Set<ExamFromRuz> set : raspMap.values()) {
      this.allExamFromRuz.addAll(set);
    }
  }

  public List<StudyGroup> getAllGroups() {
    return allGroups;
  }

  public List<String> getAllGroupsNames() {
    return allGroupsNames;
  }

  public Set<ExamFromRuz> getAllRasp() {
    return this.allExamFromRuz;
  }

  private void initRasp() {
    raspMap = new HashMap<>();
```

```
    for (Map.Entry<String, Set<Group>> entry : specMap.
       entrySet()) {
      Set<Group> list = entry.getValue();
      String specCode = entry.getKey();
125   raspMap.put(specCode, new HashSet<>());
      for (Group g : list) {
        String groupName = g.getNameGroup();
        int idGroup = g.getIdGroup();
        int level = getRealLevel(g);
130     for (int i = -2; i < 6; i++) {
          java.util.ArrayList<Day> days = RuzSpbStu.
             getScheduleByGroupIdAndDate(idGroup, LocalDate.
             now().minusWeeks(i)).getDays();
          for (Day d : days) {
            List<Lesson> lessons = d.getLessons();
            for (Lesson x : lessons) {
135           String sub = x.getSubject();
              List<Teacher> t = x.getTeachers();
              if (t != null) {
                Set<String> set = new HashSet<>();
                for (Teacher te : t) {
140               set.add(te.getFullName());
                }
                ExamFromRuz r = new ExamFromRuz(set, groupName
                   , sub, level);
                Set<ExamFromRuz> examFromRuzSet = raspMap.get(
                   specCode);
                if (examFromRuzSet.contains(r)) {
145               for (ExamFromRuz currentExamFromRuz :
                     examFromRuzSet) {
                    if (currentExamFromRuz.equals(r)) {
                      Set<String> n = currentExamFromRuz.
                         teacherName;
                      set.addAll(n);
                      examFromRuzSet.remove(currentExamFromRuz
                         );
150                   break;
                    }
                  }
                }
                ExamFromRuz ra = new ExamFromRuz(set,
                   groupName, sub, level);
155             raspMap.get(specCode).add(ra);
              }
            }
```

```
            }
          }
160     }
    }
    initAllRasp();
  }
}
```

Приложение 4

Классы-репозитории

```
  @Repository
  public interface AttestationTypeRepository extends
     CrudRepository<AttestationType, Long> {
    @Query("SELECT r FROM AttestationType r where r.type =
       ?1")
    public List<AttestationType> getAttestationTypeByName(
       String name);
  }

@Repository
public interface ClassroomRepository extends CrudRepository<
   Classroom, Long> {
  @Query("SELECT r FROM Classroom r ORDER BY r.num")
  public List<Classroom> allClassroom();
}

@Repository
public interface CurriculumRepository extends CrudRepository<
   Curriculum, Long> {
  @Query("SELECT r FROM Curriculum r where r.specCode = ?1")
  public List<Curriculum> findBySpecCode(String spec);
  @Query("SELECT r FROM Curriculum r where r.specCode in ?1")
  public List<Curriculum> findBySpecCode(List<String> spec);
}

@Repository
public interface DateTimeClassRepository extends
   CrudRepository<DateTimeClass, Long> {
}

@Repository
public interface EventRepository extends CrudRepository<Event,
    Long> {
  @Query("SELECT r FROM Event r where r.group is not null
     order by r.group.name, r.course")
  public List<Event> getEvent();
}

@Repository
```

```
public interface ExamFromRuzRepository extends CrudRepository<
    ExamFromRuz, Long> {
  @Query("SELECT count(r) FROM ExamFromRuz r")
  public int countExams();
}

@Repository
public interface ExamRepository extends CrudRepository<Exam,
    Long> {
  @Query("SELECT r FROM Exam r where r.specName = ?1")
  public List<Exam> getExamBySpec(String spec);
}

@Repository
public interface ExternalExamsLinkRepository extends
    CrudRepository<ExternalExamsLink, Long> {
}

@Repository
public interface GroupRepository extends CrudRepository<
    StudyGroup, Long> {
}

@Repository
public interface ResultLinkRepository extends CrudRepository<
    ResultLink, Long> {
}

@Repository
public interface SessionDatesRepository extends CrudRepository
    <SessionDates, Long>{
  @Query("SELECT r.day FROM SessionDates r ORDER BY r.day")
  public List<LocalDate> sessionDates();
}

@Repository
public interface SessionTimeRepository extends CrudRepository<
    SessionTime, Long> {
  @Query("SELECT time FROM SessionTime r ")
  public List<Integer> sessionTime();
  @Query("SELECT max(time) FROM SessionTime r ")
  public Integer getMaxTime();
  @Query("SELECT min(time) FROM SessionTime r ")
  public Integer getMinTime();
```

```
  @Query("SELECT count(time) FROM SessionTime r where time =
     ?1")
  public Integer countTimes(Integer time);
}

@Repository
public interface TeacherRepository extends CrudRepository<
   Teacher, Long> {
  @Query("SELECT r FROM Teacher r where r.name = ?1")
  public List<Teacher> getTeacherByName(String name);
}
```

Приложение 5

Класс SessionService

```
@Controller
public class SessionService {

  public SessionDao dao;

  @Autowired
  public SessionService(SessionDao dao) {
    this.dao = dao;
  }

  @RequestMapping(value = "/session")
  public String index(Model model) {
    return "session/index";
  }

  @RequestMapping(value = "/session/date_settings")
  public String dateSettings(Model model) {
    model.addAttribute("dates", dao.getSessionDates());
    return "session/date_settings/index";
  }

  @RequestMapping(value = "/session/curriculum")
  public String curriculum(Model model) {
    createEvents();
    Iterable<Event> events = dao.getEvents();
    model.addAttribute("events", events);
    return "session/curriculum/index";
  }

  @RequestMapping(value = "/session/class_settings")
  public String classSettings(Model model) {
    model.addAttribute("classrooms", dao.getClassrooms());
    return "session/class_settings/index";
  }
  @RequestMapping(value = "/session/clean_exam")
  public String cleanExam(Model model) {
    dao.clearExam();
    return "session/curriculum/index";
  }
```

```
@RequestMapping(value = "/session/external_exams")
public String externalExamsSettings(Model model) {
  model.addAttribute("link", dao.getExternalExamsLink());
  return "session/external_exams/index";
}

@RequestMapping(value = "/session/attestationTypes")
public String attestationTypeSettings(Model model) {
  model.addAttribute("attestationTypes", dao.
     getAttestationType());
  return "session/attestationTypes/index";
}

@RequestMapping(value = "/session/time_settings")
public String timeSettings(Model model) {
  return "session/time_settings/index";
}

@RequestMapping(value = "/session/exam_settings")
public String examSettings(Model model) {
  return "session/exam_settings/index";
}

@RequestMapping(value = "/session/teacher_settings")
public String teacherSettings(Model model) {
  return "session/teacher_settings/index";
}

@RequestMapping(value = "/session/result/clear_links")
public String clear_links(Model model) {
  dao.clearLinks();
  model.addAttribute("rasp", dao.getResultLink());
  return "session/result/link/index";
}

@RequestMapping(value = "/update_exam", method =
   RequestMethod.POST)
public @ResponseBody
String update_exam(
@RequestParam(name = "specCode") String specCode,
@RequestParam(name = "examFile") String examFile
) throws IOException {
  String tmp = "tmpexams";
  Writer w = new FileWriter(tmp);
  w.write(examFile);
```

```
    w.close();
    CSVReader reader = new CSVReader(new FileReader(tmp), ',',
        '"');
    String[] nextLine;
    List<Curriculum> curriculumList = new ArrayList<>();
    while ((nextLine = reader.readNext()) != null) {
      curriculumList.add(new Curriculum(specCode, nextLine[0],
          nextLine[1], nextLine[2]));
    }
    dao.createCurriculum(curriculumList);
    new File(tmp).delete();
    return "session";
  }
  @RequestMapping(value = "/update_exam_teacher", method =
     RequestMethod.POST)
  public @ResponseBody
  String update_exam_teacher(
  @RequestParam(name = "spec") String spec,
  @RequestParam(name = "examFile") String examFile
  ) throws IOException {
    String tmp = "tmpexamst";
    Writer w = new FileWriter(tmp);
    w.write(examFile);
    w.close();
    CSVReader reader = new CSVReader(new FileReader(tmp), ',',
        '"');
    String[] nextLine;
    List<Exam> examList = new ArrayList<>();
    while ((nextLine = reader.readNext()) != null) {
      examList.add(new Exam(spec, nextLine[2], nextLine[0],
          nextLine[3], nextLine[1], nextLine[4]));
    }
    dao.createExam(spec, examList);
    dao.getExams().forEach(System.out::println);
    new File(tmp).delete();
    return "session";
  }

  public void generateWindows(List<LocalDate> dates, List<
     Integer> time, List<Classroom> classrooms) {
    System.out.println("Start generate windows");
    if (dates.size() != 0 && time.size() != 0 && classrooms.
       size() != 0) {
      List<DateTimeClass> dtc = createListDTC(dates, time,
         classrooms);
```

```
      dao.createDateTimeClass(dtc);
    }
    System.out.println("Stop generate windows");
  }

  public static List<DateTimeClass> createListDTC(List<
     LocalDate> dates, List<Integer> time, List<Classroom>
     classrooms){
    List<DateTimeClass> dtc = new ArrayList<>();
    int d1 = dates.get(0).getDayOfWeek().getValue();
    List<DateTime> dateTimes = new ArrayList<>();
    for (LocalDate date : dates) {
      for (Classroom classroom : classrooms) {
        if (date.getDayOfWeek().getValue() == d1) {
          dateTimes = ApiRuz.getDateTimesByRoomAndDate(
             classroom.num, date);
        }
        List<Integer> dateTimesThisDate = new ArrayList<>();
        for (DateTime dateTime : dateTimes) {
          if (dateTime.date.equals(date)) {
            dateTimesThisDate.add(dateTime.time);
          }
        }
        for (Integer t : time) {
          if (!dateTimesThisDate.contains(t)) {
            dtc.add(new DateTimeClass(date, t, classroom));
          }
        }

      }
    }
    return dtc;
  }

  @RequestMapping(value = "/update_time", method =
     RequestMethod.POST)
  public @ResponseBody
  String update_time(
  @RequestParam(name = "time1") String time1,
  @RequestParam(name = "time2") String time2
  ) {
    List<SessionTime> times = new ArrayList<>();
    for (int i = Integer.parseInt(time1.substring(0, 2)); i <=
        Integer.parseInt(time2.substring(0, 2)); i++) {
      times.add(new SessionTime(i));
```

```
    }
    dao.createTimes(times);
    List<LocalDate> dates = dao.getSessionDates();
    List<Integer> time = dao.getTime();
    List<Classroom> classrooms = dao.getClassrooms();
    generateWindows(dates, time, classrooms);
    return "session";
}
@RequestMapping(value = "/update_external", method =
    RequestMethod.POST)
public @ResponseBody
String update_external(
@RequestParam(name = "externalId") String externalId
) {
    dao.createExternalExamsLink(externalId);
    return "session";
}

@RequestMapping(value = "/update_class", method =
    RequestMethod.POST)
public @ResponseBody
String updateClass(
@RequestParam(name = "classroomFile") String classroomFile
) {
    String[] lines = classroomFile.split("\\r\\n");
    List<Classroom> list = new ArrayList<>();
    for (String line : lines) {
        //Формат: номер аудитории, количество мест
        String[] str = line.split(",");
        String num = str[0];
        int type = Integer.parseInt(str[1]);
        int sits = Integer.parseInt(str[2]);
        Classroom cl = new Classroom(num, type, sits);
        list.add(cl);
    }
    dao.createClassrooms(list);
    List<LocalDate> dates = dao.getSessionDates();
    List<Integer> time = dao.getTime();
    List<Classroom> classrooms = dao.getClassrooms();
    generateWindows(dates, time, classrooms);
    return "session";
}

@RequestMapping(value = "/update_att", method =
    RequestMethod.POST)
```

```
public @ResponseBody
String updateAttestationTypes(@RequestParam(name = "
    attestationFile") String attetstaionTypes) {
  String[] lines = attetstaionTypes.split("\\r\\n");
  List<AttestationType> list = new ArrayList<>();
  for (String line : lines) {
    String[] str = line.split(",");
    AttestationType att = new AttestationType(str[0],
    Integer.parseInt(str[1])
    , Integer.parseInt(str[2])
    , Integer.parseInt(str[3])
    , Integer.parseInt(str[4]));
    list.add(att);
  }
  dao.createAttestationType(list);
  return "session";
}

public SessionService createEvents() {
  dao.clearEvents();
  dao.clearExamsFromRuz();
  Iterable<Curriculum> curriculumList = dao.getCurriculum();
  curriculumList.forEach(x -> System.out.println(x.specCode)
    );
  Set<String> specList = new HashSet<>();
  curriculumList.forEach(x -> specList.add(x.specCode));
  ApiRuz apiRuz = new ApiRuz(List.of(1, 2, 3, 4, 5, 6),
    specList);
  apiRuz.getAllRasp().forEach(x -> dao.createExamFromRuz(x))
    ;
  dao.createGroups(apiRuz.getAllGroups());
  Iterable<StudyGroup> groupIterable = dao.getGroups();
  HashMap<String, StudyGroup> groupHashMap = new HashMap<>()
    ;
  for (StudyGroup gr : groupIterable) {
    groupHashMap.put(gr.name, gr);
  }

  List<ExamFromRuz> examFromRuz = (List<ExamFromRuz>) dao.
    getExamFromRuz();
  List<AttestationType> attestationTypes = (List<
    AttestationType>) dao.getAttestationType();
  int odd = LocalDate.now().getMonth().getValue() < 3 ||
    LocalDate.now().getMonth().getValue() > 8 ? 1 : 2;
  for (ExamFromRuz exam : examFromRuz) {
```

```
      System.out.println("examFromRuz");
      String l = ((Integer) ((exam.level - 1) * 2 + odd)).
         toString();
      for (Curriculum curriculum : curriculumList) {
        if (exam.course.equalsIgnoreCase(curriculum.course)
245     && exam.groupName.startsWith(curriculum.specCode)
        && curriculum.semesters.contains(l)) {
          StudyGroup group = groupHashMap.get(exam.groupName);
          String att = curriculum.types.toLowerCase();
          for (AttestationType at : attestationTypes) {
250         if (att.contains(at.type)) {
              dao.createEvent(group, exam.getTeacherName(), at
                 , exam.course);
              break;
            }
          }
255     }
      }
    }
    Iterable<Exam> exams = dao.getExams();
    for (Exam exam : exams) {
260   String[] str = exam.teacherName.split(",");
      Set<String> teachers = new HashSet<>(Arrays.asList(str))
         ;
      for (AttestationType at : attestationTypes) {
        if (exam.type.equalsIgnoreCase(at.type)) {
          dao.createEvent(new StudyGroup(exam.groupName, 10,
             Integer.parseInt(exam.levels)/2 + 1), teachers,
             at, exam.course);
265       break;
        }
      }
    }
    return this;
270 }

    @RequestMapping(value = "/update_date", method =
       RequestMethod.POST)
    public @ResponseBody
    String updateDate(
275 @RequestParam(name = "days") String[] days
    ) {
      List<LocalDate> list = new ArrayList<>();
      for (int i = 0; i < days.length / 2; i++) {
        LocalDate date1 = null;
```

```
      LocalDate date2 = null;
      try {
        date1 = LocalDate.parse(days[2 * i]);
      } catch (Exception ignored) {
      }
      try {
        date2 = LocalDate.parse(days[2 * i + 1]);
      } catch (Exception ignored) {
      }
      if (date1 != null && date2 != null) {
        while (date1.compareTo(date2) <= 0) {
          if (date1.getDayOfWeek().getValue() != 7) {
            list.add(date1);
          }
          date1 = date1.plusDays(1);
        }
      }
    }
    if (list.size() > 0) {
      dao.createDates(list);
    }
    List<LocalDate> dates = dao.getSessionDates();
    List<Integer> time = dao.getTime();
    List<Classroom> classrooms = dao.getClassrooms();
    generateWindows(dates, time, classrooms);
    return "session";
  }
}
```

Приложение 6

Класс GoogleFormService

```
@Controller
public class GoogleFormService {
  public SessionDao dao;

  @Autowired
  public GoogleFormService(SessionDao dao) {
    this.dao = dao;
  }

  private static final String APPLICATION_NAME = "Scheduler
     ";
  private static final JsonFactory JSON_FACTORY =
     JacksonFactory.getDefaultInstance();
  private static final String TOKENS_DIRECTORY_PATH = "
     tokens";
  private static final List<String> SCOPES = Collections.
     singletonList(SheetsScopes.SPREADSHEETS_READONLY);
  private static final String REFRESH_TOKEN_PATH = "/
     refresh_token.txt";
  private static final String SPREADSHEET_ID_PATH = "/
     spreadsheet_id.txt";
  private static final String TEACHER_SPREADSHEET_ID_PATH =
     "/teacher_spreadsheet_id.txt";
  private static final String CREDENTIALS_FILE_PATH = "/
     credentials.json";
  private static String SPREADSHEET_ID;
  private static String TEACHER_SPREADSHEET_ID;
  private static String REFRESH_TOKEN;
  private static Set<String> links = new HashSet<>();

  static {
    try {
      TEACHER_SPREADSHEET_ID = new BufferedReader(new
         InputStreamReader(GoogleFormService.class.
         getResourceAsStream(TEACHER_SPREADSHEET_ID_PATH))).
         readLine();
      REFRESH_TOKEN = new BufferedReader(new
         InputStreamReader(GoogleFormService.class.
         getResourceAsStream(REFRESH_TOKEN_PATH))).readLine
         ();
```

```
      } catch (IOException e) {
        e.printStackTrace();
      }
    }

    enum ColumnName {
      PRIOR("Я совместитель"),
      DATE1("Начало интервала дат, когда предпочтительна аттес
        тация"),
      DATE2("Конец интервала дат, когда предпочтительна аттест
        ация"),
      NAME("Ваше имя"),
      HELPER("Ассистент"),
      ABS1("Первый день отсутствия"),
      ABS2("Последний день отсутствия"),
      DOF("Дни недели, в которые Вы НЕ МОЖЕТЕ присутствовать н
        а аттестациях"),
      TIME1("Самое раннее время, в которое Вы сможете начать п
        роводить аттестации"),
      TIME2("Самое позднее время, в которое может закончиться
        аттестация"),
      GROUP("Сгруппируйте те аттестации, которые желательно пр
        оводить в один день (для этого у совместных аттестаци
        й проставьте галочки в одном столбце)"),
      ROOM("Выберете тип аудитории, необходымый для проведения
         аттестации"),
      NUMS("Если необходимо проводить аттестации только в конк
        ретных аудиторях, перечислите их номера ниже через за
        пятую"),
      ;

      ColumnName(String s) {
      }
    }

    private static Map<ColumnName, Integer> columns = new
       HashMap<>();

    static {
      columns.put(ColumnName.PRIOR, 2);
      columns.put(ColumnName.DATE1, 3);
      columns.put(ColumnName.DATE2, 4);
      columns.put(ColumnName.NAME, 5);
      columns.put(ColumnName.HELPER, 6);
      columns.put(ColumnName.ABS1, 7);
```

```
    columns.put(ColumnName.ABS2, 8);
    columns.put(ColumnName.DOF, 9);
    columns.put(ColumnName.TIME1, 10);
    columns.put(ColumnName.TIME2, 11);
  }

  @RequestMapping(value = "/session/teachers/updatelink")
  public String teachers_updatelink(Model model) {
    model.addAttribute("link", "https://docs.google.com/
       spreadsheets/d/" + TEACHER_SPREADSHEET_ID + "/edit");
    return "session/teachers/updatelink/index";
  }

  @RequestMapping(value = "/session/teachers/update")
  @ResponseBody
  public String teachers_update(Model model,
  @RequestParam(name = "spec") String[] spec) {

    List<String> specs;
    if (spec != null) {
      specs = new ArrayList<>(Arrays.asList(spec));
    } else {
      specs = new ArrayList<>();
    }
    System.out.println(specs);
    List<Event> event = dao.getEvents();
    try {
      final NetHttpTransport HTTP_TRANSPORT =
         GoogleNetHttpTransport.newTrustedTransport();
      Sheets sheetsService = new Sheets.Builder(
         HTTP_TRANSPORT, JSON_FACTORY, getCredentials(
         HTTP_TRANSPORT))
      .setApplicationName(APPLICATION_NAME)
      .build();

      Spreadsheet spreadSheet = sheetsService.spreadsheets()
      .get(TEACHER_SPREADSHEET_ID)
      .execute();

      List<List<Object>> list = new ArrayList<>();
      for (int i = 0; i < event.size(); i++) {
        Event e = event.get(i);
        if (specs.isEmpty()) {
          e.teacher.forEach(t ->
          list.add(List.of(
```

```
            t.name,
            e.course,
            e.group.name
            ))
            );
          } else {
            for (String specCode : specs) {
              System.out.println(e.course+" "+e.type.type);
              if (e.group.name.startsWith(specCode)) {
                e.teacher.forEach(t ->
                list.add(List.of(
                t.name,
                e.course+" "+e.type.type,
                e.group.name
                ))
                );
                break;
              }
            }
          }
        }
        ValueRange body = new ValueRange().setValues(list);
        sheetsService.spreadsheets().values()
        .clear(spreadSheet.getSpreadsheetId(), "A1:C", new
           ClearValuesRequest())
        .execute();
        sheetsService.spreadsheets().values()
        .update(spreadSheet.getSpreadsheetId(), "A1:C", body)
        .setValueInputOption("RAW")
        .execute();
        System.out.println(spreadSheet.getSpreadsheetUrl());
        return spreadSheet.getSpreadsheetUrl();
      } catch (Exception e) {
        System.out.println(e);
        return e.toString();
      }
    }

    public List<Solution> getExternalExams() {
      List<Solution> res = new ArrayList<>();
      String link = dao.getExternalExamsLink();
      try {
        final NetHttpTransport HTTP_TRANSPORT =
           GoogleNetHttpTransport.newTrustedTransport();
```

```
        ValueRange response = new Sheets.Builder(
           HTTP_TRANSPORT, JSON_FACTORY, getCredentials(
           HTTP_TRANSPORT))
        .setApplicationName(APPLICATION_NAME)
        .build()
        .spreadsheets().values()
        .get(link, "A2:G")
        .execute();
        List<List<Object>> values = response.getValues();
        if (values == null || values.isEmpty()) {
          System.out.println("No data found.");
        } else {
          for (int i = 0; i < values.size(); i++) {
            List<Object> row = values.get(i);
            Set<Teacher> t = new HashSet<>();
            String str = row.get(1).toString();
            String[] arr = str.substring(1, str.length() - 1).
               split(", ");
            for (String n : arr) {
              t.add(new Teacher(n));
            }
            List<AttestationType> attestationTypes = (List<
               AttestationType>) dao.getAttestationType();
            AttestationType attestationType = null;
            String att = row.get(3).toString().toLowerCase();
            for (AttestationType at : attestationTypes) {
              if (att.contains(at.type)) {
                attestationType = at;
                break;
              }
            }
            if (attestationType == null) {
              attestationType = attestationTypes.get(0);
            }
            Classroom classroom = null;
            String classroomName = row.get(6).toString();
            List<Classroom> classrooms = dao.getClassrooms();
            for (Classroom cl : classrooms) {
              if (classroomName.equals(cl.num)) {
                classroom = cl;
                break;
              }
            }
            if (classroom == null) {
```

```
185               classroom = new Classroom(classroomName, 1, 100)
                      ;
                }
                Event e = new Event(new StudyGroup(row.get(0).
                   toString(), 10, 0), t, attestationType, row.get
                   (2).toString());
                LocalDate date;
                try {
190               date = LocalDate.parse(row.get(4).toString());
                } catch (Exception ex) {
                  date = LocalDate.parse(row.get(4).toString(),
                     DateTimeFormatter.ofPattern("dd.MM.yyyy"));
                }
                DateTimeClass dt = new DateTimeClass(
195             date,
                Integer.parseInt(row.get(5).toString()),
                classroom);
                Solution s = new Solution(e, dt);
                res.add(s);
200           }
          }
        } catch (Exception e) {
          System.out.println(e);
        } finally {
205       return res;
        }
      }

      public String generateLink(List<Solution> solutions) {
210     try {
          final NetHttpTransport HTTP_TRANSPORT =
             GoogleNetHttpTransport.newTrustedTransport();
          Sheets sheetsService = new Sheets.Builder(
             HTTP_TRANSPORT, JSON_FACTORY, getCredentials(
             HTTP_TRANSPORT))
          .setApplicationName(APPLICATION_NAME)
          .build();
215
          Spreadsheet spreadSheet = new Spreadsheet().
             setProperties(
          new SpreadsheetProperties().setTitle("Session Schedule
             "));

          Spreadsheet result = sheetsService
220       .spreadsheets()
```

```
        .create(spreadSheet).execute();

        List<List<Object>> list = new ArrayList<>();

        list.add(Arrays.asList(
        "Группа",
        "Преподаватели",
        "Дисциплина",
        "Тип",
        "Дата",
        "Время",
        "Аудитория"
        ));
        for (int i = 0; i < solutions.size(); i++) {
          Solution sol = solutions.get(i);
          if (i == 0 || !solutions.get(i - 1).event.equals(sol
             .event)) {
            list.add(Arrays.asList(
            sol.event.group.name,
            sol.event.teacher.toString(),
            sol.event.course,
            sol.event.type.type,
            sol.dtc.date.toString(),
            String.valueOf(sol.dtc.time),
            sol.dtc.classroom.num
            ));
          }
        }
        ValueRange body = new ValueRange().setValues(list);
        sheetsService.spreadsheets().values()
        .update(result.getSpreadsheetId(), "A1", body)
        .setValueInputOption("RAW")
        .execute();
        System.out.println(result.getSpreadsheetUrl());
        return result.getSpreadsheetUrl();
      } catch (Exception e) {
        System.out.println(e);
        return null;
      }
    }

    public static void main(String[] args) {
      try {
        final NetHttpTransport HTTP_TRANSPORT =
           GoogleNetHttpTransport.newTrustedTransport();
```

```
        Sheets sheetsService = new Sheets.Builder(
            HTTP_TRANSPORT, JSON_FACTORY, getCredentials(
            HTTP_TRANSPORT))
265         .setApplicationName(APPLICATION_NAME)
            .build();

            Spreadsheet spreadSheet = sheetsService.spreadsheets()
            .get(TEACHER_SPREADSHEET_ID)
270         .execute();

            List<List<Object>> list = new ArrayList<>();
            ValueRange body = new ValueRange().setValues(list);
            sheetsService.spreadsheets().values()
275         .clear(spreadSheet.getSpreadsheetId(), "A1:C", new
              ClearValuesRequest())
            .execute();
          } catch (Exception e) {
            System.out.println(e);
          }
280     }

        @RequestMapping(value = "/session/result")
        public String result(Model model) {
          return "session/result/index";
285     }

        @RequestMapping(value = "/session/teachers")
        public String teachers(Model model) {
          return "session/teachers/index";
290     }

        List<List<Solution>> sol = new ArrayList<>();

        @RequestMapping(value = "/session/result/create", method =
            RequestMethod.POST)
295     public @ResponseBody
        String resultCreate(Model model,
        @RequestParam(name = "spec") String[] spec,
        @RequestParam(name = "levels") Integer[] level) {
          List<String> specs;
300       List<Integer> levels;
          if (spec != null) {
            specs = new ArrayList<>(Arrays.asList(spec));
          } else {
            specs = new ArrayList<>();
```

```
305       }
          if (level != null) {
            levels = new ArrayList<>(Arrays.asList(level));
          } else {
            levels = new ArrayList<>();
310       }
          List<Event> e = getEventsWithWishes(specs, levels);
          for (Event ev : e) {
            System.out.println(ev);
          }
315       List<Classroom> classroomList = new ArrayList<>();
          for (String l : links) {
            classroomList.add(new Classroom(l, 0, 500));
          }
          List<DateTimeClass> dateTimeClassList = dao.
             getDateTimeClass();
320       dateTimeClassList.addAll(SessionService.createListDTC(
             dao.getSessionDates(), dao.getTime(), classroomList))
             ;
          ScheduleAllSolutions scheduleAllSolutions =
          new ScheduleAllSolutions(e,
          dateTimeClassList,
          getExternalExams(),
325       8, 1, new ArrayList<>());
          sol = scheduleAllSolutions.findSolutions().getSolutions
             ();
          System.out.println(sol);
          for (List<Solution> solutions : sol) {
            String link = generateLink(solutions);
330         if (link != null) {
              dao.createResultLink(link);
            }
          }
          return "session/result/link/index";
335     }

        @RequestMapping(value = "session/result/link")
        public String resultLink(Model model) {
          model.addAttribute("rasp", dao.getResultLink());
340       return "session/result/link/index";
        }

        public static Credential getCredentials(final
           NetHttpTransport HTTP_TRANSPORT) throws IOException {
```

```
        InputStream in = GoogleFormService.class.
            getResourceAsStream(CREDENTIALS_FILE_PATH);
345     if (in == null) {
          throw new FileNotFoundException("Resource not found: "
              + CREDENTIALS_FILE_PATH);
        }
        GoogleClientSecrets clientSecrets = GoogleClientSecrets.
            load(JSON_FACTORY, new InputStreamReader(in));
        String clientId = clientSecrets.getDetails().getClientId
            ();
350     String clientSecret = clientSecrets.getDetails().
            getClientSecret();
        GoogleCredential credential = new GoogleCredential.
            Builder()
        .setTransport(HTTP_TRANSPORT)
        .setJsonFactory(JSON_FACTORY)
        .setClientSecrets(clientId, clientSecret)
355     .build();
        credential.setAccessToken(getNewToken(REFRESH_TOKEN,
            clientId, clientSecret));
        credential.setRefreshToken(REFRESH_TOKEN);
        return credential;
      }
360
      public static String getNewToken(String refreshToken,
          String clientId, String clientSecret) throws
      IOException {
        ArrayList<String> scopes = new ArrayList<>();
        scopes.add(SheetsScopes.SPREADSHEETS);
365     TokenResponse tokenResponse = new
            GoogleRefreshTokenRequest(new NetHttpTransport(), new
             JacksonFactory(),
        refreshToken, clientId, clientSecret).setScopes(scopes).
            setGrantType("refresh_token").execute();
        return tokenResponse.getAccessToken();
      }

370   public List<Event> getEventsWithWishes(List<String> spec,
          List<Integer> levels) {
        List<Event> events = dao.getEvents();
        List<Event> result = new ArrayList<>();
        try {
          SPREADSHEET_ID = new BufferedReader(new
              InputStreamReader(GoogleFormService.class.
```

```
            getResourceAsStream(SPREADSHEET_ID_PATH))).readLine
            ();
375
          final NetHttpTransport HTTP_TRANSPORT =
            GoogleNetHttpTransport.newTrustedTransport();
          ValueRange response = new Sheets.Builder(
            HTTP_TRANSPORT, JSON_FACTORY, getCredentials(
            HTTP_TRANSPORT))
          .setApplicationName(APPLICATION_NAME)
          .build()
380       .spreadsheets().values()
          .get(SPREADSHEET_ID, "'Ответы на форму (1)'")
          .execute();
          List<List<Object>> values = response.getValues();
          if (values == null || values.isEmpty()) {
385         System.out.println("No data found.");
          } else {
            List<Object> colNames = values.get(0);
            for (Event event : events) {
              if (event.isGroupSpecEnabled(spec) && event.
                 isGroupLevelEnabled(levels)) {
390             List<Object> helpers = new ArrayList<>();
                for (Teacher teacher : event.teacher) {
                  for (int i = 1; i < values.size(); i++) {
                    List<Object> row = values.get(i);
                    String name = (String) row.get(columns.get(
                       ColumnName.NAME));
395                 if (teacher.name.equals(name) && row.size()
                       > columns.get(ColumnName.HELPER)) {
                      Object h = row.get(columns.get(ColumnName.
                         HELPER));
                      if (!h.equals("")) {
                        helpers.add(h);
                      }
400                 }
                  }
                }
                for (Object h : helpers) {
                  boolean f = true;
405               for (Teacher t : event.teacher) {
                    if (h.equals(t.name)) {
                      f = false;
                      break;
                    }
410               }
```

```
                if (f) event.teacher.add(new Teacher(h.
                    toString()));
              }
              for (Teacher teacher : event.teacher) {
                for (int i = 1; i < values.size(); i++) {
415               List<Object> row = values.get(i);
                  String name = (String) row.get(columns.get(
                      ColumnName.NAME));
                  if (teacher.name.equals(name)) {
                    Object date1 = row.size() > columns.get(
                        ColumnName.DATE1) ? row.get(columns.get
                        (ColumnName.DATE1)) : "";
                    Object date2 = row.size() > columns.get(
                        ColumnName.DATE2) ? row.get(columns.get
                        (ColumnName.DATE2)) : "";
420                 Object abs1 = row.size() > columns.get(
                        ColumnName.ABS1) ? row.get(columns.get(
                        ColumnName.ABS1)) : "";
                    Object abs2 = row.size() > columns.get(
                        ColumnName.ABS2) ? row.get(columns.get(
                        ColumnName.ABS2)) : "";
                    Object time1 = row.size() > columns.get(
                        ColumnName.TIME1) ? row.get(columns.get
                        (ColumnName.TIME1)) : "";
                    Object time2 = row.size() > columns.get(
                        ColumnName.TIME2) ? row.get(columns.get
                        (ColumnName.TIME2)) : "";
                    Object dof = row.size() > columns.get(
                        ColumnName.DOF) ? row.get(columns.get(
                        ColumnName.DOF)) : "";
425                 Object prior = row.size() > columns.get(
                        ColumnName.PRIOR) ? row.get(columns.get
                        (ColumnName.PRIOR)) : "";
                    teacher.time = generateTimeList(time1,
                        time2);
                    teacher.date = generateDatesList(date1,
                        date2, abs1, abs2, dof);
                    teacher.prior = prior.equals("Да") ? 2 :
                        1;
                    for (int j = 0; j < row.size(); j++) {
430                   if (!row.get(j).toString().equals("")) {
                        if (colNames.get(j).toString().
                            startsWith("Выберете тип аудитории,
                             необходымый для проведения аттеста
                            ции")) {
```

```
                        event.wishedClassroomType =
                            Classroom.getTypeByTeacherWish(
                            row.get(j).toString());
                    }
                    if (colNames.get(j).toString().
                        startsWith("Если необходимо проводи
                        ть аттестации только в конкретных а
                        удиторях")) {
                      event.wishedClassroomNums = row.get(
                          j).toString();
                    }
                    if (colNames.get(j).toString().
                        startsWith("Ссылка на")) {
                      event.link = row.get(j).toString();
                      links.add(event.link);
                    }
                  }
                }
              }
              if (teacher.date == null) {
                teacher.date = dao.getSessionDates();
              }
              if (teacher.time == null) {
                teacher.time = dao.getTime();
              }
            }
            result.add(event);
          }
        }
        int eventsInOneDay = 0;
        for (int i = 1; i < values.size(); i++) {
          List<Object> row = values.get(i);
          List<Integer> groupRow = new ArrayList<>();
          List<String> groupTitle = new ArrayList<>();
          Map<Integer, Integer> map = new HashMap<>();
          Set<Integer> set = new HashSet<>();
          for (int j = 0; j < row.size(); j++) {
            if (!row.get(j).toString().equals("") &&
                colNames.get(j).toString().startsWith("Сгрупп
                ируйте те аттестации")) {
              Integer x = Integer.parseInt(row.get(j).
                  toString().substring("Совместные аттестации
                   ".length()));
              groupRow.add(x);
```

```
                set.add(x);
                groupTitle.add(colNames.get(j).toString());
              }
            }
            int max = eventsInOneDay;
            int setSize = set.size();
            for (int j = 0; j < groupRow.size(); j++) {
              for (Event event : events) {
                if (event.eventGroup == -1 && groupTitle.get(j
                   ).contains(event.group.name)
                && groupTitle.get(j).contains(event.course)) {
                  int gr = groupRow.get(j);
                  if (!map.containsKey(gr)) {
                    map.put(gr, 1);
                    event.eventGroup = eventsInOneDay + gr;
                  } else {
                    map.put(gr, map.get(gr) + 1);
                    event.eventGroup = eventsInOneDay + gr +
                       setSize * ((map.get(gr) - 1) / 2);
                  }
                  if (event.eventGroup > max) max = event.
                     eventGroup;
                }
              }
            }
            eventsInOneDay = max;
          }
        }
      } catch (
      Exception e) {
        System.out.println(e.toString());
      }
      return result;
    }

    private List<LocalDate> generateDatesList(Object d1,
       Object d2, Object a1, Object a2, Object dof) {
      LocalDate date1, date2, abs1, abs2;
      date1 = d1 != "" ? LocalDate.parse((String) d1,
         DateTimeFormatter.ofPattern("dd.MM.yyyy")) :
         LocalDate.MIN;
      date2 = d2 != "" ? LocalDate.parse((String) d2,
         DateTimeFormatter.ofPattern("dd.MM.yyyy")) :
         LocalDate.MAX;
```

```
      abs1 = a1 != "" ? LocalDate.parse((String) a1,
         DateTimeFormatter.ofPattern("dd.MM.yyyy")) :
         LocalDate.MAX;
      abs2 = a2 != "" ? LocalDate.parse((String) a2,
         DateTimeFormatter.ofPattern("dd.MM.yyyy")) :
         LocalDate.MIN;
      List<Integer> dofs = new ArrayList<>();
505   String d = (String) dof;
      String[] week = {"Понедельник", "Втрорник", "Среда", "Че
         тверг", "Пятница", "Суббота"};
      for (int i = 0; i < week.length; i++) {
        if (d.contains(week[i])) {
          dofs.add(i);
510     }
      }
      List<LocalDate> dates = dao.getSessionDates();
      List<LocalDate> res = new ArrayList<>();
      for (LocalDate date : dates) {
515     if (!(date.compareTo(abs1) >= 0 && date.compareTo(abs2
           ) <= 0)
        && date.compareTo(date1) >= 0
        && date.compareTo(date2) <= 0
        && !dofs.contains(date.getDayOfWeek().getValue() - 1))
            {
          res.add(date);
520     }
      }
      return res;
    }

525 private List<Integer> generateTimeList(Object t1, Object
       t2) {
      int time1, time2;
      time1 = t1 != "" ? Integer.parseInt(t1.toString()) : 0;
      time2 = t2 != "" ? Integer.parseInt(t2.toString()) : 24;
      List<Integer> times = dao.getTime();
530   List<Integer> res = new ArrayList<>();
      for (Integer time : times) {
        if (time >= time1 && time <= time2) {
          res.add(time);
        }
535   }
      return res;
    }
  }
```

Приложение 7

Класс SessionDao

```
@Component
public class SessionDao {
  private SessionDatesRepository sessionDatesRepository;
  private final SessionTimeRepository sessionTimeRepository;
  private final ExamFromRuzRepository examFromRuzRepository;
  private final ExamRepository examRepository;
  private final DateTimeClassRepository
     dateTimeClassRepository;
  private final ResultLinkRepository resultLinkRepository;
  private final GroupRepository groupRepository;
  private final EventRepository eventRepository;
  private final AttestationTypeRepository
     attestationTypeRepository;
  private final CurriculumRepository curriculumRepository;
  private final TeacherRepository teacherRepository;
  private final ClassroomRepository classroomRepository;
  private final ExternalExamsLinkRepository
     externalExamsLinkRepository;

  @Autowired
  public SessionDao(SessionDatesRepository
     sessionDatesRepository,
  SessionTimeRepository sessionTimeRepository,
  AttestationTypeRepository attestationTypeRepository,
  DateTimeClassRepository dateTimeClassRepository,
  ExamFromRuzRepository examFromRuzRepository,
  ExamRepository examRepository,
  ResultLinkRepository resultLinkRepository,
  GroupRepository groupRepository,
  ExternalExamsLinkRepository externalExamsLinkRepository,
  EventRepository eventRepository,
  CurriculumRepository curriculumRepository,
  TeacherRepository teacherRepository,
  ClassroomRepository classroomRepository) {
    this.sessionDatesRepository = sessionDatesRepository;
    this.groupRepository = groupRepository;
    this.dateTimeClassRepository = dateTimeClassRepository;
    this.sessionTimeRepository = sessionTimeRepository;
    this.examFromRuzRepository = examFromRuzRepository;
    this.examRepository = examRepository;
```

```
      this.externalExamsLinkRepository =
         externalExamsLinkRepository;
      this.eventRepository = eventRepository;
      this.resultLinkRepository = resultLinkRepository;
      this.attestationTypeRepository =
         attestationTypeRepository;
      this.curriculumRepository = curriculumRepository;
      this.classroomRepository = classroomRepository;
      this.teacherRepository = teacherRepository;
    }

    @Transactional
    public Set<ExamFromRuz> getAllExamFromRuz() {
      Set<ExamFromRuz> set = new HashSet<>();
      examFromRuzRepository.findAll().forEach(set::add);
      return set;
    }

    @Transactional
    public SessionDao clearLinks() {
      resultLinkRepository.deleteAll();
      return this;
    }

    @Transactional
    public SessionDao clearExam() {
      examRepository.deleteAll();
      curriculumRepository.deleteAll();
      return this;
    }

    @Transactional
    public List<DateTimeClass> getDateTimeClass() {
      List<DateTimeClass> dtc = new ArrayList<>();
      dateTimeClassRepository.findAll().forEach(dtc::add);
      dtc.sort(DateTimeClass::compareTo);
      return dtc;
    }

    @Transactional
    public SessionDao createDateTimeClass(List<DateTimeClass>
       dtc) {
      dateTimeClassRepository.deleteAll();
      dateTimeClassRepository.saveAll(dtc);
      return this;
```

```
80      }

        @Transactional
        public SessionDao createExamFromRuz(ExamFromRuz
           examFromRuz) {
          ExamFromRuz ex = examFromRuzRepository.save(examFromRuz)
             ;
85        return this;
        }

        @Transactional
        public SessionDao createAttestationType(List<
           AttestationType> attestationTypes) {
90        attestationTypeRepository.deleteAll();
          attestationTypeRepository.saveAll(attestationTypes);
          return this;
        }

95      @Transactional
        public SessionDao createEvent(StudyGroup group,
        Set<String> teacher,
        AttestationType type,
        String course) {
100       Set<Teacher> set = teacher.stream().map(Teacher::new).
             collect(Collectors.toSet());
          Event event = new Event(group, set, type, course);
          teacherRepository.saveAll(set);
          groupRepository.save(group);
          attestationTypeRepository.save(type);
105       eventRepository.save(event);
          return this;
        }

        @Transactional
110     public SessionDao clearEvents() {
          //teacherRepository.deleteAll();
          eventRepository.deleteAll();
          return this;
        }
115
        @Transactional
        public SessionDao clearExamsFromRuz() {
          //  teacherExamRepository.deleteAll();
          examFromRuzRepository.deleteAll();
120       return this;
```

```
    }

    @Transactional
    public Iterable<AttestationType> getAttestationType() {
      return attestationTypeRepository.findAll();
    }

    @Transactional
    public List<Event> getEvents() {
      List<Event> list = new ArrayList<>();
      eventRepository.findAll().forEach(x -> {
        if (x.group != null && x.teacher != null) {
          list.add(x);
        }
      });
      list.sort((o1, o2) -> {
        int s1 = o1.teacher.size();
        int s2 = o2.teacher.size();
        if (s1 != s2) return Integer.compare(s1, s2);
        Teacher t1 = o1.teacher.stream().findFirst().get();
        Teacher t2 = o2.teacher.stream().findFirst().get();
        int tp = Integer.compare(t1.prior, t2.prior);
        int tn = t1.name.compareTo(t2.name);
        int g = o1.group.compareTo(o2.group);
        return tp != 0 ? tp : tn != 0 ? tn : g == 0 ? o1.
           course.compareTo(o2.course) : g;
      });
      return list;
    }

    @Transactional
    public SessionDao createGroups(List<StudyGroup> groups) {
      groupRepository.deleteAll();
      groupRepository.saveAll(groups);
      return this;
    }

    @Transactional
    public Iterable<StudyGroup> getGroups() {
      return groupRepository.findAll();
    }

    @Transactional
    public SessionDao createTimes(List<SessionTime> time) {
      sessionTimeRepository.deleteAll();
```

```
165       sessionTimeRepository.saveAll(time);
          return this;
        }

        @Transactional
170     public SessionDao createCurriculum(List<Curriculum>
           curriculumList) {
          String spec = curriculumList.get(0).specCode;
          curriculumRepository.deleteAll(curriculumRepository.
             findBySpecCode(spec));
          curriculumRepository.saveAll(curriculumList);
          return this;
175     }

        @Transactional
        public SessionDao createExam(String spec, List<Exam>
           examList) {
          examRepository.deleteAll(examRepository.getExamBySpec(
             spec));
180       examRepository.saveAll(examList);
          return this;
        }

        @Transactional
185     public SessionDao createResultLink(String link) {
          resultLinkRepository.save(new ResultLink(link));
          return this;
        }

190     @Transactional
        public Iterable<ResultLink> getResultLink() {
          return resultLinkRepository.findAll();
        }

195     @Transactional
        public Iterable<Curriculum> getCurriculumBySpec(String
           spec) {
          return curriculumRepository.findBySpecCode(spec);
        }

200     @Transactional
        public Teacher getTeacherByName(String name) {
          List<Teacher> teachers = teacherRepository.
             getTeacherByName(name);
          return teachers != null ? teachers.get(0) : null;
```

```
    }

    @Transactional
    public List<Teacher> getTeachers() {
      List<Teacher> res = new ArrayList<>();
      teacherRepository.findAll().forEach(res::add);
      return res;
    }

    @Transactional
    public Iterable<Curriculum> getCurriculumBySpec(List<
       String> spec) {
      return curriculumRepository.findBySpecCode(spec);
    }

    @Transactional
    public Iterable<Curriculum> getCurriculum() {
      return curriculumRepository.findAll();
    }

    @Transactional
    public Iterable<ExamFromRuz> getExamFromRuz() {
      return examFromRuzRepository.findAll();
    }

    @Transactional
    public Iterable<Exam> getExams() {
      return examRepository.findAll();
    }

    @Transactional
    public SessionDao createTime(Integer t) {
      if (sessionTimeRepository.countTimes(t) == 0) {
        sessionTimeRepository.save(new SessionTime(t));
      }
      return this;
    }

    @Transactional
    public SessionDao createClassrooms(List<Classroom>
       classrooms) {
      classroomRepository.deleteAll();
      classroomRepository.saveAll(classrooms);
      return this;
    }
```

```
    @Transactional
    public SessionDao createDates(List<LocalDate> days) {
250   sessionDatesRepository.deleteAll();
      for (LocalDate d : days) {
        sessionDatesRepository.save(new SessionDates(d));
      }
      return this;
255 }

    @Transactional
    public List<LocalDate> getSessionDates() {
      return sessionDatesRepository.sessionDates();
260 }

    @Transactional
    public Integer getMinTime() {
      return sessionTimeRepository.getMinTime();
265 }

    @Transactional
    public List<Integer> getTime() {
      List<Integer> list = new ArrayList<>();
270   sessionTimeRepository.findAll().forEach(x -> list.add(x.
        time));
      return list;
    }

    @Transactional
275 public Integer getMaxTime() {
      return sessionTimeRepository.getMaxTime();
    }

    @Transactional
280 public List<Classroom> getClassrooms() {
      return classroomRepository.allClassroom();
    }

    @Transactional
285 public SessionDao createExternalExamsLink(String
       externalId) {
      externalExamsLinkRepository.deleteAll();
      externalExamsLinkRepository.save(new ExternalExamsLink(
        externalId));
      return this;
```

```
    }

    @Transactional
    public String getExternalExamsLink() {
      List<ExternalExamsLink> list = new ArrayList<>();
      externalExamsLinkRepository.findAll().forEach(list::add)
          ;
      if (list.size() > 0) {
        return list.get(0).link;
      } else {
        return "";
      }
    }
  }
```

Приложение 8

Тестовый класс

```
public class SchedulerTest {
  @Test
  public void generateDtcTest(){
    LocalDate d1 = LocalDate.parse("2021-01-01");
    LocalDate d2 = LocalDate.parse("2021-01-02");
    LocalDate d3 = LocalDate.parse("2021-01-05");
    List<LocalDate> dateList = List.of(d1,d2,d3);
    List<Integer> time = List.of(8,9);
    Classroom c1 = new Classroom("101", 10, 10);
    Classroom c2 = new Classroom("102", 10, 10);
    List<Classroom> classrooms = List.of(c1, c2);
    List<DateTimeClass> dtc = SessionService.createListDTC(
       dateList, time, classrooms);
    List<DateTimeClass> exp = new ArrayList<>();
    exp.add(new DateTimeClass(d1,8, c1));
    exp.add(new DateTimeClass(d1,9, c1));
    exp.add(new DateTimeClass(d1,8, c2));
    exp.add(new DateTimeClass(d1,9, c2));
    exp.add(new DateTimeClass(d2,8, c1));
    exp.add(new DateTimeClass(d2,9, c1));
    exp.add(new DateTimeClass(d2,8, c2));
    exp.add(new DateTimeClass(d2,9, c2));
    exp.add(new DateTimeClass(d3,8, c1));
    exp.add(new DateTimeClass(d3,9, c1));
    exp.add(new DateTimeClass(d3,8, c2));
    exp.add(new DateTimeClass(d3,9, c2));

    Assert.assertArrayEquals(exp.toArray(), dtc.toArray());
  }

  @Test
  public void checkDuration(){
    List<LocalDate> dateList = List.of(LocalDate.parse
       ("2021-01-01"), LocalDate.parse("2021-01-02"),
       LocalDate.parse("2021-01-15"));
    List<Integer> time = List.of(8,9,10,11,12,13);
    List<Classroom> classrooms = List.of(new Classroom("101",
       10, 10), new Classroom("102", 10, 10));
    List<DateTimeClass> dtc = SessionService.createListDTC(
       dateList, time, classrooms);
```

```
    Set<Teacher> teachers = new HashSet<>();
    Teacher teacher = new Teacher("T1");
    teacher.time = time;
    teacher.date = dateList;
    teachers.add(teacher);
    List<Event> events = new ArrayList<>();
    int duration = 4;
    Event e1 = new Event(new StudyGroup("gr1", 10, 1),
       teachers, new AttestationType("Экз", 0, 2 , duration,
       1),"Course1");
    events.add(e1);
    SchedulePriorAlgorithm alg = new SchedulePriorAlgorithm(
       events, dtc, new ArrayList<>(), 8, 1, new ArrayList<>()
       );

    alg.findSolutions().getSolutions().forEach(System.out::
       println);
    List<Solution> list = alg.findSolutions().getSolutions().
       get(0);

    Assert.assertEquals(list.size(), duration);
  }

  @Test
  public void checkPauseBetweenExams1(){
    List<LocalDate> dateList = List.of(LocalDate.parse
       ("2021-01-01"), LocalDate.parse("2021-01-02"),
       LocalDate.parse("2021-01-15"));
    List<Integer> time = List.of(8,9,10,11,12,13);
    List<Classroom> classrooms = List.of(new Classroom("101",
       10, 10), new Classroom("102", 10, 10));
    List<DateTimeClass> dtc = SessionService.createListDTC(
       dateList, time, classrooms);
    Set<Teacher> teachers = new HashSet<>();
    Teacher teacher = new Teacher("T1");
    teacher.time = time;
    teacher.date = dateList;
    teachers.add(teacher);
    List<Event> events = new ArrayList<>();
    Event e1 = new Event(new StudyGroup("gr1", 10, 1),
       teachers, new AttestationType("Экз", 0, 2 , 4, 1),"
       Course1");
    Event e2 = new Event(new StudyGroup("gr1", 10, 1),
       teachers, new AttestationType("Экз", 0, 2 , 4, 1),"
       Course2");
```

```
    Event e3 = new Event(new StudyGroup("gr1", 10, 1),
        teachers, new AttestationType("Экз", 0, 2 , 4, 1),"
        Course3");
    events.add(e1);
    events.add(e2);
    events.add(e3);
    SchedulePriorAlgorithm alg = new SchedulePriorAlgorithm(
        events, dtc, new ArrayList<>(), 8, 1, new ArrayList<>()
        );
    List<List<Solution>> list = alg.findSolutions().
        getSolutions();
    Assert.assertEquals(0, list.size());
}

@Test
public void checkPauseBetweenExams2(){
    List<LocalDate> dateList = List.of(LocalDate.parse
        ("2021-01-01"), LocalDate.parse("2021-01-02"),
        LocalDate.parse("2021-01-15"));
    List<Integer> time = List.of(8,9,10,11,12,13);
    List<Classroom> classrooms = List.of(new Classroom("101",
        10, 10), new Classroom("102", 10, 10));
    List<DateTimeClass> dtc = SessionService.createListDTC(
        dateList, time, classrooms);
    Set<Teacher> teachers = new HashSet<>();
    Teacher teacher = new Teacher("T1");
    teacher.time = time;
    teacher.date = dateList;
    teachers.add(teacher);
    List<Event> events = new ArrayList<>();
    Event e1 = new Event(new StudyGroup("gr1", 10, 1),
        teachers, new AttestationType("Экз", 0, 2 , 4, 1),"
        Course1");
    Event e2 = new Event(new StudyGroup("gr1", 10, 1),
        teachers, new AttestationType("Экз", 0, 2 , 4, 1),"
        Course2");
    Event e3 = new Event(new StudyGroup("gr2", 10, 1),
        teachers, new AttestationType("Экз", 0, 2 , 4, 1),"
        Course3");
    events.add(e1);
    events.add(e2);
    events.add(e3);
    SchedulePriorAlgorithm alg = new SchedulePriorAlgorithm(
        events, dtc, new ArrayList<>(), 8, 1, new ArrayList<>()
        );
```

```
    List<List<Solution>> list = alg.findSolutions().
        getSolutions();
    System.out.println(list);
    Assert.assertEquals(1, list.size());
  }

  @Test
  public void priorAlgTeachersTime(){
    List<LocalDate> dateList = List.of(LocalDate.parse
        ("2021-01-01"), LocalDate.parse("2021-01-02"),
        LocalDate.parse("2021-01-15"));
    List<Integer> time = List.of(8,9,10,11,12,13,14,15,16,17);
    Classroom c1 = new Classroom("101", 10, 10);
    Classroom c2 = new Classroom("102", 10, 10);
    List<Classroom> classrooms = List.of(c1, c2);
    List<DateTimeClass> dtc = SessionService.createListDTC(
        dateList, time, classrooms);
    Set<Teacher> teachers = new HashSet<>();
    Teacher teacher = new Teacher("T1");
    teacher.time = List.of(14,15,16,17);
    teacher.date = dateList;
    teachers.add(teacher);
    List<Event> events = new ArrayList<>();
    Event e1 = new Event(new StudyGroup("gr1", 10, 1),
        teachers, new AttestationType("Экз", 0, 2 , 4, 1),"
        Course1");
    Event e2 = new Event(new StudyGroup("gr1", 10, 1),
        teachers, new AttestationType("Экз", 0, 2 , 4, 1),"
        Course2");
    events.add(e1);
    events.add(e2);
    SchedulePriorAlgorithm alg = new SchedulePriorAlgorithm(
        events, dtc, new ArrayList<>(), 8, 1, new ArrayList<>()
        );
    List<Solution> list = alg.findSolutions().getSolutions().
        get(0);
    for (Solution s: list) {
      Assert.assertTrue(s.dtc.time>=14 && s.dtc.time<=17);
    }
  }
}
```

Приложение 9

JavaScript файл

```
document.addEventListener("DOMContentLoaded", () => {

  const classroom = document.getElementsByClassName("
    hidden_checkbox");
  const days = document.getElementById("calendar");
  const specs = document.getElementById("specs");
  const specs_updt = document.getElementById("specs_updt");
  const clear_links = document.getElementById("clear_links");
  const add_days = document.getElementById("add_new_date");
  const add_spec = document.getElementById("add_spec");
  const add_spec_updt = document.getElementById("add_spec_updt
    ");
  const submit_exam = document.getElementById("submit_exam");
  const submit_class = document.getElementById("submit_class")
    ;
  const submit_att = document.getElementById("submit_att");
  const submit_time = document.getElementById("submit_time");
  const submit_date = document.getElementById("submit_date");
  const submit_groups = document.getElementById("submit_groups
    ");
  const submit_groups_updt = document.getElementById("
    submit_groups_updt");
  const submit_teacher = document.getElementById("
    submit_teacher");
  const submit_external = document.getElementById("
    submit_external");
  const clean_exam = document.getElementById("clean_exam");

  let date_count = 0;
  let spec_count = 0;
  let spec_updt_count = 0;
  let dels = [];
  dels[0] = document.getElementById("del_0");
  if (dels[0]) {
    dels[0].onclick = () => {
      document.getElementById('dates' + 0).style.display = "
        none";
    };
  }
```

```
let dels_spec = [];
dels_spec[0] = document.getElementById("del_s_0");
if (dels_spec[0]) {
  dels_spec[0].onclick = () => {
    document.getElementById('spec' + 0).style.display = "
       none";
  };
}
let dels_spec_updt = [];
dels_spec[0] = document.getElementById("del_s_updt_0");
if (dels_spec_updt[0]) {
  dels_spec_updt[0].onclick = () => {
    document.getElementById('spec_updt' + 0).style.display =
        "none";
  };
}
if (add_days) {
  add_days.onclick = () => {
    date_count++;
    let el = document.createElement("div");
    el.innerHTML = createNewDays(date_count);
    days.appendChild(el);
    days.onclick = function (e) {
      if (e.target.className === "del") {
        e.target.parentElement.remove();
      }
    };
  };
}

if (clear_links) {
  clear_links.onclick = () => {
    var request = new XMLHttpRequest();
    request.open('POST', '/session/result/clear_links', true
       );
    request.setRequestHeader(header, token);
    request.addEventListener('readystatechange', function ()
        {
      if ((request.readyState === 4) && (request.status ===
         200)) {
        console.log("checked");
      }
    });
    request.send();
    //window.open('/session/result/link', '_top');
```

```
    };
  }
  if (clean_exam) {
    clean_exam.onclick = () => {
      var request = new XMLHttpRequest();
      request.open('POST', '/session/clean_exam', true);
      request.setRequestHeader(header, token);
      request.addEventListener('readystatechange', function ()
          {
        if ((request.readyState === 4) && (request.status ===
           200)) {
          console.log("checked");
        }
      });
      request.send();
    };
  }
  if (submit_class) {
    submit_class.onclick = () => {
      const classroomFile = document.getElementById("classfile
         ");
      submitClass(classroomFile.files[0]);
    };
  }
  if (submit_att) {
    submit_att.onclick = () => {
      const attestationFile = document.getElementById("attfile
         ");
      submitAtt(attestationFile.files[0]);
    };
  }
  if (submit_external) {
    submit_external.onclick = () => {
      let external_id = document.getElementById("external_id")
         .value;
      submitExternal(external_id);
    };
  }
  if (submit_time) {
    submit_time.onclick = () => {
      const time1 = document.getElementById("time1");
      const time2 = document.getElementById("time2");
      submitTime(time1.value, time2.value);
    };
  }
```

```
if (add_spec) {
  add_spec.onclick = () => {
    spec_count++;
    let el = document.createElement("div");
    el.innerHTML = createNewSpec(spec_count);
    specs.appendChild(el);
    specs.onclick = function (e) {
      if (e.target.className === "del") {
        e.target.parentElement.remove();
      }
    };
  };
}
if (add_spec_updt) {
  add_spec_updt.onclick = () => {
    spec_updt_count++;
    let el = document.createElement("div");
    el.innerHTML = createNewSpecUpdt(spec_updt_count);
    specs_updt.appendChild(el);
    specs_updt.onclick = function (e) {
      if (e.target.className === "del") {
        e.target.parentElement.remove();
      }
    };
  };
}
if (submit_exam) {
  submit_exam.onclick = () => {
    let specCode = document.getElementById("specCode").value
       ;
    const examFile = document.getElementById("examfile").
       files[0];
    console.log(examFile);
    const examFileTeacher = document.getElementById("
       examfileteacher").files[0];
    console.log(examFileTeacher);
    if (specCode !== '' && examFile !== undefined) {
      submitExam(examFile, specCode);
    } else if (specCode !== '' && examFileTeacher !==
       undefined) {
      submitExamTeacher(examFileTeacher, specCode);
    }
  };
}
if (submit_teacher) {
```

```
    submit_teacher.onclick = () => {
      const teacherFile = document.getElementById("teacherfile
         ");
      submitTeacher(teacherFile.files[0]);
    };
  }
  if (submit_groups) {
    submit_groups.onclick = () => {
      const specitem = document.getElementsByClassName("
         spec__item");
      let specsarr = [];
      if (specitem.length === 0) {
        specsarr = null;
      } else {
        for (let i = 0; i < specitem.length; i++) {
          specsarr.push(specitem.item(i).value);
        }
      }
      const lev = document.getElementsByClassName("levelOption
         ");
      let levelarr = [];
      let i = 0;
      while (i < 6) {
        if (lev[i].checked) {
          levelarr.push(lev[i].value);

        }
        i++;
      }
      submitGroups(specsarr, levelarr);

    };
  }
  if (submit_groups_updt) {
    submit_groups_updt.onclick = () => {
      const specitem = document.getElementsByClassName("
         spec_updt__item");
      let specsarr = [];
      if (specitem.length === 0) {
        specsarr = null;
      } else {
        for (let i = 0; i < specitem.length; i++) {
          specsarr.push(specitem.item(i).value);
        }
      }
```

```
      console.log(specsarr)
      submitGroupsUpdt(specsarr);
    };
  }
  if (submit_date) {
    submit_date.onclick = () => {
      const daysIntervals = document.getElementsByClassName("
         date__item");
      let daysarr = [];
      if (daysIntervals.length === 0) {
        daysarr = null;
      } else {
        for (let i = 0; i < daysIntervals.length; i++) {
          daysarr.push(daysIntervals.item(i).value);
        }
      }
      submitDate(daysarr);
    };
  }
}
);
function createNewSpec(spec_count) {
  return '<div class="item" id="spec' +spec_count+ '">' +
  '<input type="text" name="spec" class="spec__item date__item
     spec">' +
  '<div class="del" id="del_s_' +spec_count+ '"> &#10006;</div
     >' +
  '</div>'
};
function createNewSpecUpdt(spec_count_updt) {
  return '<div class="item" id="spec_updt' +spec_count_updt+
     '">' +
  '<input type="text" name="spec_updt" class="spec_updt__item
     date__item spec_updt">' +
  '<div class="del" id="del_s_updt_' +spec_count_updt+ '">
     &#10006;</div>' +
  '</div>'
};
function createNewDays(date_count) {
  return '<div class="item" id="dates' + date_count + '">\n' +
  '                    <div class="date__items">\n' +
  '                        c \n' +
  '                        <input type="date" name="date1"
     class="date__item">\n' +
  '                        до \n' +
```

```
  '                          <input type="date" name="date2"
     class="date__item">\n' +
  '                      </div>\n' +
  '                      <div class="del" id="del_' + date_count
      + '"> &#10006;</div>\n' +
  '                  </div>'
};
const token = getMeta("_csrf");
const header = getMeta("_csrf_header");

function getMeta(metaName) {
  const metas = document.getElementsByTagName('meta');

  for (let i = 0; i < metas.length; i++) {
    if (metas[i].getAttribute('name') === metaName) {
      return metas[i].getAttribute('content');
    }
  }

  return '';
}

function createNewClassroom() {
  return '<div class="item">\n' +
  '                          Введите номер аудитории:\n' +
  '                          <input type="number" name="
     classroom_number">\n' +
  '                      </div>'
};

function readFile(input) {
  let file = input;

  let reader = new FileReader();

  reader.readAsText(file);

  reader.onload = function () {
    let res = reader.result;
    console.log(res);
    return res;
  };

  reader.onerror = function () {
    console.log(reader.error);
```

```
  };

}
function submitGroups(specarr, levelarr) {
  var request = new XMLHttpRequest();
  request.open('POST', '/session/result/create', true);
  request.setRequestHeader(header, token);
  request.addEventListener('readystatechange', function () {
    if ((request.readyState === 4) && (request.status === 200)
       ) {
      console.log("checked");
    }
  });
  var frm = new FormData();
  frm.append('spec', specarr)
  frm.append('levels', levelarr)
  console.log(specarr);
  console.log(levelarr);
  request.send(frm)
  window.open('/session', '_top');
};
function submitGroupsUpdt(specarr) {
  var request = new XMLHttpRequest();
  request.open('POST', '/session/teachers/update', true);
  request.setRequestHeader(header, token);
  request.addEventListener('readystatechange', function () {
    if ((request.readyState === 4) && (request.status === 200)
       ) {
      console.log("checked");
    }
  });
  var frm = new FormData();
  frm.append('spec', specarr)
  console.log(specarr);
  request.send(frm)
  window.open('/session/teachers/updatelink', '_top');
};
function submitSession(classroomFile, time1, time2, daysarr) {
  var request = new XMLHttpRequest();
  console.log(daysarr);
  request.open('POST', '/update', true);
  request.setRequestHeader(header, token);
  request.addEventListener('readystatechange', function () {
    if ((request.readyState === 4) && (request.status === 200)
       ) {
```

```
      console.log("checked");
    }
});
  var frm = new FormData();
  //            let fileText = readFile(classroomFile);
  //            console.log(fileText);
  frm.append('time1', time1);
  console.log(time1);
  frm.append('time2', time2);
  console.log(time2);
  frm.append('days', daysarr);
  console.log("SEND FRM");
  let fileText;
  let reader = new FileReader();
  reader.readAsText(classroomFile);
  reader.onload = function () {
    fileText = reader.result;
    console.log(fileText);
    frm.append('classroomFile', fileText);
    request.send(frm)
  };
  // window.open('/sended','_top');
};

function submitTime(time1, time2) {
  var request = new XMLHttpRequest();
  request.open('POST', '/update_time', true);
  request.setRequestHeader(header, token);
  request.addEventListener('readystatechange', function () {
    if ((request.readyState === 4) && (request.status === 200)
       ) {
      console.log("checked");
    }
  });
  var frm = new FormData();
  frm.append('time1', time1);
  console.log(time1);
  frm.append('time2', time2);
  console.log(time2);
  request.send(frm);
  window.open('/session', '_top');
};

function submitClass(classroomFile) {
  var request = new XMLHttpRequest();
```

```
  request.open('POST', '/update_class', true);
  request.setRequestHeader(header, token);
  request.addEventListener('readystatechange', function () {
    if ((request.readyState === 4) && (request.status === 200)
       ) {
      console.log("checked");
    }
  });
  var frm = new FormData();
  let fileText;
  let reader = new FileReader();
  reader.readAsText(classroomFile);
  reader.onload = function () {
    fileText = reader.result;
    console.log(fileText);
    frm.append('classroomFile', fileText);
    request.send(frm)
  };
  window.open('/session', '_top');
};
function submitAtt(attFile) {
  var request = new XMLHttpRequest();
  request.open('POST', '/update_att', true);
  request.setRequestHeader(header, token);
  request.addEventListener('readystatechange', function () {
    if ((request.readyState === 4) && (request.status === 200)
       ) {
      console.log("checked");
    }
  });
  var frm = new FormData();
  let fileText;
  let reader = new FileReader();
  reader.readAsText(attFile);
  reader.onload = function () {
    fileText = reader.result;
    console.log(fileText);
    frm.append('attestationFile', fileText);
    request.send(frm)
  };
  window.open('/session', '_top');
};

function submitTeacher(teacherFile) {
  var request = new XMLHttpRequest();
```

```
405   request.open('POST', '/update_teacher', true);
      request.setRequestHeader(header, token);
      request.addEventListener('readystatechange', function () {
        if ((request.readyState === 4) && (request.status === 200)
           ) {
          console.log("checked");
410     }
      });
      var frm = new FormData();
      console.log("SEND FRM");
      let fileText;
415   let reader = new FileReader();
      reader.readAsText(teacherFile);
      reader.onload = function () {
        fileText = reader.result;
        console.log(fileText);
420     frm.append('teacherFile', fileText);
        request.send(frm)
      };
      window.open('/session', '_top');
};
425
function submitExam(examFile, specCode) {
      var request = new XMLHttpRequest();
      request.open('POST', '/update_exam', true);
      request.setRequestHeader(header, token);
430   request.addEventListener('readystatechange', function () {
        if ((request.readyState === 4) && (request.status === 200)
           ) {
          console.log("checked");
        }
      });
435   var frm = new FormData();
      frm.append('specCode', specCode);
      let fileText;
      let reader = new FileReader();
      reader.readAsText(examFile);
440   reader.onload = function () {
        fileText = reader.result;
        console.log(fileText);
        frm.append('examFile', fileText);
        request.send(frm)
445   };
      window.open('/session', '_top');
};
```

```
function submitExamTeacher(examFileTeacher, specCode) {
  var request = new XMLHttpRequest();
  request.open('POST', '/update_exam_teacher', true);
  request.setRequestHeader(header, token);
  request.addEventListener('readystatechange', function () {
    if ((request.readyState === 4) && (request.status === 200)
       ) {
      console.log("checked");
    }
  });
  var frm = new FormData();
  frm.append('spec', specCode);
  let fileText;
  let reader = new FileReader();
  reader.readAsText(examFileTeacher);
  reader.onload = function () {
    fileText = reader.result;
    console.log(fileText);
    frm.append('examFile', fileText);
    request.send(frm)
  };
  window.open('/session', '_top');
};
function submitExternal(external_id) {
  var request = new XMLHttpRequest();
  request.open('POST', '/update_external', true);
  request.setRequestHeader(header, token);
  request.addEventListener('readystatechange', function () {
    if ((request.readyState === 4) && (request.status === 200)
       ) {
      console.log("checked");
    }
  });
  var frm = new FormData();
  frm.append('externalId', external_id);
  request.send(frm)
  window.open('/session', '_top');
};

function submitDate(daysarr) {
  var request = new XMLHttpRequest();
  console.log(daysarr);
  request.open('POST', '/update_date', true);
  request.setRequestHeader(header, token);
  request.addEventListener('readystatechange', function () {
```

```
    if ((request.readyState === 4) && (request.status === 200)
       ) {
      console.log("checked");
    }
  });
  var frm = new FormData();
  frm.append('days', daysarr);
  request.send(frm)
  window.open('/session', '_top');
}
```

Приложение 10

AppScript для генерации google-формы

```
function refresfForm(){
  /** id таблицы */
  var sh = SpreadsheetApp.openById("...");
  /** id формы */
  var form = FormApp.getActiveForm();
  clearForm(form);
  form.setTitle('Предпочтения преподавателей к проведению сессии')
  var pass = form.addTextItem().setTitle('Введите пароль').setRequired(true)
  var textValidation = FormApp.createTextValidation()
  .requireTextMatchesPattern('пароль')
  .setHelpText("Неверный пароль!")
  .build()
  pass.setValidation(textValidation);

  form.addPageBreakItem().setTitle('Предпочтения преподавателей к проведению сессии')
  var prior = form.addMultipleChoiceItem().setTitle('Я совместитель').setRequired(true)
  var priorPage = form.addPageBreakItem().setTitle('Раздел только для совместителей')
  .setHelpText('Далее можете выбрать интервал дат, в которые предпочтительно проводить аттестации. (Не менее 14 дней!)');
  form.addDateItem().setTitle('Начало интервала дат, когда предпочтительна аттестация')
  form.addDateItem().setTitle('Конец интервала дат, когда предпочтительна аттестация')

  var mainPage = form.addPageBreakItem().setTitle('Предпочтения к времени проведения аттестаций');
  var teachers = form.addListItem().setTitle('Ваше имя').setRequired(true);
  var helper = form.addListItem().setTitle('Выберете ассистента, если он есть');
  prior.setChoices([
  prior.createChoice('Да',priorPage),
  prior.createChoice('Нет',mainPage)])
```

```
form.addSectionHeaderItem().setTitle('Выберете интервал дат, в которые Вы НЕ будете доступны для проведения аттестаций')
.setHelpText('В этом пункте необходимо указать промежуток времени, на который Вы предварительно согласовали, к примеру, поездку на конференцию или другую уважительную причину отсутствия. ЕСЛИ ТАКОГО ПОЛЯ НЕТ, ОСТАВЬТЕ СЛЕДУЮЩИЕ ДВА ВОПРОСА БЕЗ ОТВЕТА.')
form.addDateItem().setTitle('Первый день отсутствия');
form.addDateItem().setTitle('Последний день отсутствия');

var dof = form.addCheckboxItem().setTitle('Дни недели, в которые Вы НЕ МОЖЕТЕ присутствовать на аттестациях');
dof.setChoices([dof.createChoice('Понедельник'),
dof.createChoice('Вторник'),
dof.createChoice('Среда'),
dof.createChoice('Четверг'),
dof.createChoice('Пятница'),
dof.createChoice('Суббота')]);

var t1 = form.addListItem().setTitle('Самое раннее время, в которое Вы сможете начать проводить аттестации');
t1.setChoices([t1.createChoice(8),
t1.createChoice(9),
t1.createChoice(10),
t1.createChoice(11),
t1.createChoice(12),
t1.createChoice(13),
t1.createChoice(14),
t1.createChoice(15),
t1.createChoice(16),
t1.createChoice(17),
t1.createChoice(18)])

var t2 = form.addListItem().setTitle('Самое позднее время, в которое может закончиться аттестация');
t2.setChoices([t2.createChoice(10),
t2.createChoice(11),
t2.createChoice(12),
t2.createChoice(13),
t2.createChoice(14),
t2.createChoice(15),
t2.createChoice(16),
t2.createChoice(17),
t2.createChoice(18),
```

```
    t2.createChoice(19),
    t2.createChoice(20)])

    var items = [];
    var itemsHelper = [];
    var course;
    var group;
    var greed;
    var greed_room;
    var cnt;
    var pages = [];


    var listlist = sh.getDataRange().getValues().sort();

    for (var i = 0; i < listlist.length; i++) {
      if (i==0 || listlist[i-1][0]!=listlist[i][0]){
        course = [];
        group = [];
        cnt = 0;
        var name = listlist[i][0];
        Logger.log(name);
        var newPage = form.addPageBreakItem().setTitle(name);
        pages.push(newPage);
        items.push(teachers.createChoice(name, newPage));
        itemsHelper.push(helper.createChoice(name));
        if (!((i==0 || listlist[i-1][0]!=listlist[i][0]) && (
           listlist.length-1==i || listlist[i+1][0]!=listlist[i
           ][0]))){
          greed = form.addGridItem().setTitle('Сгруппируйте те а
             ттестации, которые желательно проводить в один день
              (для этого у совместных аттестаций проставьте гало
             чки в одном столбце).  Проводится в 1 день могут то
             лько 2 аттестации. При этом, если аттестация - экза
             мен, то Вам необходимо обязательно указать ассистир
             ующего преподавателя, иначе предпочтение будет прои
             гнорировано.').setHelpText('Для аттестаций, которые
              можно проводить по одной в день можно не выбирать
             ничего');
        }
        //greed_room = form.addGridItem().setTitle('Выберете тип
            аудитории, необходымый для проведения аттестации');
        // form.addTextItem()
```

```
      // .setTitle('Если необходимо проводить аттестации тольк
         о в конкретных аудиторях, перечислите их номера ниже
         через запятую')
      // .setHelpText('Пример: 241,237');
    }
    cnt++;
    course.push(listlist[i][1]+' '+listlist[i][2])
    group.push('Совместные аттестации '+cnt);
    form.addSectionHeaderItem().setTitle(listlist[i][1]+' '+
       listlist[i][2])
    form.addTextItem().setTitle('Ссылка на Teams для проведени
       я дистанционной аттестации')
    .setHelpText(listlist[i][1]+' '+listlist[i][2]).
       setRequired(true);
    form.addTextItem().setTitle('Ссылка на курс на dl.spbstu.
       ru')
    .setHelpText(listlist[i][1]+' '+listlist[i][2]).
       setRequired(true);
    // if (i==listlist.length-1 || listlist[i][0]!= listlist[i
       +1][0]){

      if (!((i==0 || listlist[i-1][0]!=listlist[i][0]) && (
         listlist.length-1==i || listlist[i+1][0]!=listlist[i
         ][0]))){
        greed.setRows(course).setColumns(group);
      }
      //   greed_room.setRows(course).setColumns(['Любая','Люб
         ой компьютерный класс',
      //   'Любая аудитория с проектором', 'Дистанционно'])
      // }
  }
  teachers.setChoices(items);
  helper.setChoices(itemsHelper);
  var last = form.addPageBreakItem().setTitle('Обратная связь
     ');
  pages.forEach(p => p.setGoToPage(last));
  form.addParagraphTextItem().setTitle('Если приведённая выше
     форма не учитывает какие-то Ваши пожелания, напишите их т
     ут, или оставьте это поле пустым')
}

function generateFormBlock(form, help){
  form.addDateItem().setTitle('Для аттестации "' + help + '" н
     ужно назначить определённую дату?').setHelpText(help);
}
```

```
function clearForm(form){
  while(form.getItems().length!=0){
    try{
      form.deleteItem(0);
    } catch(error)
    {
      Logger.log(error);
    }
  }
}
```

Приложение 11

Многопоточная вариация алгоритма DFS

```
public class ParallelPriorAlgorithm {

  public List<List<Solution>> schedule(List<Event> events1,
     List<DateTimeClass> dtc1,
  List<Solution> externalExams, int maxTimePerDay,
  int solutionsMaxSize, int parallel) {
    List<int[]> solutions = new ArrayList<>();
    List<Event> events;
    List<Solution> external;
    List<DateTimeClass> dtc;
    Map<Event, DateTimeClass> externalMap = new HashMap<>();
    List<DateTimeClass> externalDtc = new ArrayList<>();
    int[] solution;
    int eventSize;
    int dtcSize;
    boolean[] ignoreWishes;
    events = new ArrayList<>();
    dtc = new ArrayList<>();

    events.addAll(events1);
    Collections.sort(events);
    Set<DateTimeClass> dtcSet = new HashSet<>(dtc1);
    //сначала те события, для которых определено время
    if (externalExams != null && externalExams.size() > 0) {
      external = externalExams;
      for (Solution sol : external) {
        Event e = sol.event;
        // System.out.println("ext: " + e);
        externalMap.put(e, sol.dtc);
        for (int i = 0; i < e.type.duration; i++) {
          DateTimeClass dtc2 = new DateTimeClass(sol.dtc.date,
              sol.dtc.time + i, sol.dtc.classroom);
          externalDtc.add(dtc2);
        }
        int i = events.size() - 1;
        while (i >= 0) {
          Event ev = events.get(i);
          if (e.course.equals(ev.course) && e.group.name.
             equals(ev.group.name)) {
            events.remove(ev);
```

```
            break;
          }
          i--;
        }
      }
      dtcSet.addAll(externalDtc);
    }
    Collections.sort(events);
    events.addAll(0, externalMap.keySet());
    dtc.addAll(dtcSet);
    Collections.sort(dtc);
    eventSize = events.size();
    dtcSize = dtc.size();
    solution = new int[dtcSize];
    for (int i = 0; i < dtcSize; i++) {
      solution[i] = -1;
    }
    for (Event event : externalMap.keySet()) {
      for (int i = 0; i < dtcSize; i++) {
        if (dtc.get(i).equals(externalMap.get(event))) {
          submitDateTimeClass(events, solution, events.indexOf
             (event), i);
          break;
        }
      }
    }
    ignoreWishes = new boolean[eventSize];
    ScheduleThread[] scheduleThreads = new ScheduleThread[
       parallel];
    Thread[] threads = new Thread[parallel];
    for (int i = 0; i < parallel; i++) {
      scheduleThreads[i] = new ScheduleThread(new ArrayList<>(
         events),
      new ArrayList<>(dtc), new HashMap<>(externalMap),
      new ArrayList<>(solutions), maxTimePerDay,
         solutionsMaxSize,
      solution.clone(), eventSize, dtcSize, ignoreWishes.clone
         (), (dtcSize * i / parallel));
      threads[i] = new Thread(scheduleThreads[i], "Thread " +
         (dtcSize * i / parallel));
      threads[i].start();
    }
    while (true) {
      int alive = parallel;
      for (int i = parallel - 1; i >= 0; i--) {
```

```
        if (!threads[i].isAlive()) {
          alive--;
          if (!scheduleThreads[i].solutions.isEmpty()) {
            solutions = scheduleThreads[i].solutions;
          }
        }
      }
      if (alive == 0 || !solutions.isEmpty()) {
        for (int j = 0; j < parallel; j++) {
          threads[j].interrupt();
        }
        break;
      }
    }
    return getSolutions(new ArrayList<>(solutions), events,
       dtc);
  }

  // бронинирование за событием помещения и времени
  private int[] submitDateTimeClass(List<Event> events, int[]
     solution, int event, int nextTime) {
    int duration = events.get(event).type.duration;
    for (int i = nextTime; i < nextTime + duration; i++) {
      solution[i] = event;
    }
    return solution;
  }

  private List<List<Solution>> getSolutions(List<int[]>
     solutions, List<Event> events, List<DateTimeClass> dtc) {
    if (solutions.size() == 0) return new ArrayList<>();
    List<List<Solution>> res = new ArrayList<>();
    for (int[] ints : solutions) {
      List<Solution> sh = new ArrayList<>();
      for (int j = 0; j < ints.length; j++) {
        if (ints[j] != -1) {
          sh.add(new Solution(events.get(ints[j]), dtc.get(j))
             );
        }
      }
      res.add(sh);
    }
    return res;
  }
}
```

```
120 class ScheduleThread implements Runnable {
    public List<int[]> solutions;
    List<Event> events;
    List<DateTimeClass> dtc;
    Map<Event, DateTimeClass> externalMap;
125 int[] solution;
    int eventSize;
    int dtcSize;
    int start;
    int maxTimePerDay;
130 int solutionsMaxSize;
    boolean[] ignoreWishes;

    public ScheduleThread(List<Event> events,
    List<DateTimeClass> dtc,
135 Map<Event, DateTimeClass> externalMap,
    List<int[]> solutions,
    int maxTimePerDay,
    int solutionsMaxSize,
    int[] solution,
140 int eventSize,
    int dtcSize,
    boolean[] ignoreWishes,
    int start) {
      this.solutions = solutions;
145   this.events = events;
      this.dtc = dtc;
      this.start = start;
      this.externalMap = externalMap;
      this.solution = solution;
150   this.eventSize = eventSize;
      this.dtcSize = dtcSize;
      this.maxTimePerDay = maxTimePerDay;
      this.solutionsMaxSize = solutionsMaxSize;
      this.ignoreWishes = ignoreWishes;
155 }

    @Override
    public void run() {
      int externalSize = externalMap.keySet().size();
160   int event = externalSize;
      if (externalSize == eventSize) {
        solutions.add(solution);
        return;
```

```
    }
    boolean f = false;
    while (event < eventSize) {
      int time = start;
      if (f) {
        time = getTime(event, dtcSize, solution);
      }
      f = true;
      solution = cleanEvent(event, dtcSize, solution);
      int nextTime = findNextStartTime(event, time, events,
         dtc, solution, dtcSize, maxTimePerDay, ignoreWishes);
      //переходим в режим игнорирования пожеланий преподавател
         я, если не смогли найти идеального решения
      if (!ignoreWishes[event] && nextTime == -1) {
        ignoreWishes[event] = true;
        nextTime = findNextStartTime(event, 0, events, dtc,
           solution, dtcSize, maxTimePerDay, ignoreWishes);
      }
      if (nextTime == -1) { //если не удалось найти другого по
         дходящего DateTimeClass для этого события
        if (event == externalSize) {
          // System.out.println("END FIND SOL :(");
          return; //если речь о первом событии, то уже были пе
             ребраны все остальные варинты даже без учёта поже
             ланий преподавателя
        } else {
          ignoreWishes[event] = false;
          event--; // иначе возвращаемся к предыдущему событию
              и пробуем изменить для него DateTimeClass
        }
      } else {
        solution = submitDateTimeClass(events, solution, event
           , nextTime); //бронируем за событием время и аудито
           рию
        event++; // переходим к следующему событию
      }
      // если решение найдено, добавляем его в список решений
         и продолжаем искать решения
      if (event == eventSize) {
        solutions.add(solution.clone());
        if (solutions.size() >= solutionsMaxSize) return;
        event--;
      }
    }
    return;
```

```
  }

  private int[] cleanEvent(int event, int dtcSize, int[]
     solution) {
    for (int i = 0; i < dtcSize; i++) {
      if (solution[i] == event) solution[i] = -1;
    }
    return solution;
  }

  // бронинирование за событием помещения и времени
  private int[] submitDateTimeClass(List<Event> events, int[]
     solution, int event, int nextTime) {
    int duration = events.get(event).type.duration;
    for (int i = nextTime; i < nextTime + duration; i++) {
      solution[i] = event;
    }
    return solution;
  }

  // возвращает следующий свободный индекс DateTimeClass для у
     казанного события
  private int getTime(int event, int dtcSize, int[] solution)
     {
    int min = -1;
    boolean contains = false;
    for (int i = 0; i < dtcSize - 1; i++) {
      // если аудитория в это время ещё никем не забронирована
      if (solution[i] == -1) {
        if (contains) {
          return i;
        } else if (min == -1) {
          min = i;
        }
      }
      // если это событие уже бронировало какое-то время и мес
         то
      if (solution[i] == event) {
        if (contains) return i;
        contains = true;
      }
    }
    // если последняя ячейка была занята текущим событием, то
       следующей для него нет
```

```
    if (contains) return -1;
    return min;
}

private int findNextStartTime(int event, int time, List<
    Event> events, List<DateTimeClass> dtc,
int[] solution, int dtcSize, int maxTimePerDay, boolean[]
    ignoreWishes) {
    if (time < 0) return -1;
    Event ev = events.get(event);
    StudyGroup group = ev.group;
    Set<Teacher> teachers = ev.teacher;
    int duration = ev.type.duration - 1;
    LocalDate dateGroupEvent = null;
    Integer timeGroupEvent = null;
    if (ev.eventGroup != -1 && event != 0) {
      for (int i = 0; i < solution.length; i++) {
        if (solution[i] != -1 && solution[i] != event
        && events.get(solution[i]).eventGroup.equals(ev.
            eventGroup)) {
          if (!(ev.group.name.equals(events.get(solution[i]).
              group.name) && Math.min(ev.type.countPerDay,
              events.get(solution[i]).type.countPerDay) < 2)) {
            dateGroupEvent = dtc.get(i).date;
          }
          if (ev.teacher.size() > 1) {
            timeGroupEvent = dtc.get(i).time;
          }
          break;
        }
      }
    }
    for (int i = time; i < dtcSize; i++) {
      DateTimeClass location = dtc.get(i);
      // если для события установлено, что оно должно проводит
          ься в один день с другим
      if ((dateGroupEvent != null && !dateGroupEvent.equals(
          location.date))
      ) {
        continue;
      }
      boolean teacherWishes = teachers.stream().mapToInt(
          teacher -> {
        if (
```

```
        teacher.date.stream().anyMatch(dt -> dt.equals(
            location.date))
275     && teacher.time.stream().anyMatch(t -> t.equals(
            location.time))
        && teacher.time.stream().anyMatch(t -> t.equals(
            location.time + duration))) return 0;
        else return 1;
      }).sum() == 0;
      // в режиме игнорирования учитываем только события НЕ уд
         овлетворяющие пожелания преподавателя
280   if (ignoreWishes[event]) {
        teacherWishes = !teacherWishes;
      }

      boolean classRoom;
285   if (ev.link != null && !ev.link.equals("")) {
        classRoom = location.classroom.num.equals(ev.link);
      } else if (ev.wishedClassroomNums != null && !ev.
         wishedClassroomNums.equals("")) {
        classRoom = ev.wishedClassroomNums.contains(location.
            classroom.num);
      } else {
290     classRoom = ev.wishedClassroomType <= location.
            classroom.type;
      }
      if (classRoom //есть ли в аудитории нужное оборудование
      && teacherWishes // подходит ли дата и время преподавате
         лю
      && solution[i] == -1 // аудитория в это время свободна
295   //&& group.size <= location.classroom.size //влезает ли
         группа в аудиторию
      // не слишком ли сейчас поздно для начала проведения дли
         тельного события
      && i + duration < dtcSize
      && location.classroom.num.equals(dtc.get(i + duration).
         classroom.num)
      && location.time + duration == dtc.get(i + duration).
         time // в массиве времени нет разрывов
300   && location.date.equals(dtc.get(i + duration).date)
      ) {
        Map<String, Integer> teacherTime = new HashMap<>();
        int k = 0;
        int countPerDay = 0;
305     while (k < dtcSize) {
```

```
            if (k == 0 || !dtc.get(k).date.equals(dtc.get(k - 1)
               .date)) {
              countPerDay = 0;
            }
            // если время-место свободны, они не повлияют на огр
               аничения
310         if (solution[k] == -1) {
              k++;
              continue;
            }
            Event currentEvent = events.get(solution[k]);
315         DateTimeClass currentLocation = dtc.get(k);
            // если проверяемая аудитория в это время занята
            if (location.date.equals(currentLocation.date) &&
               currentLocation.time >= location.time &&
               currentLocation.time < location.time + ev.type.
               duration && location.classroom.num.equals(
               currentLocation.classroom.num)) {
              break;
            }
320         boolean success = false;
            // если у преподавателя в этот день уже были занятия
            if (currentLocation.date.equals(location.date)
            && !(dateGroupEvent != null && ev.eventGroup.equals(
               currentEvent.eventGroup) && (ev.teacher.size() >
               1))
            ) {
325           for (Teacher curT : currentEvent.teacher) {
                for (Teacher teacher : teachers) {
                  if (curT.name.equals(teacher.name)) {
                    if (!teacherTime.containsKey(teacher.name))
                       {
                      teacherTime.put(teacher.name, 1);
330                 } else {
                      teacherTime.compute(teacher.name, (key, v)
                          -> v + 1);
                    }
                    // если преподаватель в это время ведёт друг
                       ую аттестацию
                    if (currentLocation.time >= location.time
335                 && currentLocation.time <= location.time +
                       duration) {
                      success = true;
                      break;
                    }
```

```
                    // если преподаватель уже достаточно отработ
                       ал в этот день.
                    if (teacherTime.get(teacher.name) >
                       maxTimePerDay - duration) {
                      success = true;
                      break;
                    }
                  }
                }
                if (success) break;
              }
              if (success) {
                break;
              }
            }
            if (currentEvent.group.name.equals(group.name)) {
              //если у группы занятие прямо в этот момент
              if (location.date.equals(currentLocation.date) &&
                 location.time == currentLocation.time) {
                break;
              }
              //если в этот день уже были занятия у этой группы
              if (location.date.equals(currentLocation.date)) {
                countPerDay++;
                if (Math.min(ev.type.countPerDay * ev.type.
                   duration, currentEvent.type.countPerDay *
                   currentEvent.type.duration) > countPerDay) {
                  break;
                }
              }
              //если до или после события есть другое событие, д
                 о которого меньше "отдыха", чем нужно
              if (location.date.minusDays(Math.max(ev.type.
                 pauseBefore, currentEvent.type.pauseAfter)).
                 compareTo(currentLocation.date) < 0
              && location.date.plusDays(Math.max(ev.type.
                 pauseAfter, currentEvent.type.pauseBefore)).
                 compareTo(currentLocation.date) > 0) {
                break;
              }
            }
            k++;
          }
          if (k == dtcSize) {
            return i;
```

```
            }
375       }
        }
        return -1;
    }
}
```

Приложение 12

Замеры времени и памяти

```
public class TimeTest {
  private List<DateTimeClass> generateDTC(int datecount, int
     classcount) {
    LocalDate d1 = LocalDate.parse("2021-01-01");
    List<LocalDate> dateList = new ArrayList<>();
    for (int i = 0; i < datecount; i++) {
      dateList.add(d1.plusDays(i));
    }
    List<Integer> time = List.of(8, 9, 10, 11, 12, 13, 14, 15,
        16, 17);
    List<Classroom> classrooms = new ArrayList<>();
    for (int i = 0; i < classcount; i++) {
      classrooms.add(new Classroom("c" + i, 10, 10));
    }
    return SessionService.createListDTC(dateList, time,
       classrooms);
  }

  private List<Event> generateEvents(int teacherCount, int
     teacherDateCount, int countGroups, int countEventsByGroup
     , int attDuration) {
    List<Teacher> teachers = new ArrayList<>();
    LocalDate d1 = LocalDate.parse("2021-01-01");
    for (int i = 0; i < teacherCount; i++) {
      Teacher teacher = new Teacher("t" + i);
      List<LocalDate> dateList = new ArrayList<>();
      for (int j = 0; j < teacherDateCount; j++) {
        dateList.add(d1.plusDays(j));
      }
      teacher.time = List.of(8, 9, 10, 11, 12, 13, 14, 15, 16,
          17, 18);
      teacher.date = dateList;
      for (int j = 0; j < countEventsByGroup * countGroups /
         teacherCount; j++) {
        teachers.add(teacher);
      }
      Collections.shuffle(teachers);
    }
    List<StudyGroup> groups = new ArrayList<>();
```

```
    for (int i = 0; i < countGroups; i++) {
      groups.add(new StudyGroup("gr" + i, 1, 1));
    }
    AttestationType att = new AttestationType("Экз", 0, 0,
       attDuration, 3);
    List<Event> events = new ArrayList<>();
    for (int i = 0; i < countGroups; i++) {
      for (int j = 0; j < countEventsByGroup; j++) {
        events.add(new Event(groups.get(i), Set.of(teachers.
           get(countEventsByGroup * i + j)), att, "course" + i
            * countEventsByGroup + j));
      }
    }
    return events;
  }

  private long[] getTime(List<DateTimeClass> dtc, List<Event>
     events) {
    int n = 10;
    long time = 0;
    long mem = 0;
    long start = System.nanoTime();
    for (int i = 0; i < n; i++) {
      SchedulePriorAlgorithm algorithm = new
         SchedulePriorAlgorithm(events, dtc, new ArrayList<>()
         , 8, 1000);
      algorithm.findSolutions();
      long finish = System.nanoTime();
      time += finish - start;
      System.gc();
      mem += (Runtime.getRuntime().totalMemory() - Runtime.
         getRuntime().freeMemory());
    }
    return new long[]{(time / n) / 1000000, mem/n};
  }
  private long[] getParallelTime(List<DateTimeClass> dtc, List
     <Event> events) {
    int n = 10;
    long time = 0;
    long mem = 0;
    long start = System.nanoTime();
    for (int i = 0; i < n; i++) {
      ParallelPriorAlgorithm algorithm = new
         ParallelPriorAlgorithm();
```

```
70        algorithm.schedule(events, dtc, new ArrayList<>(), 8,
              1000, 2);
          long finish = System.nanoTime();
          time += finish - start;
          System.gc();
          mem += (Runtime.getRuntime().totalMemory() - Runtime.
              getRuntime().freeMemory());
75      }
        return new long[]{(time / n) / 1000000, mem/n};
    }

    private long[] getTimeDynamic(List<DateTimeClass> dtc, List<
        Event> events) {
80      int n = 10;
        long time = 0;
        long mem = 0;
        long start = System.nanoTime();
        for (int i = 0; i < n; i++) {
85        DynamicScheduleAlgorithm algorithm = new
              DynamicScheduleAlgorithm(events, dtc, 8, 1);
          algorithm.findSolutions();
          algorithm.getSolutions();
          long finish = System.nanoTime();
          time += finish - start;
90        System.gc();
          mem += (Runtime.getRuntime().totalMemory() - Runtime.
              getRuntime().freeMemory());

        }
        return new long[]{(time / n) / 1000000, mem/n};
95  }

    private List<Event> resize(List<Event> events) {
        List<Event> res = new ArrayList<>();
        res.addAll(events);
100     for (int i = 0; i < events.size(); i++) {
          Set<Teacher> set = new HashSet<>();
          events.get(i).teacher.forEach(x -> {
            Teacher t = new Teacher(x.name + "c");
            t.time = x.time;
105         t.date = x.date;
            set.add(t);
          });
          res.add(new Event(new StudyGroup(events.get(i).group.
              name + "c", 10, 1),
```

```
      set, events.get(i).type, events.get(i).course + "c"));
    }
    return res;
  }

  @Test
  public void time1() {
    long[][] res = new long[20][4];
    List<Event> events = generateEvents(5, 50, 5, 3, 2);
    List<List<Event>> e = new ArrayList<>();
    List<Event> events1 = resize(events);
    List<Event> events2 = resize(events1);
    List<Event> events3 = resize(events2);
    e.add(events);
    e.add(events1);
    e.add(events2);
    e.add(events3);
    List<List<DateTimeClass>> d = new ArrayList<>();
    d.add(generateDTC(10, 5));
    d.add(generateDTC(20, 10));
    d.add(generateDTC(30, 15));
    d.add(generateDTC(50, 25));
    d.add(generateDTC(50, 50));
    for (int i = 0; i < d.size(); i++) {
      for (int j = 0; j < e.size(); j++) {
        long[] t = getTime(d.get(i), e.get(j));
        res[i][j] = t[0];
        res[i+5][j] = t[1];
      }
    }
    System.gc();
    for (int i = 0; i < d.size(); i++) {
      for (int j = 0; j < e.size(); j++) {
        long[] t = getParallelTime(d.get(i), e.get(j));
        res[i+10][j] = t[0];
        res[i+15][j] = t[1];
      }
    }
    System.out.println(googleRes(res));
  }

  @Test
  public void time2() {
    long[][] res = new long[20][4];
    List<Event> events = generateEvents(1, 15, 1, 1, 2);
```

```
    List<List<Event>> e = new ArrayList<>();
    List<Event> events1 = resize(events);
    List<Event> events2 = resize(events1);
    List<Event> events3 = resize(events2);
    e.add(events);
    e.add(events1);
    e.add(events2);
    e.add(events3);

    List<List<DateTimeClass>> d = new ArrayList<>();
    d.add(generateDTC(3, 3));
    d.add(generateDTC(4, 4));
    d.add(generateDTC(5, 5));
    d.add(generateDTC(6, 6));
    d.add(generateDTC(7, 7));
    for (int i = 0; i < d.size(); i++) {
      for (int j = 0; j < e.size(); j++) {
        long[] t = getTimeDynamic(d.get(i), e.get(j));
        res[i+10][j] = t[0];
        res[i+15][j] = t[1];
      }
    }
    System.gc();
    for (int i = 0; i < d.size(); i++) {
      for (int j = 0; j < e.size(); j++) {
        long[] t = getTime(d.get(i), e.get(j));
        res[i][j] = t[0];
        res[i+5][j] = t[1];
      }
    }
    System.out.println(googleRes(res));
  }

  private static final String APPLICATION_NAME = "Scheduler";
  private static final JsonFactory JSON_FACTORY =
     JacksonFactory.getDefaultInstance();
  private static final String REFRESH_TOKEN_PATH = "/
     refresh_token.txt";
  private static final String CREDENTIALS_FILE_PATH = "/
     credentials.json";
  private static String REFRESH_TOKEN;

  static {
    try {
```

```
      REFRESH_TOKEN = new BufferedReader(new InputStreamReader
        (GoogleFormService.class.getResourceAsStream(
        REFRESH_TOKEN_PATH))).readLine();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  public static String googleRes(long[][] res) {
    try {
      final NetHttpTransport HTTP_TRANSPORT =
        GoogleNetHttpTransport.newTrustedTransport();
      Sheets sheetsService = new Sheets.Builder(HTTP_TRANSPORT
        , JSON_FACTORY, getCredentials(HTTP_TRANSPORT))
      .setApplicationName(APPLICATION_NAME)
      .build();

      Spreadsheet spreadSheet = sheetsService.spreadsheets()
      .get("11hYJlt4qiWWEKrj8AciEeMSbxH8ZSy8wgyEQWfrpNTk")
      .execute();
      List<List<Object>> list = new ArrayList<>();


      for (int i = 0; i < 20; i++) {
        List<Object> l = new ArrayList<>();
        for (int j = 0; j < 4; j++) {
          l.add(String.valueOf(res[i][j]));
        }
        list.add(l);
      }

      ValueRange body = new ValueRange().setValues(list);
      sheetsService.spreadsheets().values()
      .clear(spreadSheet.getSpreadsheetId(), "A1:H", new
        ClearValuesRequest())
      .execute();
      sheetsService.spreadsheets().values()
      .update(spreadSheet.getSpreadsheetId(), "A1:H", body)
      .setValueInputOption("RAW")
      .execute();
      return spreadSheet.getSpreadsheetUrl();

    } catch (Exception e) {
      System.out.println(e);
      return null;
```

```
235     }
    }

    public static void main(String[] args) {
      long[][] res = new long[5][4];
240   for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 4; j++) {
          res[i][j] = i * j;
        }
      }
245   googleRes(res);
    }

    public static Credential getCredentials(final
       NetHttpTransport HTTP_TRANSPORT) throws IOException {
      InputStream in = GoogleFormService.class.
         getResourceAsStream(CREDENTIALS_FILE_PATH);
250   if (in == null) {
        throw new FileNotFoundException("Resource not found: " +
            CREDENTIALS_FILE_PATH);
      }
      GoogleClientSecrets clientSecrets = GoogleClientSecrets.
         load(JSON_FACTORY, new InputStreamReader(in));
      String clientId = clientSecrets.getDetails().getClientId()
         ;
255   String clientSecret = clientSecrets.getDetails().
         getClientSecret();
      GoogleCredential credential = new GoogleCredential.Builder
         ()
      .setTransport(HTTP_TRANSPORT)
      .setJsonFactory(JSON_FACTORY)
      .setClientSecrets(clientId, clientSecret)
260   .build();
      credential.setAccessToken(getNewToken(REFRESH_TOKEN,
         clientId, clientSecret));
      credential.setRefreshToken(REFRESH_TOKEN);
      return credential;
    }
265
    public static String getNewToken(String refreshToken, String
        clientId, String clientSecret) throws
    IOException {
      ArrayList<String> scopes = new ArrayList<>();
      scopes.add(SheetsScopes.SPREADSHEETS);
```

```
270     TokenResponse tokenResponse = new
           GoogleRefreshTokenRequest(new NetHttpTransport(), new
           JacksonFactory(),
        refreshToken, clientId, clientSecret).setScopes(scopes).
           setGrantType("refresh_token").execute();
        return tokenResponse.getAccessToken();
      }
    }
```

Приложение 13

Алгоритм поиска в ширину

```
public class DynamicScheduleAlgorithm {
  List<Event> events;
  List<DateTimeClass> dtc;
  List<List<Integer>>[][] solutionsMatrix;
  List<List<Integer>> solutions;
  int eventSize;
  int dtcSize;
  int maxTimePerDay;
  int maxSolutionSize;

  public DynamicScheduleAlgorithm(List<Event> events, List<
     DateTimeClass> dtc, int maxTimePerDay, int
     maxSolutionSize) {
    this.events = events;
    this.dtc = dtc;
    Collections.sort(this.dtc);
    this.eventSize = events.size();
    this.dtcSize = dtc.size();
    this.maxTimePerDay = maxTimePerDay;
    this.maxSolutionSize = maxSolutionSize;
    this.solutions = new ArrayList<>();
    this.solutionsMatrix = new ArrayList[eventSize][dtcSize];
    System.out.println("dtcSize " + dtcSize + " eventSize " +
       eventSize);
  }

  public List<List<Integer>> findSolutions() {
    //заполнение матрицы решений
    for (int i = 0; i < dtcSize; i++) {
      if (simpleCheck(0, i)) {
        solutionsMatrix[0][i] = new ArrayList<>();
        solutionsMatrix[0][i].add(Collections.singletonList(i)
           );
      }
    }
    for (int i = 1; i < eventSize; i++) {
      int nullCount = 0;
      if (i > 1) {
        for (int j = 0; j < dtcSize; j++) {
          solutionsMatrix[i - 2] = null;
```

```
        }
      }
      System.gc();
      for (int j = 0; j < dtcSize; j++) {
        // если сама аудитория не подходит для проведения атте
           стации, нет смысла проверять её загруженность
        if (simpleCheck(i, j)) {
          for (int k = 0; k < dtcSize; k++) {
            if (solutionsMatrix[i - 1][k] != null) {
              for (List<Integer> path : solutionsMatrix[i -
                 1][k]) {
                if (pathsCheck(i, j, path)) {
                  if (solutionsMatrix[i][j] == null) {
                    solutionsMatrix[i][j] = new ArrayList<>();
                  }
                  List<Integer> newPath = new ArrayList<>(path
                     );
                  newPath.add(j);
                  solutionsMatrix[i][j].add(newPath);
                }
              }
            }
          }
        }
        /* Если во время попытки разместить какую-то аттестаци
           ю выясняется, что её не возможно поставить,
        сразу возвращаем null, чтобы не обходить зря следующие
            аттестации
        */
        if (solutionsMatrix[i][j] == null) {
          nullCount++;
        }
        if (nullCount == dtcSize) {
          System.out.println("NULL" + dtcSize);
          return null;
        }
      }
    }
    // собираем все возможные решения расстановки всех событий
        в один массив
    solutions = new ArrayList<>();
    for (int i = 0; i < dtcSize; i++) {
      if (solutionsMatrix[eventSize - 1][i] != null) {
        if (solutions.size() < maxSolutionSize)
        solutions.addAll(solutionsMatrix[eventSize - 1][i]);
```

```
      }
    }
    return solutions;
  }

  public List<Solution> getSolutions() {
    List<List<Solution>> res = new ArrayList<>();
    for (int i = 0; i < solutions.size(); i++) {
      List<Solution> sh = new ArrayList<>();
      for (int j = 0; j < events.size(); j++) {
        sh.add(new Solution(events.get(j), dtc.get(solutions.
            get(i).get(j))));
      }
      res.add(sh);
    }
    return res.get(0);
  }

  // проверка условий, которые завязаны на уже существующем ра
     списании
  private boolean pathsCheck(int i, int j, List<Integer> path)
      {
    Event ev = events.get(i);
    DateTimeClass dateTimeClass = dtc.get(j);
    Set<Teacher> teachers = ev.teacher;
    StudyGroup group = ev.group;
    LocalDate pauseStart = dateTimeClass.date.minusDays(ev.
       type.pauseBefore);
    LocalDate pauseEnd = dateTimeClass.date.plusDays(ev.type.
       pauseAfter);
    int duration = ev.type.duration - 1;
    int groupsEventsToday = 1;
    for (int k = 0; k < path.size(); k++) {
      DateTimeClass d = dtc.get(path.get(k));
      Event e = events.get(k);
      if (d.date == dateTimeClass.date) {
        if (d.time >= dateTimeClass.time && d.time <=
            dateTimeClass.time + duration) {
          for (Teacher t : teachers) {
            for (Teacher t2 : e.teacher) {
              if (t.name.equals(t2.name)) {
                return false;
              }
            }
          }
```

```
        }
        if (group.name.equals(e.group.name)) {
          groupsEventsToday++;
        }
      }
      if (
      //если в этот день есть аттестации, которые по времени н
         акладываются с текущей
      (d.date.equals(dateTimeClass.date) && (d.time + e.type.
         duration > dateTimeClass.time) && (dateTimeClass.time
          + duration > d.time))
      // если одна из предыдущих аттестаций попадает в окно от
         дыха до и после текущей аттестации
      || (d.date.compareTo(pauseStart) > 0) && (d.date.
         compareTo(pauseEnd) < 0)
      // если у группы больше аттестаций этого типа, чем полож
         ено
      || (groupsEventsToday > ev.type.countPerDay)
      ) {
        return false;
      }
    }
    return true;
  }

  // проверка условий, которые не завязаны на уже существующем
      расписании
  private boolean simpleCheck(int i, int j) {
    Event ev = events.get(i);
    DateTimeClass location = dtc.get(j);
    StudyGroup group = ev.group;
    Set<Teacher> teachers = ev.teacher;
    int duration = ev.type.duration - 1;
    boolean teacherWishes = teachers.stream().mapToInt(teacher
        -> {
      if (
      teacher.date.stream().anyMatch(dt -> dt.equals(location.
         date))
      && teacher.time.stream().anyMatch(t -> t.equals(location
         .time))
      && teacher.time.stream().anyMatch(t -> t.equals(location
         .time + duration))) return 0;
      else return 1;
    }).sum() == 0;
    boolean classRoom;
```

```
150     if (ev.link != null && !ev.link.equals("")) {
          classRoom = location.classroom.num.equals(ev.link);
        } else if (ev.wishedClassroomNums != null && !ev.
           wishedClassroomNums.equals("")) {
          classRoom = ev.wishedClassroomNums.contains(location.
             classroom.num);
        } else {
155       classRoom = ev.wishedClassroomType <= location.classroom
             .type;
        }
        boolean res = classRoom //есть ли в аудитории нужное обору
           дование
        && teacherWishes // подходит ли дата и время преподавателю
        && group.size <= location.classroom.size //влезает ли груп
           па в аудиторию
160     // не слишком ли сейчас поздно для начала проведения длите
           льного события
        && i + duration < dtcSize
        && location.classroom.num.equals(dtc.get(i + duration).
           classroom.num)
        && location.time + duration == dtc.get(i + duration).time
           // в массиве времени нет разрывов
        && dtc.get(i).date.equals(dtc.get(i + duration).date);
165     return res;
      }
    }
```