# РЕФЕРАТ

Дипломный проект 70 стр., 22 рис., 12 источников, 1 приложение, 6 таблиц.

ГЕНЕРАЦИЯ РАСПИСАНИЯ УРОКОВ, ГЕНЕРАТОР РАСПИВАНИЯ, РЕЛЯЦИОННАЯ БАЗА ДАННЫХ, ТЕСТИРОВАНИЕ, MVC.

Цель данного дипломного проекта – создание информационной системы генерации расписания уроков, которое облегчит преподавательскому составу составлять расписание уроков, а также повысит их производительность.

Данный продукт найдет применение в руках учителей школ, преподавателей университетов и любого другого сотрудника, занимающегося составлением учебных расписаний.

За основу проекта был взят фреймворк Spring Boot. Он упрощает разработку веб-приложений. В качестве архитектуры программного продукта была выбрана архитектура MVC.

# СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc79341429)

[1 АНАЛИЗ ОБЪЕКТА 7](#_Toc79341430)

[1.1 Описание предметной области 7](#_Toc79341431)

[1.2 Построение концептуальной модели предметной области 11](#_Toc79341432)

[2 ПОСТАНОВКА ЗАДАЧИ 14](#_Toc79341433)

[2.1 Определение требований к программной системе 14](#_Toc79341434)

[2.2 Описание аналогов системы 14](#_Toc79341435)

[2.3 Обзор и обоснование выбора инструментальных средств 19](#_Toc79341436)

[3 ПРОЕКТИРОВАНИЕ 24](#_Toc79341437)

[3.1 Разработка архитектуры программного продукта 24](#_Toc79341438)

[3.2 Проектирование структур хранения данных 27](#_Toc79341439)

[3.3 Описание реализации вариантов использования 29](#_Toc79341440)

[4 РЕАЛИЗАЦИЯ 31](#_Toc79341441)

[4.1 Разработка классов информационной системы 31](#_Toc79341442)

[4.2 Разработка интерфейса программного продукта 38](#_Toc79341443)

[4.3 Разработка алгоритмов реализации вариантов использования 47](#_Toc79341444)

[4.4 Модульное тестирование алгоритмов реализации вариантов использования 49](#_Toc79341445)

[5 СИСТЕМНОЕ ТЕСТИРОВАНИЕ 52](#_Toc79341446)

[5.1 Функциональное тестирование 52](#_Toc79341447)

[5.2 Оценка безопасности 55](#_Toc79341448)

[ЗАКЛЮЧЕНИЕ 58](#_Toc79341449)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 60](#_Toc79341450)

[ПРИЛОЖЕНИЕ А. Исходный код программы 62](#_Toc79341451)

# ВВЕДЕНИЕ

Качество подготовки в школах вузах и особенно эффективность использования научно-педагогического потенциала зависят в определенной степени от уровня организации учебного процесса. Одна из основных составляющих этого процесса расписание занятий регламентирует трудовой ритм, влияет на творческую отдачу преподавателей, поэтому его можно рассматривать как фактор оптимизации использования ограниченных трудовых ресурсов преподавательского состава.

Объектом автоматизации являются информационные процессы школ, связанные с составлением расписания в школе.

Предметом автоматизации является автоматизированная система составления расписания в школе.

Целью работы является повышение эффективности и снижение трудозатрат персонала школы посредством автоматизации информационных процессов составления расписания в школе.

Актуальность работы заключается в необходимости проектирования, разработки и внедрения автоматизированной системы составления расписания в школе, способной оказывать качественную поддержку для успешной реализации описанного выше информационного процесса. Необходимо минимизировать ручной ввод и автоматизировать обработку информации по составлению расписания в школе, тем самым снизив трудозатраты специалиста и повысить эффективность его работы в целом.

Проблематика данной работы заключается в отсутствии качественных бесплатных автоматизированных инструментов по составлению расписания в школе наряду с повышением нагрузки на персонал школы, снижением финансирования учебных учреждений и обострением экономической ситуации в целом

Исходя из цели, в дипломном проекте поставлены следующие задачи:

* Проанализировать предметную область
* Определить требования к программе
* Выбрать средства реализации приложения
* Разработать программное средство

# 1 АНАЛИЗ ОБЪЕКТА

## 1.1 Описание предметной области

Учебный режим школы должен соответствовать функциональным возможностям учащихся. Объем, содержание и организация учебного процесса должны обеспечивать такое состояние организма, при котором утомление полностью исчезало бы за период отдыха.

Основные критерии оценки уроков с точки зрения функциональных возможностей учащихся – трудность и утомительность. Утомительность характеризуется изменением работоспособности, а трудность предмета – уровнем успеваемости, то есть степенью усвоения учебного материала. Следовательно, при составлении расписания необходимо учитывать оба фактора в равной степени [1].

При составлении расписания уроков должны учитываться следующие факторы:

* работоспособность (характеристика труда, т. е. качество и количество усвоенных знаний или действий в определенное время) зависит от возраста человека, его индивидуального биоритма, времени суток, дня недели, времени года и т. д.;
* степень утомляемости зависит от социальных, психологических, биологических причин.

Чтобы снизить утомляемость учащихся и учителей в течение одного дня, необходимо правильное распределение уроков в продуктивные и непродуктивные часы и дни.

Предметы, требующие больших временных затрат на домашнюю подготовку, не должны группироваться в один день школьного расписания. В наиболее непродуктивные часы (с 11.30 до 14.30) уроки должны строиться с позиций здоровьесбережения, что подразумевает валеологический подход к организации урока, выбор форм, методов, приемов урока, смену видов деятельности учащихся и видов преподавания учителя (через каждые 5–7 мин).

При составлении расписания можно пользоваться шкалами трудности учебных предметов, разработанными И.Г. Сивковым (для начальной школы) и сотрудниками НИИ гигиены и охраны здоровья детей и подростков М.И. Степановой, И.Э. Александровой, А.С. Седовой (для учеников 5–9-х классов)

Используя данные приведенных выше таблиц, можно оценить, правильно ли составлено расписание уроков какого-либо класса.

При выполнении всех рекомендаций можно добиться оптимального распределения нагрузки.

Расписание составлено правильно, если:

* в школе существует единое расписание занятий первой и второй половины дня (для учеников и учителей);
* наибольшее количество баллов за день по сумме всех предметов приходится для 8–11-х классов – на вторник, среду и четверг; для 1–7-х классов – на вторник и четверг (среда – несколько облегченный день);
* в течение учебного дня чередуются уроки “трудные” и “облегченные”;
* уроки одного предмета чередуются с уроками другого предмета, а не сдвоены (исключение – модульная технология обучения);
* основные предметы для младших школьников проводятся на 2–3-х уроках, а для среднего и старшего возраста – на 2–4-х уроках;
* названия учебных предметов в расписании и учебном плане одинаковые.

Расписание составлено неправильно, если:

* наибольшее количество баллов за день приходится на крайние дни недели или когда оно одинаково во все дни недели;
* “трудные” уроки сдвоены;
* “трудные” уроки стоят в расписании подряд;
* “трудные” уроки в расписании на первом или последнем уроке;
* количество домашних подготовок равно количеству уроков.

Для оптимального распределения нагрузки не допускаются:

* нулевые уроки;
* перемены продолжительностью 5 мин;
* сдвоенные уроки в 1–5-х классах, превышающие по шкале трудности 8 баллов, если поурочное планирование не предусматривает вторым часом практическую или лабораторную работу;
* группирование в один день предметов, требующих большой домашней подготовки;
* рассогласование в расписании первой и второй половины дня;
* несоответствие названий учебных предметов в сетке расписания и учебном плане школы.

Таким образом, при составлении расписания занятий необходимо учитывать:

* распределение учебной нагрузки по дням недели;
* распределение учебной нагрузки в течение каждого дня отдельно;
* чередование предметов разных видов деятельности;
* соблюдение гигиенических требований к расписанию уроков.

Для этого расписание анализируется и по итогам составляется аналитическая справка.

Рекомендации по распределению уроков:

1. Суммарная нагрузка в начале и конце недели должна быть наименьшей (по ранговой шкале трудности).
2. Контрольные работы следует проводить на 2–4-х уроках в середине учебной недели.
3. Основная учебная нагрузка в течение дня в старших и средних классах должна приходиться на 2–4-е уроки.
4. В понедельник и пятницу не рекомендуется ставить в расписание двухчасовые уроки.
5. В начальной школе недопустимо проведение сдвоенных уроков по одному и тому же предмету. В этом случае степень утомления детей возрастает в 7 раз.
6. В 5-м классе сдвоенные уроки по одному предмету допустимы только в исключительных случаях и при условии их чередования с более легкими уроками (или уроком).
7. В средних и старших классах сдвоенные уроки на фоне модульной технологии построения учебного процесса, наоборот, способствуют сохранению здоровья учащихся.
8. Уроки, требующие большого умственного напряжения (математика, русский язык, физика, химия), рекомендуется проводить первыми или вторыми. Уроки по искусству, окружающему миру, художественному труду следует проводить после динамической паузы (третий урок), а уроки с преобладанием двигательного компонента (ритмика, физическая культура) – последними.
9. Занятия второй половины дня должны проводиться не ранее чем через 45 мин после окончания последнего урока первой половины дня.
10. При расстановке уроков в течение одного учебного дня следует учитывать преобладающие виды деятельности учащихся на этом уроке (чтение, рассматривание, письмо, активно­двигательная деятельность, практическая работа, слушание, говорение и т. д.) и пытаться чередовать их.
11. Ежедневно количество домашних подготовок должно быть меньше, чем количество уроков в расписании.
12. Не разрешается проводить нулевые уроки!
13. Предметы, требующие больших затрат времени на домашнюю подготовку (история, алгебра), не должны группироваться в один день школьного расписания.
14. Следует помнить о том, что при составлении расписания уроков нужно учитывать количество часов, отведенное базисным учебным планом на выполнение домашних заданий.

## 1.2 Построение концептуальной модели предметной области

***Концептуальное проектирование*** – сбор, анализ и редактирование требований к данным [2]. Для этого осуществляются следующие мероприятия:

– обследование предметной области, изучение ее информационной структуры;

– выявление всех фрагментов, каждый из которых характеризуется пользовательским представлением, информационными объектами, связями между ними и процессами;

– моделирование и интеграция всех представлений.

Концептуальное проектирование начинается с анализа предметной области, включает анализ концептуальных требований и информационных потребностей, выявление информационных объектов и связей между ними, построение концептуальной модели (схемы) данных.

Главными элементами концептуальной модели данных являются объекты и отношения.

Объекты представляют собой любой конкретный (реальный) объект в рассматриваемой области.

Исходя из спецификации требования, определим основные типы сущностей.

*Сущностью* называется некоторая принятая в конкретной постановке задачи абстракция реального мира, процесса или явления, о котором необходимо хранить информацию в системе. В качестве синонима термина «сущность» используется также термин «информационный объект» [3].

Объекты в каждый момент времени характеризуются определенным состоянием, которое описывается набором свойств и отношений (или связей) с другими объектами.

Характеристика, описывающая какое-либо свойство сущности, которое можно сформулировать и записать, называется *атрибутом*. Атрибут, который однозначно определяет сущность, называется идентификатором.

Сущность - объект любой природы данные, о котором хранятся в отношении (таблице, в которой содержатся данные).

Каждый объект предметной области характеризуется некоторым наборов атрибутов, отображающим свойства объекта. Атрибуты используются для определения того, какая информация должна быть собрана об объекте. Примерами атрибутов для объекта «Преподаватель» служат ФИО и т.д.

В рассматриваемой предметной области можно выделить следующие сущности:

1. Класс – содержит информацию о классе, с которым будет проводиться урок (ид, номер класса)
2. Учитель – содержит информацию об учителе, который будет проводить урок (ид, ФИО учителя)
3. Предмет - содержит информацию о предмете, который будет изучаться на уроке (ид, название предмета)
4. Учительские предметы – содержит информацию о том, какие учителя какие предметы ведут (ид учителя, ид предмета)
5. Нагрузка – содержит информацию о часах нагрузки для класса, которую необходимо учитывать при составлении расписания (ид, часы, ид класса, ид предмета)
6. Урок (ид, день, время, ид класса, ид предмета, ид учителя)

Между сущностями возможны четыре типа связей: один - к одному (1 ↔1), один – ко многим (1↔∞), многие к одному (∞↔1), многие ко многим (∞ ↔ ∞ ) [4].

Связь 1 ↔ 1 означает, что в любой момент времени каждому экземпляру первого информационного объекта (ИО) соответствует 1 экземпляр другого ИО.

Связь 1↔ ∞ означает: одному экземпляру ИО соответствует 1,2, … экземпляров другого и, наоборот, каждому экземпляру второго ИО соответствует 1 экземпляр первого ИО. Аналогично определяется тип связи ∞ ↔ 1.

Связь ∞ ↔ ∞ означает, что одному экземпляру первого ИО соответствует 1, 2, … экземпляров другого ИО и наоборот.

Таблица "Класс" имеет отношение один ко многим с таблицей "Урок".

Таблица "Класс " имеет отношение один ко многим с таблицей "Нагрузка".

Таблица "Предмет" имеет отношение один ко многим с таблицей "Нагрузка".

Таблица "Предмет" имеет отношение один ко многим с таблицей "Учительские предметы".

Таблица "Предмет" имеет отношение один ко многим с таблицей "Урок".

Таблица "Учитель" имеет отношение один ко многим с таблицей "Учительские предметы".

Таблица "Учитель" имеет отношение один ко многим с таблицей "Урок".

# 2 ПОСТАНОВКА ЗАДАЧИ

## [2.1 Определение требований к программной системе](#_Toc474748992)

Постановка задачи - один из наиболее важных этапов при создании программы. От того, насколько полно, точно и ясно определены требования к разрабатываемому ПО, его функции и предполагаемые возможности, во многом зависит качество и стоимость разработки [5].

К программной системе определены следующие требования:

1. Запускаться на Windows 7, 8, 10.
2. Поддерживать ввод информации с клавиатуры.
3. Иметь возможность добавлять изменять и удалять учителей
4. Иметь возможность добавлять, изменять и удалять нагрузку
5. Иметь возможность добавлять, изменять и удалять предметы
6. Иметь возможность добавлять, изменять и удалять классы
7. Иметь возможность генерировать расписание уроков на основе введенных данных
8. Иметь дружелюбный интерфейс.

## 2.2 [Описание аналогов системы](#_Toc474748993)

На данный момент времени сектор рынка программного обеспечения систем составления расписания занятий представлен большим количеством различных программных продуктов. Ниже будут описаны наиболее популярные их них.

«**ДЕКАНАТ**»

Возможности системы

* Подготовка и выполнение контроля учебного процесса студентов (формирование учебных планов, подготовка рабочих программ);
* Организация процедуры приема студентов (от приема документов, через проведение вступительных испытаний с конкурсным отбором по их итогам, до формирования представления к зачислению);
* Сопровождение процессов обучения студента в ВУЗ-е (переводы, отчисления, оценки, задолженности и т.д.);
* Обеспечение процесса выпуска студентов (проведение государственных аттестаций, защиты дипломных проектов, подготовку и печать дипломов, академических справок и т.д.);
* Формирование различных документов с поименным составом входящих в них преподавателей (заседаний комиссий, ведомостей и т.д.);
* Анализ различных аспектов учебного процесса (сверка выполнения учебных планов, движение, успеваемость и т.д.)

**«РЕКТОР»**

Ректор 3 - программа для составления расписания в средних школах.

Ректор-Школа - программа для составления расписания в средних школах с профильным обучением.

Программы позволяют быстро и удобно вводить все дисциплины, классы, нагрузки учителей, взаимосвязи уроков, кабинеты. Она позволяет создать группы, объединить несколько классов в один урок и иметь уроки, обучаемые через две недели. Возможно определить и большее количество учителей для одного урока.

Мы можем выбрать точный интервал времени для каждой дисциплины, класса, кабинета и учителя. Например, можно определить, что Математика не может быть 6-ым уроком, учительница Иванова не может обучать 6-ой урок во вторник и 3-ий в среду, и спортзал должен быть свободен в течение 4-ых и 5-ых уроков, так как в эти дни приходят тренироваться ученики спортшколы.

Программа соблюдает условия:

* Число окон для учителей, ограничение максимального числа окон, ограничение дней, когда преподаватель может обучать.
* Равномерное распределение дисциплин в течений недели
* Проверка последовательности уроков полных и разделенных классов
* Размещение уроков в кабинеты

**Ректор-Колледж** - программа для составления расписания в средних специальных учебных заведениях.

**Ректор-ВУЗ** - программа для составления расписания в ВУЗах.

Программа позволяет составить расписание занятий студентов исходя из имеющихся данных о наличии свободных аудиторий, наличие свободных преподавателей, а также учебного планы группы.

Программа позволяет ввести учебный план группы на семестр и далее предоставляет интерактивные интерфейс для ввода расписания занятий, при этом контролируя следующие параметры:

* Соответствие вновь вводимого (редактируемого) занятия учебному плану группы;
* Свободность аудитории;
* Свободность преподавателя.

СИСТЕМНЫЕ ТРЕБОВАНИЯ

Операционная система - MS Windows XP или MS Vista. Объем оперативной памяти - 512 Мб или выше. Свободное место на жестком диске - не менее 100 Мб. Рекомендуемая глубина цвета, поддерживаемая видеокартой и монитором - 16 разрядов (бит) или больше. Минимальное разрешение монитора - 1024 x 768. Мышь - не менее двух кнопок.

**Система АВТОР-2+**

Система АВТОР-2+ предназначена для быстpого и удобного составления расписаний занятий и сопровождения их в течение всего учебного года. АВТОР-2+ - универсальная система. Есть несколько версий программы, рассчитанные на любые учебные заведения:

* сpедние и специализиpованные (математические, языковые и т.п.) школы, лицеи, гимназии;
* техникумы, училища и колледжи;
* ВУЗы с одним учебным корпусом;
* ВУЗы с несколькими учебными корпусами (с учетом переездов между корпусами).

АВТОР-2+ позволяет максимально облегчить и автоматизиpовать сложный тpуд составителей расписания. Система помогает легко стpоить, коppектиpовать и pаспечатывать в виде удобных и наглядных документов:

* pасписания занятий классов (учебных групп);
* расписания пpеподавателей;
* расписание аудиторий (кабинетов);
* учебные планы;
* тарификацию.

АВТОР-2+ имеет пpиятный дизайн и дpужеcтвенный сеpвис. Программа достаточно проста в освоении. Имеется подробное руководство, в котором описаны все возможности и способы работы с программой.

Программа работает на любых IBM-совместимых компьютерах, начиная с 486DX с оперативной памятью 516Mb (и выше), занимает около 75 Mb на жестком диске. Операционная система: WINDOWS XP.

Время работы зависит от размерности учебного заведения и мощности компьютера. Полный расчет и оптимизация расписания школы среднего размера (30 классов, 60 преподавателей, две смены) идет около 5 минут на компьютере типа Celeron-1700

Программа отличается уникальным и очень мощным алгоритмом построения и оптимизации расписания. Полученное автоматическое расписание практически не требует ручной доработки, то есть даже при очень сложных и жестких ограничениях автоматически размещаются все возможные занятия. Если в исходных данных имеются неразрешимые противоречия, то их можно обнаружить и устранить, используя специальный блок анализа. АВТОР-2+ позволяет:

* оптимизировать "окна" в расписании;
* учитывать требуемый диапазон дней/часов как для классов, так и для преподавателей;
* оптимально pазмещать занятия по кабинетам (аудиториям) с учетом особенностей классов, предметов, пpеподавателей и вместимости кабинетов;
* учитывать хаpактеp pаботы и пожелания как штатных сотpудников, так и совместителей-почасовиков;
* легко соединять ("спаpивать") несколько классов (учебных групп) в потоки пpи пpоведении любых занятий;
* pазделять классы пpи пpоведении занятий по иностранному языку, физической культуре, тpуду, информатике (и любым другим предметам) на любое количество подгрупп (до десяти!);
* вводить (помимо основных пpедметов) спецкуpсы и факультативы;
* оптимизировать равномерность и трудоемкость расписания.

По желанию заказчика АВТОР-2+ модифициpуется под условия конкретного учебного заведения.

Проведя анализ аналогов систем делается вывод, что ни одна из них не подходит под наши требования. Многие из них более специализированные и имеют множество лишнего функционала, за который таже будет идти переплата. Кроме того, некоторые имеют слишком высокую стоимость. К примеру, Приложение Автор-2 стоит 4800б.р. В конце концов, все их них требуют использования windows, что означает доступ к приложению только с определенного компьютера.

В связи с вышесказанным делается вывод о необходимости разработки собственного приложения генерации расписания.

## [2.3 Обзор и обоснование выбора инструментальных средств](#_Toc474748994)

Архитектура данного проекта – это веб-приложение. Архитектура приложения во многом предопределило и технологии, используемые для построения проекта.

Основные технологии, применяемые при создании приложения:

– серверный язык JAVA; – язык запросов SQL (MySQL);

– фреймворк Spring; – язык разметки HTML;

– таблица стилей CSS;

– клиентский язык JavaScript.

Каждая из выбранных технологий отвечает за разные аспекты работы программы.

Язык Java появился в 1995 году – 90-е годы были вообще урожайными на новые языки и концепции программирования [6]. В таком Эдеме языков важно было не заблудиться, по ошибке приняв за Священный Грааль технологию, которая не пройдет испытания временем. Java прошел испытания, хотя и очень долгие. Очень не рекомендуется путать этот язык с JavaScript – они по виду похожи, но это совсем разные языки.

Вероятно, в Java впервые реализовали концепцию того, что язык должен быть максимально изолирован от платформы разработки, чтобы применять его без изменений везде: в компьютерах, часах, сотовых телефонах, бытовой технике. С «железной частью» должна была справляться виртуальная машина (JVM), которая, собственно, и создавалась индивидуально под каждое устройство. Сам же язык был неизменен и в качестве результата выдавал байткод.

С самого начала было известно, что код не может исполняться очень быстро, но многие устройства не требовали высокой скорости исполнения. Кроме того, со временем появились оптимизирующие компиляторы, так что, в среднем, программа на Java работает раза в 2-3 медленнее, чем на C++.

Постоянное сравнение с C/C++ здесь не случайно: многие современные языки взяли за основу его конструкции и синтаксис, так что, бывает, узнать сходу язык очень трудно. Вместе с тем, Java с тех пор сильно «размножилась», и даже J#, J и прочие аналоги являются не родными братьями, а лишь подобием.

Сама идея языка, вполне, кстати, достаточного для создания софта любой сложности, была сначала не понята: был ли, мол, смысл создавать между аппаратурой и кодом промежуточные слои исполняющих машин? Со временем сомнения рассеялись: появилась мультиязычная платфома .

NET, и даже в Windows появились слои – аппаратно-зависимые, платформо-независимые. Самое же простое объяснение – софт стал очень сложным, а программисты очень ленивыми, чтобы переписывать программы под каждый отдельный аппарат.

Но вернемся к языку. Как уже говорилось, чем-то он похож на C++, чем-то на старый добрый Бейсик. Нет сейчас ни одного языка, который бы не хвалился своими возможностями ООП, и Java здесь не отличается от канонов: классы и объекты здесь используются везде, даже в самых примитивных задачах вроде вывода строки на экран. Из особенностей можно отметить, что все объекты в языке создаются только динамически, а все функции являются методами классов.

Множественное наследование не поддерживается, как в C++, как и «опасные» указатели. ООП дает много преимуществ, но и требует слишком многого – в случае Java памяти устройства никогда не будет слишком много.

В остальном же, имеются библиотеки классов для практически всех задач; преимущественно – под написание клиентских и серверных приложений. Хозяин Java – Oracle – успешно использует язык для использования в разработках своей одноименной СУБД. На сегодняшний день язык считается наиболее востребованным на рынке.

Spring Framework (или коротко Spring) — универсальный фреймворк с открытым исходным кодом для Java-платформы. Также существует форк для платформы .NET Framework, названный Spring.NET [7].

Первая версия была написана Родом Джонсоном, который впервые опубликовал её вместе с изданием своей книги «Expert One-on-One Java EE Design and Development» (Wrox Press, октябрь 2002 года).

Фреймворк был впервые выпущен под лицензией Apache 2.0 license в июне 2003 года. Первая стабильная версия 1.0 была выпущена в марте 2004. Spring 2.0 был выпущен в октябре 2006, Spring 2.5 — в ноябре 2007, Spring 3.0 в декабре 2009, и Spring 3.1 в декабре 2011. Текущая версия — 5.2.x.

Несмотря на то, что Spring не обеспечивал какую-либо конкретную модель программирования, он стал широко распространённым в Java-сообществе главным образом как альтернатива и замена модели Enterprise JavaBeans. Spring предоставляет бо́льшую свободу Java-разработчикам в проектировании; кроме того, он предоставляет хорошо документированные и лёгкие в использовании средства решения проблем, возникающих при создании приложений корпоративного масштаба.

Между тем, особенности ядра Spring применимы в любом Java-приложении, и существует множество расширений и усовершенствований для построения веб-приложений на Java Enterprise платформе. По этим причинам Spring приобрёл большую популярность и признаётся разработчиками как стратегически важный фреймворк.

Следующие две технологии – это HTML и CSS, которые предопределены архитектурой проекта. Имеется единственное обоснование их выбора – это веб-архитектура разрабатываемого программного средства.

HTML – язык гипертекстовой разметки, который используется для структурирования и отображения веб-страницы и ее контента [8]. Для разработки данного программного средства будет применяться версия этого языка HTML5.

CSS – это формальный язык описания внешнего вида документа, написанного с использованием языка разметки. Преимущественно используется как средство описания, оформления внешнего вида веб-страниц, написанных с помощью языков разметки HTML [9]. Применяемая версия данной технологии – CSS3.

Последняя из основных применяемый технологий – это JavaScript. С помощью данного языка программирования, можно создавать динамически обновляемый контент, управлять мультимедиа, анимировать изображения. Что касается обоснований выбора в качестве клиентского языка JavaScript, то здесь выбор достаточно очевиден, ведь конкурентов у данного языка в данной сфере практически нет.

Лишь некоторые достоинства данного языка: возможность навигации и управления по DOM HTML-страницы, возможность управления браузером, гибкость подхода объектно-ориентированного программирования, поддержка асинхронности.

Для проектирования программного средства используется язык графического описания UML. UML – язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур [10].

UML является языком широкого профиля, это – открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML-моделью.

UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. UML не является языком программирования, на основании UML моделей возможна генерация кода.

# 3 ПРОЕКТИРОВАНИЕ

## 3.1 Разработка архитектуры программного продукта

Фреймворк Spring MVC обеспечивает архитектуру паттерна Model — View — Controller (Модель — Отображение (далее — Вид) — Контроллер) при помощи слабо связанных готовых компонентов. Паттерн MVC разделяет аспекты приложения (логику ввода, бизнес-логику и логику UI), обеспечивая при этом свободную связь между ними [11].

− Model (Модель) инкапсулирует (объединяет) данные приложения, в целом они будут состоять из POJO («Старых добрых Java-объектов», или бинов).

− View (Отображение, Вид) отвечает за отображение данных Модели, — как правило, генерируя HTML, которые мы видим в своём браузере.

− Controller (Контроллер) обрабатывает запрос пользователя, создаёт соответствующую Модель и передаёт её для отображения в Вид.

Вся логика работы Spring MVC построена вокруг DispatcherServlet, который принимает и обрабатывает все HTTP-запросы (из UI) и ответы на них. Рабочий процесс обработки запроса DispatcherServlet'ом проиллюстрирован на следующей диаграмме:

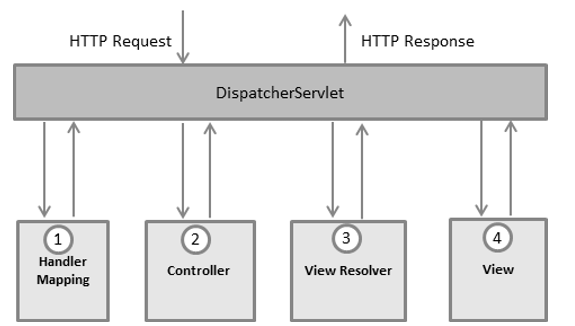


Рисунок 3.1 - Рабочий процесс обработки запроса

Ниже приведена последовательность событий, соответствующая входящему HTTP-запросу:

− После получения HTTP-запроса DispatcherServlet обращается к интерфейсу HandlerMapping, который определяет, какой Контроллер должен быть вызван, после чего, отправляет запрос в нужный Контроллер.

− Контроллер принимает запрос и вызывает соответствующий служебный метод, основанный на GET или POST. Вызванный метод определяет данные Модели, основанные на определённой бизнес-логике и возвращает в DispatcherServlet имя Вида (View).

− При помощи интерфейса ViewResolver DispatcherServlet определяет, какой Вид нужно использовать на основании полученного имени.

− После того, как Вид (View) создан, DispatcherServlet отправляет данные Модели в виде атрибутов в Вид, который в конечном итоге отображается в браузере.

Все вышеупомянутые компоненты, а именно, HandlerMapping, Controller и ViewResolver, являются частями интерфейса WebApplicationContext extends ApplicationContext, с некоторыми дополнительными особенностями, необходимыми для создания web-приложений. DispatcherServlet отправляет запрос контроллерам для выполнения определённых функций.

Аннотация @Controllerannotation указывает, что конкретный класс является контроллером. Аннотация @RequestMapping используется для мапинга (связывания) с URL для всего класса или для конкретного метода обработчика.

Аннотация Controller определяет класс как Контроллер Spring MVC. В первом случае, @RequestMapping указывает, что все методы в данном Контроллере относятся к URL-адресу "/hello".

Следующая аннотация @RequestMapping(method = RequestMethod.GET) используется для объявления метода printHello() как дефолтного метода для обработки HTTP-запросов GET.

Вы можете определить любой другой метод как обработчик всех POST-запросов по данному URL-адресу. Вы можете написать вышеуказанный

Контроллер по-другому, указав дополнительные атрибуты для аннотации @RequestMapping следующим образом @RequestMapping(value = "/hello", method = RequestMethod.GET) Атрибут «value» указывает URL, с которым мы связываем данный метод (value = "/hello"), далее указывается, что этот метод будет обрабатывать GET-запросы (method = RequestMethod.GET).

Также, нужно отметить важные моменты в отношении приведённого выше контроллера: − Вы определяете бизнес-логику внутри связанного таким образом служебного метода. Из него Вы можете вызывать любые другие методы.

− Основываясь на заданной бизнес-логике, в рамках этого метода Вы создаёте Модель (Model). Вы можете добавлять аттрибуты Модели, которые будут добавлены в Вид (View).

В примере выше мы создаём Модель с атрибутом «message». − Данный служебный метод возвращает имя Вида в виде строки String. В данном случае, запрашиваемый Вид имеет имя «hello».

Spring MVC поддерживает множество типов Видов для различных технологий отображения страницы. В том числе — JSP, HTML, PDF, Excel, XML, Velocity templates, XSLT, JSON, каналы Atom и RSS, JasperReports и проч. Но чаще всего используются шаблоны JSP, написанные при помощи JSTL или HTML файлы, используя Thymeleaf

## 3.2 [Проектирование структур хранения данных](#_Toc474748997)

Логическая структура реляционной базы данных определяется совокупностью логически взаимосвязанных реляционных таблиц. Каждая реляционная таблица имеет структуру, определяемую реквизитным составом одного из информационных объектов полученной ИЛМ [12]. Логические связи таблиц соответствуют структурным связям между объектами.

Логическая структура реляционной базы данных, построенная на основе полученной пунктом ранее концептуальной модели, приведена на рисунке 3.2. На этой схеме реляционные таблицы представлены структурой, определяемой составом и последовательностью полей (атрибутов). Наименования ключевых полей выделены ключом. Логические связи изображены линиями между соответствующими ключами связи.

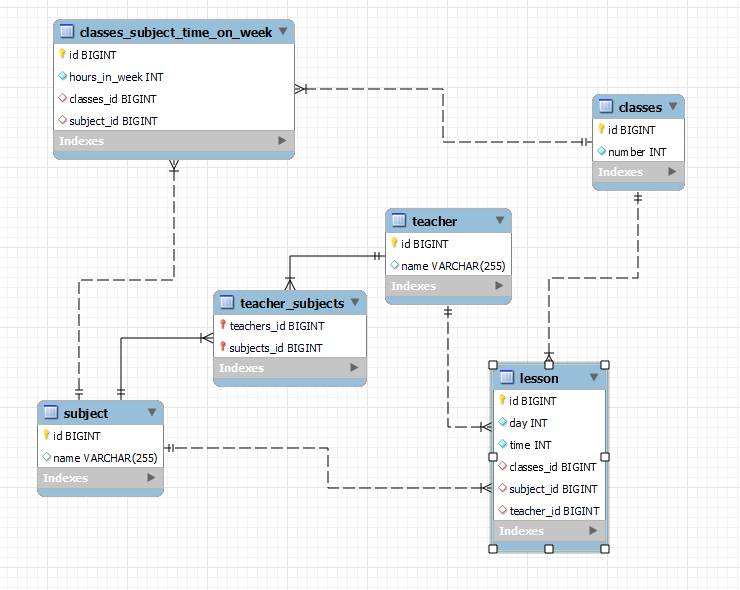


Рисунок 3.2 – Структура базы данных

Для того чтобы понять приведена ли база данных к третьей нормальной форме, воспользуемся определениями первой, второй и третьей нормальных форм.

Переменная отношения находится в первой нормальной форме (1НФ) тогда и только тогда, когда в любом допустимом значении отношения каждый его кортеж содержит только одно значение для каждого из атрибутов.

Переменная отношения находится во второй нормальной форме тогда и только тогда, когда она находится в первой нормальной форме, и каждый не ключевой атрибут функционально полно зависит от ее потенциального ключа.

Переменная отношения находится в третьей нормальной форме тогда и только тогда, когда она находится во второй нормальной форме, и отсутствуют транзитивные функциональные зависимости не ключевых атрибутов от ключевых. Ссылаясь на эти определения, можно утверждать, что созданная база данных находится в третьей нормальной форме.

## 3.3 [Описание реализации вариантов использования](#_Toc474748998)

Моделирование необходимо для понимания системы. Обычно, при этом единственной модели никогда не бывает достаточно. Наоборот, для понимания практически любой нетривиальной системы приходится разрабатывать большое количество взаимосвязанных моделей. В применении к программным системам это означает, что необходим язык, с помощью которого можно с различных точек зрения описать представления архитектуры системы на протяжении цикла ее разработки.

Диаграмма вариантов использования показывает, какая функциональность должна быть реализована в системе, основные функции, которые должны быть включены в систему, их окружение и взаимодействие функций с окружением [13].

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью, так называемых вариантов использования. При этом актером или действующим лицом называется любая сущность, взаимодействующая с системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик. В свою очередь, вариант использования служит для описания сервисов, которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой.

Ниже приводится диаграмма вариантов использования, разработанной программы учета и регистрации поступлений материалов. Данная диаграмма отображает возможности и функционал разрабатываемой программы. На рисунке 3.3 представлена диаграмма вариантов использования, на которой указаны все возможные действия, которые может осуществлять пользователь при работе с данной программой.

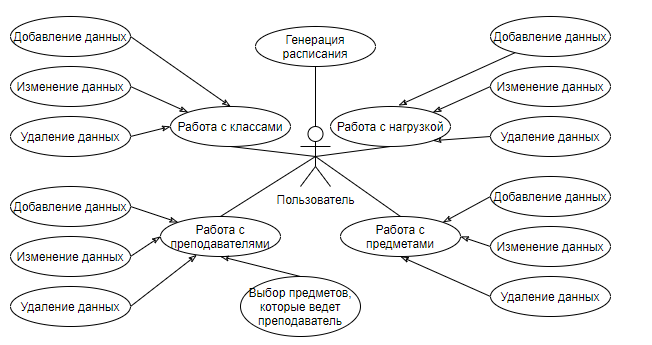


Рисунок 3.3 - Диаграмма вариантов использования

# 4 РЕАЛИЗАЦИЯ

## 4.1 Разработка классов информационной системы

Так как мы используем шаблон MVC, идет отдельно разработка классов модели (сущности), отдельно вида (страниц для браузера) и контроллеров (обработки действий пользователя). Так как данные в шаблоне будут изменяться по требованию администратора ресурса, нам нужно создать класс-сущность для хранения этих данных. Сущность (entity) – это реальный или представляемый тип объекта, информация о котором должна сохраняться и быть доступна. Сущности уже были спроектированы раньше для базы данных. На их основе пишутся классы сервиса. Пример описания сущности приведен ниже.

Листинг 1. Сущность Предмета

package com.hescha.teacher\_workload\_accounting.entity;  
  
import java.util.ArrayList;  
import java.util.HashSet;  
import java.util.List;  
import java.util.Objects;  
import java.util.Set;  
  
import javax.persistence.CascadeType;  
import javax.persistence.Entity;  
import javax.persistence.FetchType;  
import javax.persistence.ManyToMany;  
import javax.persistence.OneToMany;  
  
import org.hibernate.annotations.Fetch;  
import org.hibernate.annotations.FetchMode;  
  
import lombok.Data;  
  
@Data  
@Entity  
public class Subject extends AbstractEntity {  
  
 private String name;  
  
 @ManyToMany(mappedBy = "subjects", fetch = FetchType.EAGER)  
 private Set<Teacher> teachers = new HashSet<>();  
 @Fetch(value = FetchMode.SELECT)  
 @OneToMany(fetch = FetchType.EAGER, mappedBy = "subject", cascade = CascadeType.PERSIST)  
 private List<ClassesSubjectTimeOnWeek> subjectTimeOnWeeks = new ArrayList<>();  
 @Fetch(value = FetchMode.SELECT)  
 @OneToMany(fetch = FetchType.EAGER, mappedBy = "subject", cascade = CascadeType.PERSIST)  
 private List<Lesson> lessons = new ArrayList<>();  
  
 @Override  
 public String toString() {  
 return name;  
 }  
  
 @Override  
 public boolean equals(Object obj) {  
 if (this == obj) {  
 return true;  
 }  
 if (!super.equals(obj)) {  
 return false;  
 }  
 if (getClass() != obj.getClass()) {  
 return false;  
 }  
 Subject other = (Subject) obj;  
 return Objects.equals(name, other.name);  
 }  
  
 @Override  
 public int hashCode() {  
 final int prime = 31;  
 int result = super.hashCode();  
 result = prime \* result + Objects.hash(name);  
 return result;  
 }  
  
}

По описанию сущности автоматически будет сгенерирована таблица в базе данных, при ее отсутствии. Сущность написана, переходим к созданию сервисов для доступа к бд. Сперва опишем интерфейс с основными командами.

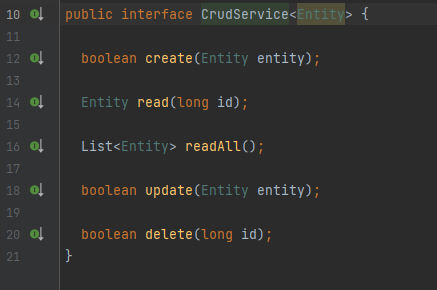


Рисунок 4.1 – Интерфейс с основными командами

Затем переходим к написанию реализации этого интерфейса. На самом деле, реализация простая, мы вызываем соответствующие методы репозитория.

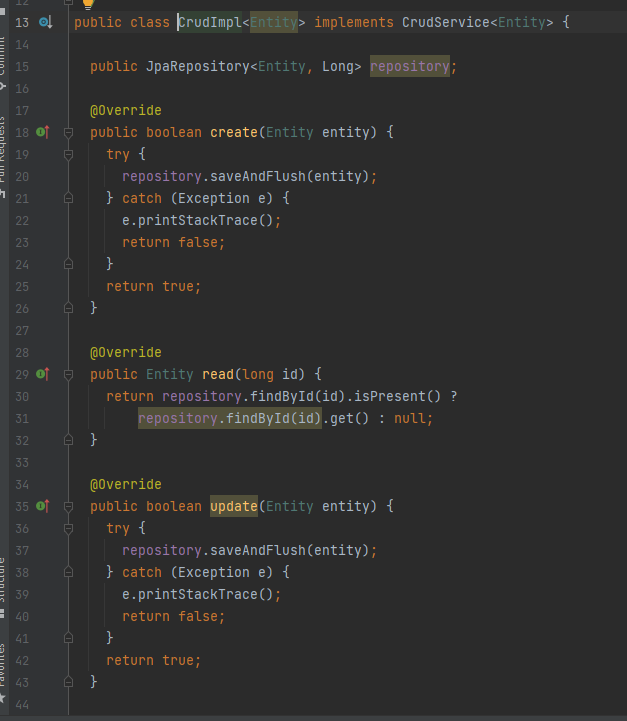


Рисунок 4.2 – Реализация интерфейса сервисов

Реализация есть, осталось связать нашу сущность шаблона с доступом к бд. Для этого наследуемся от только-что созданной реализации, и в конструктор передаем нашу сущность.

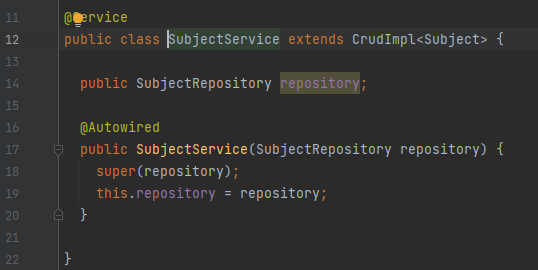


Рисунок 4.3 – Реализация сервиса предметов

Проделываем так для каждой сущности и готово, теперь, используя данный сервис, мы можем добавлять/редактировать/удалять данные в коде приложения.

Но это только в коде, пользователю это пока недоступно. Чтобы пользователь смог как-то взаимодействовать с приложением, создаем контроллеры страниц.

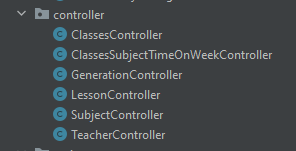


Рисунок 4.4 – Контроллеры страниц

По одному контроллеру для работы с соответствующими таблицами в базе данных и для генерации расписания.

Контроллер работает с данными таблиц базы данных, считывая или записывая в нее данные от пользователя. При запросе отображения, он возвращает страницу отображения.

В этом контроллере реализованы методы создания, редактирования и удаления шаблонов по соответствующим адресам.

Листинг 2. Контроллер предмета

package com.hescha.teacher\_workload\_accounting.controller;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RequestMethod;  
  
import com.hescha.teacher\_workload\_accounting.entity.Subject;  
import com.hescha.teacher\_workload\_accounting.service.SubjectService;  
  
@Controller  
@RequestMapping("/subjects")  
public class SubjectController {  
  
 @Autowired  
 private SubjectService service;  
  
 @GetMapping  
 public String getList(Model model) {  
 model.addAttribute("list", service.repository.findAll());  
 return "subjects";  
 }  
  
 @RequestMapping(path = {"/edit", "/edit/{id}"})  
 public String edit(Model model, @PathVariable(name = "id", required = false) Long id) {  
 if (id != null) {  
 Subject entity = service.read(id);  
 model.addAttribute("entity", entity);  
 } else {  
 model.addAttribute("entity", new Subject());  
 }  
 return "subjects-add-edit";  
 }  
  
 @RequestMapping(path = "/delete/{id}")  
 public String delete(Model model, @PathVariable("id") Long id) {  
 service.delete(id);  
 return "redirect:/subjects";  
 }  
  
 @RequestMapping(path = "/create", method = RequestMethod.*POST*)  
 public String createOrUpdate(Subject entity) {  
 service.create(entity);  
 return "redirect:/subjects";  
 }  
  
}

Класс наследует стандартный класс настройки безопасности. В нем мы сразу создаем данные для входа администратора и в методе конфигурации прописываем, какие пользователи куда могут попасть.

Поскольку использованные шаблоны нужно где-то хранить, создаем настройку подключения к базе данных. Все это делается в файле application.properties.

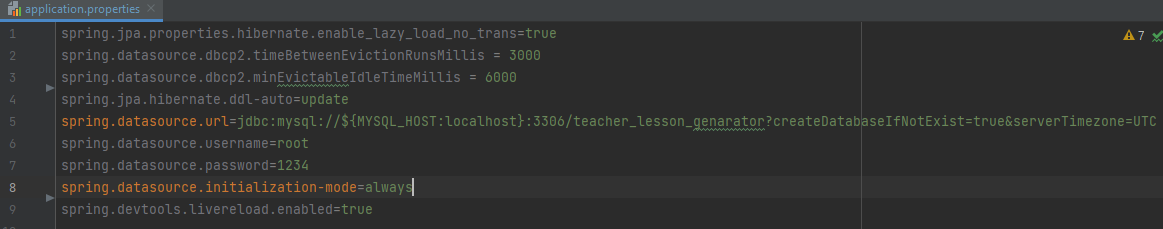


Рисунок 4.5 – Файл настройщик приложений

Проделав все эти операции, мы имеем следующую диаграмму классов



Рисунок 4.6 – Диаграмма классов

## 4.2 [Разработка интерфейса программного продукта](#_Toc474749001)

Остался последний шаг – связывание страниц шаблонов с данными приложения. Для этого воспользуемся шаблонизатором Thymeleaf, который работает в связке с Spring. Суть его проста – те данные, что мы передавали в модель отображения, таймлиф можем выводить простой командой th:text=”${}”.

В скобках пишется имя переменной. Обращение идет как к переменным класса. В нужных местах вставляем этот код. Так, для отображения списка нагрузок будут использоваться строки листинга 3.

Листинг 3. Шаблон для отображения списка нагрузок

<!DOCTYPE html>  
<html lang="en" xmlns:th="http://www.thymeleaf.org">  
<head>  
<title>Нагрузка</title>  
<div th:replace="~{commons :: head}"></div>  
</head>  
<body>  
  
 <div th:replace="~{commons :: nav}"></div>  
  
 <section class="list">  
 <div class="container my-2">  
 <div class="card">  
 <div class="row">  
 <div class="card-body col">  
 <h2 class="text-center">Нагрузка</h2>  
 <div th:switch="${list}" class="container my-5">  
 <div class="col-md-12">  
 <h2 th:case="null">Записей не найдено</h2>  
 <p class="d-flex justify-content-end">  
 <a href="/classesSubjectTimeOnWeeks/edit" class="myButton mx-2">  
 <i class="fas fa-plus"> Добавить запись </i>  
 </a>  
 </p>  
 <div th:case="\*" class="overScroll">  
 <table class="table table-striped table-responsive-md">  
 <thead>  
 <tr>  
 <th>Класс</th>  
 <th>Предмет</th>  
 <th>Количество часов</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr th:each="entity : ${list}">  
 <td th:text="${entity.classes}"></td>  
 <td th:text="${entity.subject}"></td>  
 <td th:text="${entity.hoursInWeek}"></td>  
 <td class="widthLastCol "><a  
 th:href="@{/classesSubjectTimeOnWeeks/edit/{id}(id=${entity.id})}"  
 class="myButton "> <i class="fas fa-edit">Редактировать</i>  
 </a></td>  
 <td class="widthLastCol "><a  
 th:href="@{/classesSubjectTimeOnWeeks/delete/{id}(id=${entity.id})}"  
 class="myButton"> <i class="fas fa-trash">Удалить</i>  
 </a></td>  
 </tr>  
 </tbody>  
 </table>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 </section>  
  
 <div th:replace="~{commons :: footer}"></div>  
  
</body>  
</html>

После создания всех шаблон можно просмотреть полученный интерфейс.

При переходе на сайт, если в базе данных еще нет информации о нагрузке, пользователь увидит лишь пустую таблицу с заголовками. В нее будет заносится информация о нагрузке.

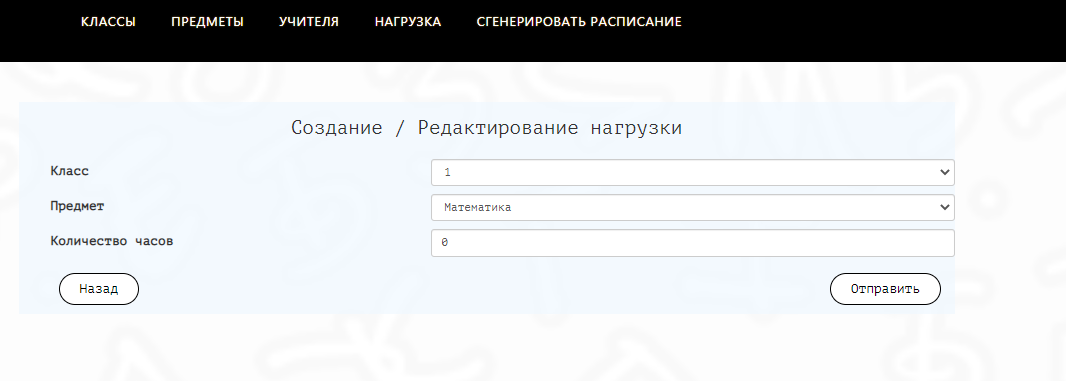


Рисунок 4.7 - Страница добавления нагрузки

На текущей страницу будет заноситься информация, какой предмет какому классу сколько часов в неделю необходимо преподавать. Для выбора предмета и класса используется выпадающий список с ранее занесенной информацией.

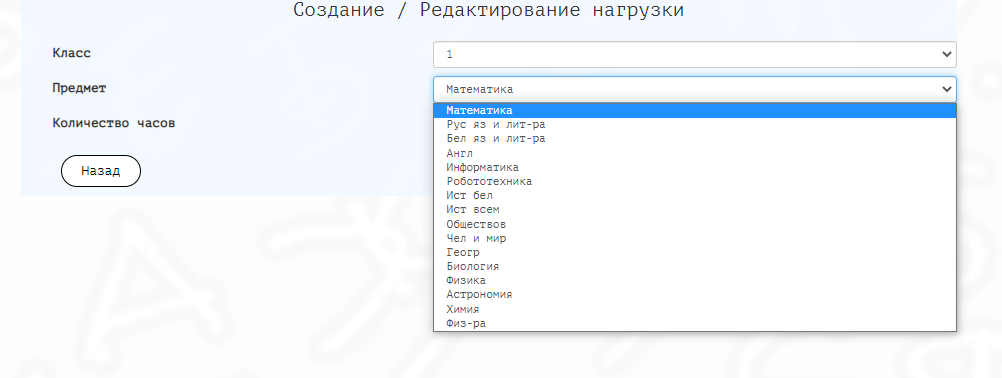


Рисунок 4.8 - Пример выпадающего списка при добавлении нагрузки

Первым делом необходимо занести классы в систему. На случай, если не используется sql код дампа базы, добавлена страница редактирования классов

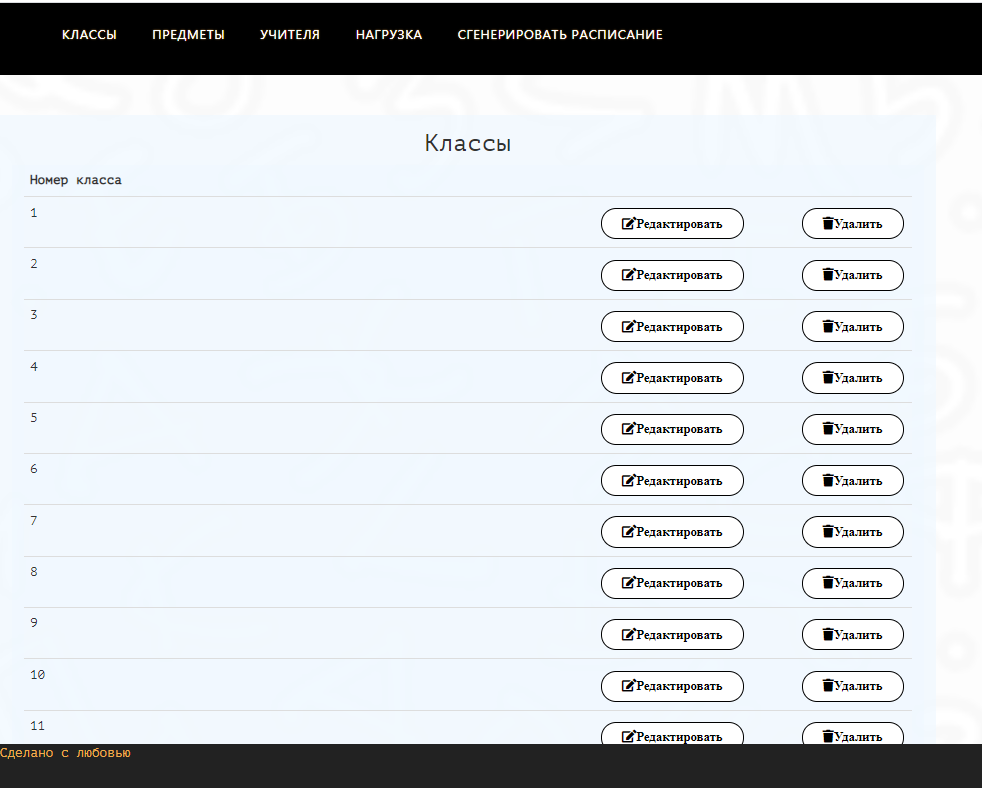


Рисунок 4.9 - Страница работы с классами

После занесения классов следует добавить в систему предметы (уроки). Ограничений по добавлениям нет, главное чтоб имена не повтрялись.

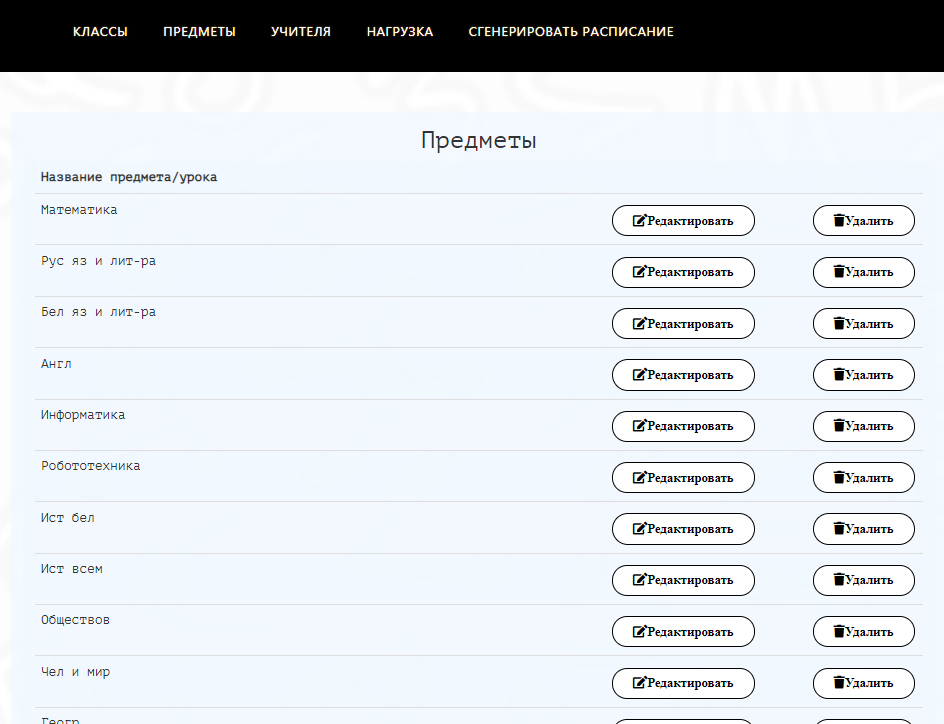


Рисунок 4.10 - Страница работы с предметами

После занесения предметов следует добавить в систему преподавателей (учителей).

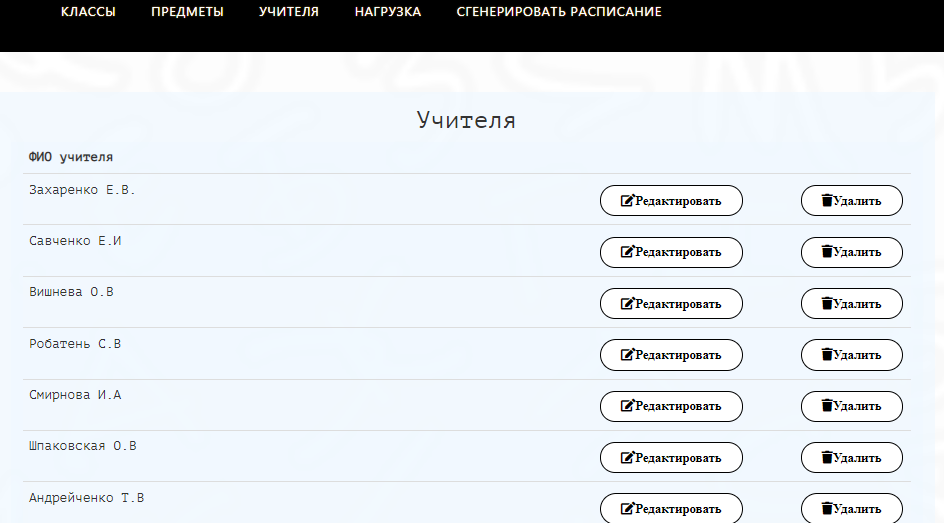


Рисунок 4.11 - Страница работы с учителями

Делается это после добавления предметов оттого, что при добавлении учителя выбирается список предметов, которые этот учитель может вести.

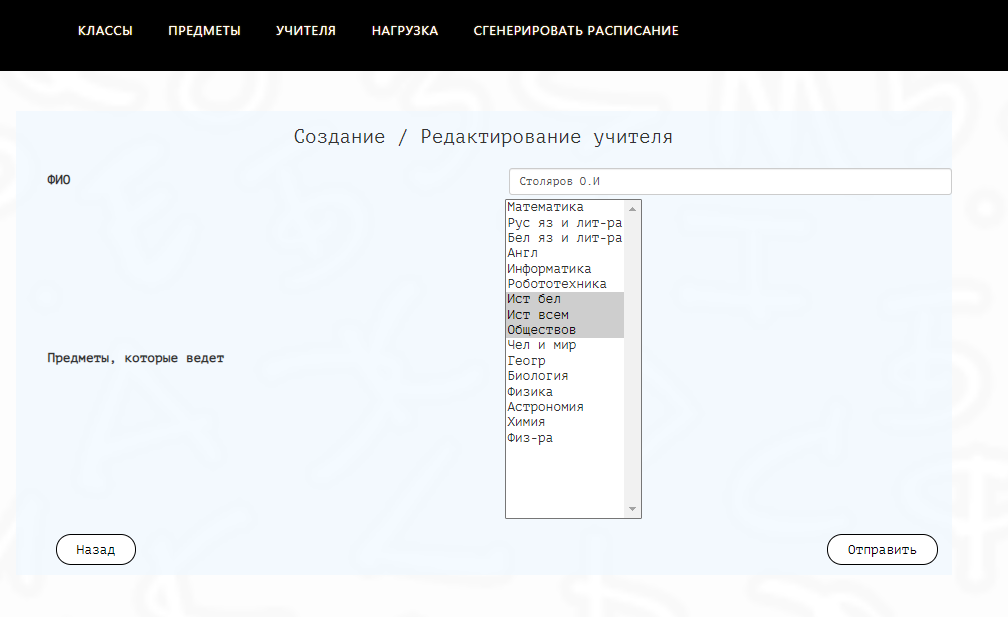


Рисунок 4.12 - Страница добавления учителя

После занесения всей информации в систему можно переходить на страницу генерации расписания. При переходе на нее сразу будет сгенерировано новое расписание. Для удобства, оно сразу разделено по дням от понедельника до пятницы.

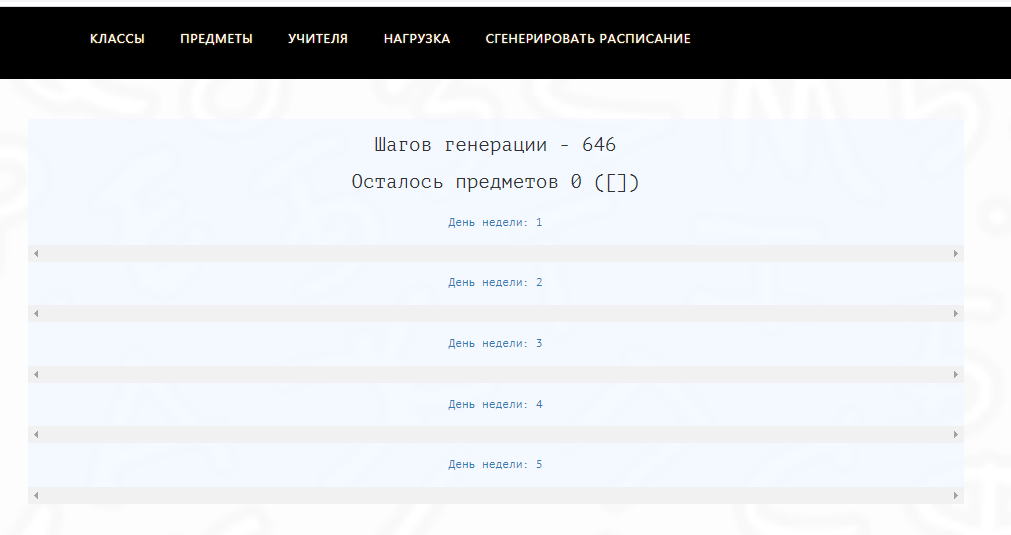


Рисунок 4.13 - Страница генерации расписания

При открытии любого дня по очереди будут показаны расписания уроков от 1 и до 11 класса.

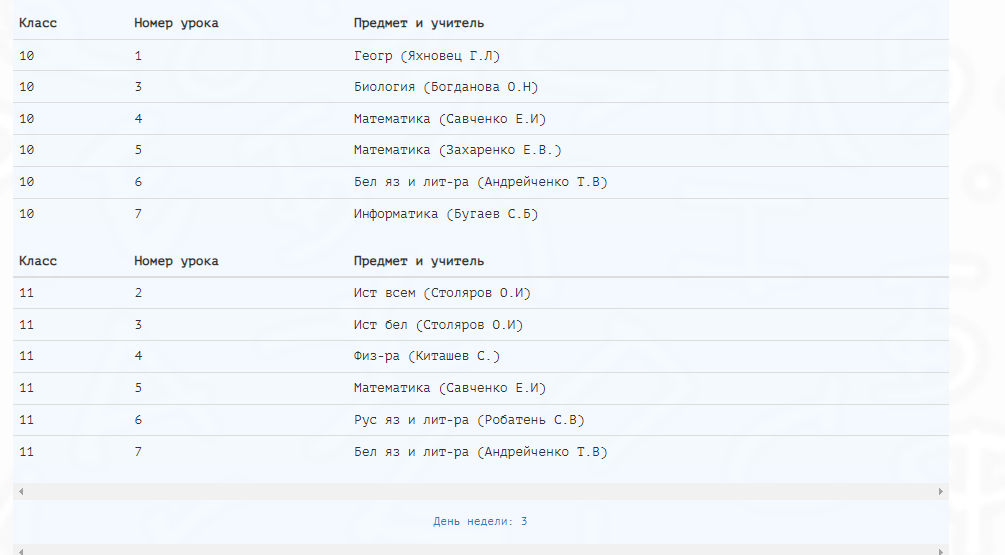


Рисунок 4.14 - Подробное расписание второго дня

Также, при каждой генерации показывается дополнительная информация, а именно количество итераций для генерации и предметы, которые не удалось добавить в расписание. Если за 1 000 000 шагов не удастся составить расписание, пользователю будет показано текущий вариант с указанием предмета, который не удалось добавить.

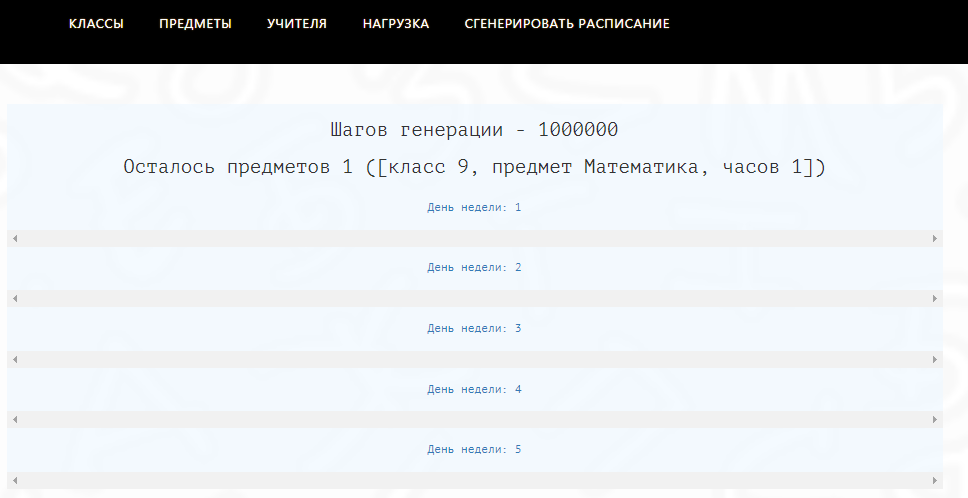


Рисунок 4.15 - Информация о не вошедших уроках в расписание

Если все предметы были добавлены, информация об этом также будет выведена на экран.

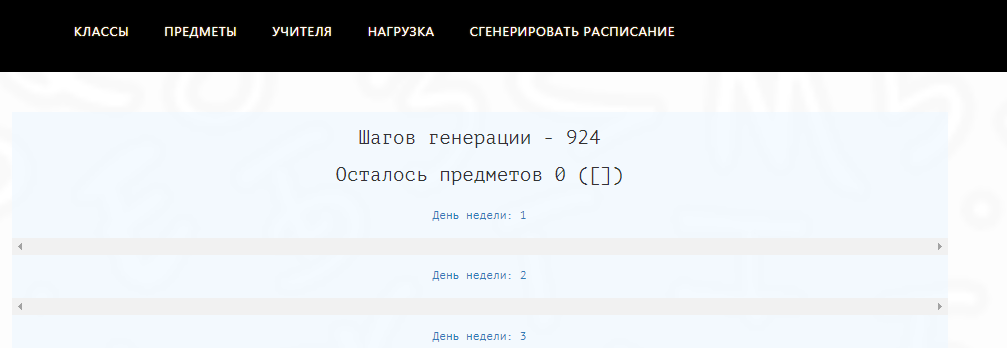


Рисунок 4.16 - Информация, что все предметы вошли в расписание

О том, что каждый раз генерируется новое расписание можно убедиться простой перезагрузкой страницы. Сравним два расписания 11 класса в понедельник.

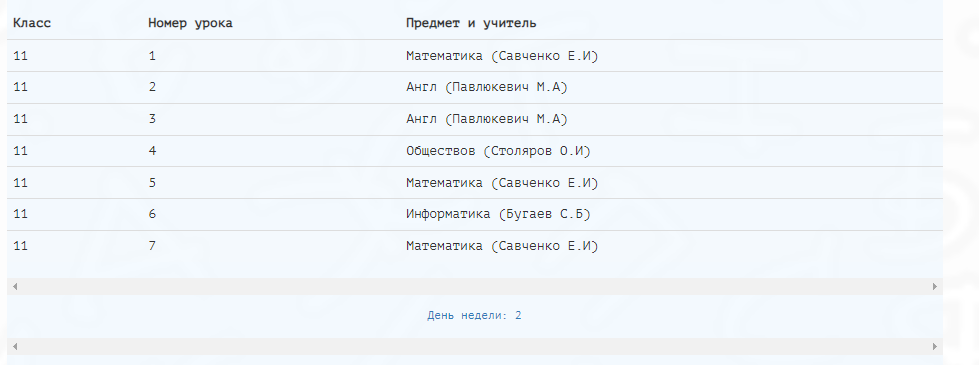


Рисунок 4.17 - Первый вариант расписания

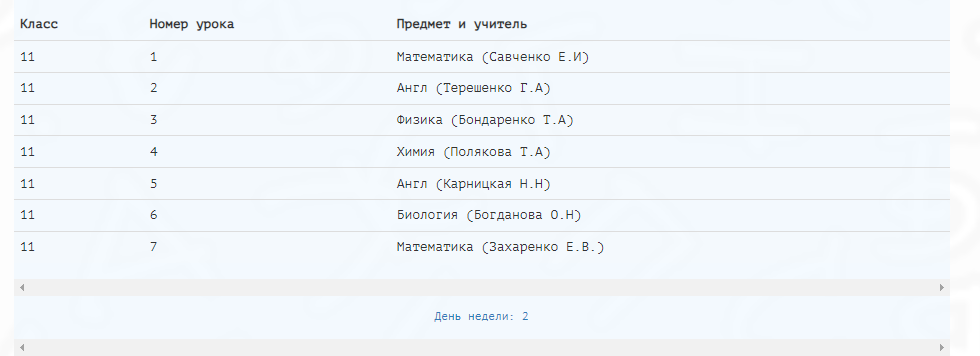


Рисунок 4.18 - Второй вариант расписания

Таким образом, если расписание не будет нравится кому-то из преподавателей, достаточно будет перезагружать старнциу и подобрать под себя лучшее.

## 4.3 [Разработка алгоритмов реализации вариантов использования](#_Toc474749002)

При решении общей задачи, возникали более мелкие, для решения которых необходимо было использовать и разрабатывать алгоритмы. Несколько таких примеров будет показано ниже.

1. Сохранение нагрузки с использованием выпадающих списоков

@RequestMapping(path = "/create", method = RequestMethod.*POST*)  
public String createOrUpdate(ClassesSubjectTimeOnWeek entity,  
 @Param("subjectId") Long subjectId,  
 @Param("classesId") Long classesId) {  
 entity.setClasses(classesService.read(classesId));  
 entity.setSubject(subjectService.read(subjectId));  
 service.create(entity);  
 return "redirect:/classesSubjectTimeOnWeeks";  
}

1. Генерация расписания

Для генерации расписания через рандомайзер получали случайное время урока с случайный день и случайное требование нагрузки. После чего проверяли, занят ли день, и если ли учителя, которые могут вести урок. Если учителя были, то проверяли, не ведет ли этот учитель урок в другом классе в это же время

private void makeList() {  
 iter = 0;  
 while (iter < 1\_000\_000 && subjectTimeOnWeeks.size() > 0) {  
 iter++;  
  
 // получаем случайное требование  
 int randomLessonConditionId = rand.nextInt(subjectTimeOnWeeks.size());  
 ClassesSubjectTimeOnWeek subjectTimeOnWeek = subjectTimeOnWeeks.get(randomLessonConditionId);  
  
 // получаем случайный день недели  
 int randomDayOfWeekNumber = rand.nextInt(lessons.length);  
 Lesson[][] randomDay = lessons[randomDayOfWeekNumber];  
  
 // получаем все уроки класса  
 Lesson[] classLessons = randomDay[subjectTimeOnWeek.getClasses().getNumber() - 1];  
 // получаем случайный номер урока по счету выбранного класса в выбранный день  
 int lessonOnDayNumber = rand.nextInt(classLessons.length);  
 Lesson lesson = classLessons[lessonOnDayNumber];  
 // если место не свободно - пропустить  
 if (lesson != null) {  
 continue;  
 }  
  
 // если свободно - выбрать препода, проверить на совместимость  
 Teacher teacher = getTeacherBySubject(subjectTimeOnWeek);  
 // проверяем, нет ли в этот же день у других классов с этим же преподом урока в  
 // это же время  
 boolean canSetLesson = checkPossibleSetLesson(randomDay, lessonOnDayNumber, teacher);  
 if (!canSetLesson) {  
 continue;  
 }  
  
 //создаем урок  
 Lesson newLesson = createLesson(subjectTimeOnWeek, randomDayOfWeekNumber, lessonOnDayNumber,  
 teacher);  
  
 addLessonToArray(randomLessonConditionId, subjectTimeOnWeek, randomDayOfWeekNumber,  
 lessonOnDayNumber, newLesson);  
 }  
  
}

## Модульное тестирование алгоритмов реализации вариантов использования

Модульное тестирование, или юнит-тестирование - это процесс в программировании, который направлен на проверку отдельных небольших частей приложения, также называемых атомарными, которые можно исследовать изолированно от других подобных частей. При выполнении данного тестирования могут проверяться как отдельные функции или методы классов, так и сами классы, взаимодействие классов, небольшие библиотеки и отдельные части приложения.

Довольно часто данный вид тестирования реализуется с использованием специальных технологий и инструментальных средств автоматизации тестирования, значительно упрощающих и ускоряющих разработку соответствующих тест-кейсов. Таким образом, оценивая каждый элемент отдельно и подтверждая правильность его работы, установить проблему значительно проще чем, если бы элемент был частью системы.

Примеры тестирования алгоритмов реализации вариантов использования:

1. Тестирование вывода преподавателей (таблица 4.1).

Таблица 4.1 - Тестирование вывода преподавателей

|  |  |  |  |
| --- | --- | --- | --- |
| Краткое описание | Предварительные условия | Шаги для воспроизведения | Ожидаемый результат |
| Тестирование вывода списка преподавателей | Запустить программу  Добавить по крайней мере двух преподавателей в базу данных | Используя меню навигации перейти на страницу преподавателей | При переходе на страницу отображен список преподавателей из базы данных |

1. Тестирование добавления нагрузки (таблица 4.2).

Таблица 4.2 - Тестирование добавления нагрузки

|  |  |  |  |
| --- | --- | --- | --- |
| Краткое описание | Предварительные условия | Шаги для воспроизведения | Ожидаемый результат |
| Тестирование добавления нагрузки | Запустить программу  Добавить информацию о классах, предметах, учителях | Перейти на страницу добавления нагрузки. Выбрать данные из выпадающего списка и заполнить поля. Отправить данные. | После сохранения, новая нагрузка выводится на странице вместе с другими существующими |

1. Тестирование изменения преподавателя (таблица 4.3).

Таблица 4.3 - Тестирование изменения преподавателя

|  |  |  |  |
| --- | --- | --- | --- |
| Краткое описание | Предварительные условия | Шаги для воспроизведения | Ожидаемый результат |
| Тестирование изменение преподавателя | Запустить программу. Добавить преподавателя. | Перейти на страницу учителей. Выбрать из списка последнего для редактирования. Изменить данные и отправить. | В таблице учителей отображаются новые данные |

# 5 СИСТЕМНОЕ ТЕСТИРОВАНИЕ

Системное тестирование - это тестирование программного обеспечения, выполняемое на полной, интегрированной системе, с целью проверки соответствия системы исходным требованиям. Системное тестирование относится к методам тестирования чёрного ящика, и, тем самым, не требует знаний о внутреннем устройстве системы. Тестирование бывает:

* функциональное тестирование;
* тестирование пользовательского интерфейса;
* тестирование совместимости;
* тестирование безопасности;
* тестирование производительности;
* автоматическое тестирование;
* интеграционное тестирование.

## 5.1 Функциональное тестирование

Функциональное тестирование - это тестирование программного обеспечения в целях проверки реализуемости функциональных требований, то есть способности программного обеспечения в определённых условиях решать задачи, нужные пользователям. Функциональные требования определяют, что именно делает программное обеспечение, какие задачи оно решает.

Функциональные требования включают в себя:

* функциональная пригодность;
* точность;
* способность к взаимодействию;
* соответствие стандартам и правилам;
* защищённость.

Тестирование и отладка программы являются наиважнейшими этапами разработки любых программных продуктов. Цель этого этапа – проверка правильности и точности реализации функций, выполнение которых возлагается на данный программный продукт. В случае выявления некоторых неточностей и ошибок необходимо проведение работ по их исправлению и доработке программного продукта до требуемого уровня.

На основе функций, которые должны быть протестированы в разработанной информационной обучающей системе, составлен чек-лист (checklist), приведенный в таблице 5.1.

Таблица 5.1 – Чек-лист

|  |  |  |  |
| --- | --- | --- | --- |
| Тестируемый модуль | | Тестируемая функция | Результат |
| 1 | | 2 | 3 |
| Запуск приложения | | Функция запуска | Выполнено успешно |
| Выход из приложения | | Функция закрытия программы |
| Ввод данных | | Определение соответствие типа вводимых данных | Выполнено успешно |
| Ввод ошибочных данных | Вывод сообщении об ошибке | | Выполнено успешно |
| Сохранение и загрузка | Сохранение данных в базу данных | | Выполнена успешно |
| Чтение данных из базы данных | |

В качестве тестирования программного продукта был выбран тест-кейс (**Test Case)**.

**Test Case** - это артефакт, описывающий совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части.

Тестирование производилось на ОС «Windws 10.

Таблица 5.2 – Тест-кейсы

|  |  |  |  |
| --- | --- | --- | --- |
| Краткое описание | Предварительные условия | Шаги для воспроизведения | Ожидаемый результат |
| Тестирование генерации раписания | База данных должна хранить половину достаточный набор данных для генерации расписания | Перейти на страницу Генерации. Обновить страницу | При каждом обновлении страницы генерируется новое расписание. Учителя не пересекаются в один и тот же промежуток времени |
| Тестирование неправильного ввода | Переход на страницу добавления нагрузки | В поля для чисел ввести буквы  Не заполнять выпадающие списки | Пользователь увидит сообщение о некорректном вводе |
| Тестирование сортировки | В базе данных должны быть записи о нагрузке | Перейти на страницу нагрузки. Через заголовки таблицы сортировать | На странице нагрузки показываются сортированные данные по выбранному столбцу |

Таблица 5.3 – Набор тестов отображения информации

|  |  |  |  |
| --- | --- | --- | --- |
| Место проведения теста | Содержание теста | Ожидаемый результат | Отметка о прохождении теста |
| Все страницы | Перейти на любую из вкладок в произвольном порядке. | Корректное отображение информации, отображение того, чего ожидает пользователь. | Да |
| Все вкладки | Увеличение масштаба отображения средствами браузера. | Появление скроллов справа и снизу, корректное отображение информации. | Да |
| Все вкладки | Обновление страницы используя средства браузера. | Состояние веб-сервиса до обновления и после не изменилось. | Да |
| Меню браузера | Нажатие функциональных кнопок браузера «Назад» и «Вперед» в произвольный момент работы веб-приложения. | Корректное поведение веб-сервиса. Отображение информации. | Да |

## 5.2 Оценка безопасности

Обеспечение безопасности информационных систем представляет собой ряд мероприятий, направленных на предотвращение несанкционированных воздействий на защищаемую информацию, а также её утечки. Поскольку приложение построено на базе фреймворка Spring, вопросы безопасности берет на себя Spring Security.

Spring Security это Java/JavaEE framework, предоставляющий механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для корпоративных приложений, созданных с помощью Spring Framework. Проект был начат Беном Алексом (Ben Alex) в конце 2003 года под именем «Acegi Security», первый релиз вышел в 2004 году. Впоследствии проект был поглощён Spring'ом и стал его официальным дочерним проектом. Впервые публично представлен под новым именем Spring Security 2.0.0 в апреле 2008 года.

Ключевые объекты контекста Spring Security:

SecurityContextHolder, в нем содержится информация о текущем контексте безопасности приложения, который включает в себя подробную информацию о пользователе(Principal) работающем в настоящее время с приложением. По умолчанию SecurityContextHolder используетThreadLocal для хранения такой информации, что означает, что контекст безопасности всегда доступен для методов, исполняющихся в том же самом потоке. Для того что бы изменить стратегию хранения этой информации можно воспользоваться статическим методом класса SecurityContextHolder.setStrategyName(String strategy). Более подробно SecurityContextHolder.

SecurityContext, содержит объект Authentication и в случае необходимости информацию системы безопасности, связанную с запросом от пользователя.

GrantedAuthority отражает разрешения выданные пользователю в масштабе всего приложения, такие разрешения (как правило называются «роли»), например ROLE\_ANONYMOUS, ROLE\_USER, ROLE\_ADMIN.

UserDetails предоставляет необходимую информацию для построения объекта Authentication из DAO объектов приложения или других источников данных системы безопасности. Объект UserDetailsсодержит имя пользователя, пароль, флаги: isAccountNonExpired, isAccountNonLocked, isCredentialsNonExpired, isEnabled и Collection — прав (ролей) пользователя.

UserDetailsService, используется чтобы создать UserDetails объект путем реализации единственного метода этого интерфейса.

UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;

Позволяет получить из источника данных объект пользователя и сформировать из него объект UserDetails который будет использоваться контекстом Spring Security.

**5.3 Тестирование производительности**

Тестирование производительности**-** это тестирование, которое проводится для определения скорости работы системы или её части при заданной нагрузке. Тестирование производительности стремится учесть производительность уже на стадии проектирования и моделирования и системы, до начала основной стадии разработки.

Тестирование производительности служит таким типичным целям:

* для демонстрации того, что система удовлетворяет критериям производительности;
* для определения производительность какой из двух или нескольких систем лучше;
* для определения, какой элемент нагрузки или часть системы приводит к снижению производительности.
* Для оценки времени загрузки страниц системы был использован специализированный веб-сервис «https://developers.google.com». PageSpeed Insights анализирует содержание веб-страницы и предлагает решения, которые позволят ускорить ее загрузку.

В процессе тестирования были произведены замеры:

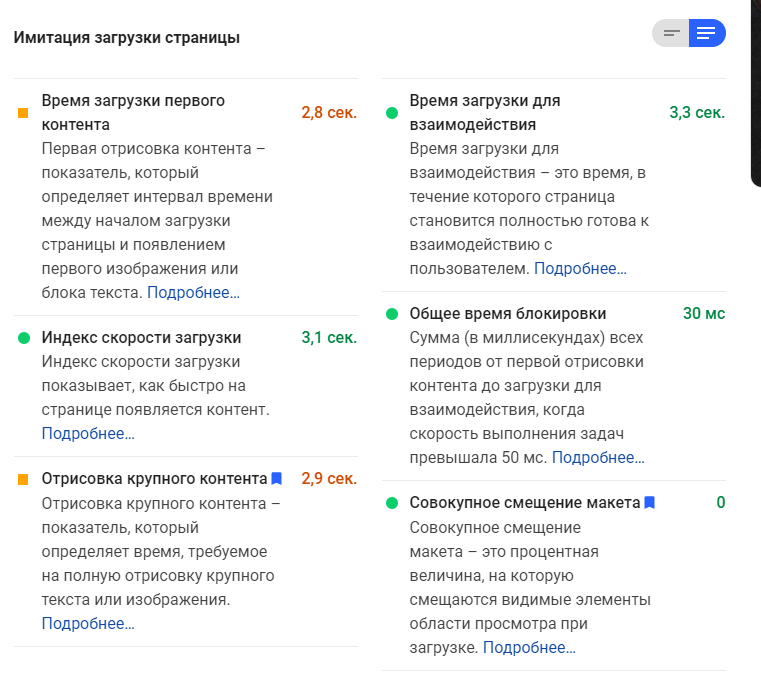
****

Рисунок 5.1 - Тестирование

# ЗАКЛЮЧЕНИЕ

Итогом написания дипломного проекта является программное веб-приложение, которое предоставляет собой сайт с возможностью генерировать расписание на основе введенных данных.

Данное приложение является доступным и понятным любому пользователю, располагает удобным и минималистичным интерфейсом, а также соответствует всем требованиям, предъявленным к проекту.

Данная программа является web- приложением, что говорит о том, что множество пользователей, расположенных на определенном расстоянии друг от друга, могут одновременно общаться с сервером. Вся используемая информация хранится в базе данных, разработанной для данного проекта.

За время написания дипломной работы были изучены функции и возможности написания приложений на Java, а также получены основные навыки разработки приложений с графическим интерфейсом пользователя.

Вследствие разработки программы были выполнены следующие задачи:

* Проанализирована предметная область.
* Определены требования к программе.
* Выбраны средства реализации приложения.
* Разработано программное средство.

Разработанное приложение прошло все этапы тестирования и полностью соответствует всем определенным к нему требованиям.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Расписание уроков [Электронный ресурс]. – Режим доступа: https://blanker.ru/doc/raspisanie-urokov. – Дата доступа: 08.08.2021.
2. Этапы проектирования данных [Электронный ресурс]. – Режим доступа: http://www.mstu.edu.ru/study/materials/zelenkov/ch\_5\_1.html. – Дата доступа: 08.08.2021.
3. Концептуальная модель базы данных [Электронный ресурс]. – Режим доступа: https://prog.bobrodobro.ru/66666. – Дата доступа: 08.08.2021.
4. Типы связей в реляционных базах данных [Электронный ресурс]. – Режим доступа: https://office-menu.ru/uroki-sql/41-tipy-svyazej-v-relyatsionnykh-bazakh-dannykh. – Дата доступа: 08.08.2021.
5. Процесс проектирования как последовательная трансляция требований, предъявляемых к системе [Электронный ресурс]. – Режим доступа: https://studref.com/316098/informatika/proektirovanie\_programmnyh\_sistem\_opredelenie\_trebovaniy\_tseley\_programmnogo\_produkta. – Дата доступа: 08.08.2021.
6. Что такое технология Java и каково ее применение? [Электронный ресурс]. – Режим доступа: https://www.java.com/ru/download/help/whatis\_java.html. – Дата доступа: 08.08.2021.
7. Spring Framework [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Spring\_Framework. – Дата доступа: 08.08.2021.
8. Что такое HTML? Основы языка разметки гипертекста [Электронный ресурс]. – Режим доступа: https://www.hostinger.ru/rukovodstva/shto-takoje-html/. – Дата доступа: 08.08.2021.
9. CSS [Электронный ресурс]. – Режим доступа: http://web.spt42.ru/index.php/chto-takoe-css. – Дата доступа: 08.08.2021.
10. Общая характеристика языка UML [Электронный ресурс]. – Режим доступа: http://www.informicus.ru/default.aspx?SECTION=6&id=73&subdivisionid=2. – Дата доступа: 08.08.2021.
11. Структура реляционных баз данных [Электронный ресурс]. – Режим доступа: https://www.oracle.com/ru/database/what-is-a-relational-database/. – Дата доступа: 08.08.2021.
12. Лекция 3: Элементы графической нотации диаграммы вариантов использования [Электронный ресурс]. – Режим доступа: https://intuit.ru/studies/courses/32/32/lecture/1004. – Дата доступа: 08.08.2021.

# ПРИЛОЖЕНИЕ А. Исходный код программы

package com.hescha.teacher\_workload\_accounting.controller;  
  
import java.util.List;  
import java.util.Random;  
import java.util.Set;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestMapping;  
  
import com.hescha.teacher\_workload\_accounting.entity.Classes;  
import com.hescha.teacher\_workload\_accounting.entity.ClassesSubjectTimeOnWeek;  
import com.hescha.teacher\_workload\_accounting.entity.Lesson;  
import com.hescha.teacher\_workload\_accounting.entity.Subject;  
import com.hescha.teacher\_workload\_accounting.entity.Teacher;  
import com.hescha.teacher\_workload\_accounting.service.ClassesService;  
import com.hescha.teacher\_workload\_accounting.service.ClassesSubjectTimeOnWeekService;  
import com.hescha.teacher\_workload\_accounting.service.SubjectService;  
import com.hescha.teacher\_workload\_accounting.service.TeacherService;  
  
@Controller  
@RequestMapping("/generation")  
public class GenerationController {  
  
 @Autowired  
 private ClassesSubjectTimeOnWeekService classesSubjectTimeOnWeekService;  
  
 @Autowired  
 private SubjectService subjectService;  
  
 @Autowired  
 private ClassesService classesService;  
  
 @Autowired  
 private TeacherService teacherService;  
  
 int iter;  
  
 List<Subject> subjects;  
 List<Teacher> teachers;  
 List<Classes> classes;  
 List<ClassesSubjectTimeOnWeek> subjectTimeOnWeeks;  
 // день недели  
 // класс  
 // урок  
 Lesson[][][] lessons = new Lesson[5][11][7];  
  
 Random rand = new Random();  
  
  
 @GetMapping  
 public String getList(Model model) {  
  
 readAllToGeneration();  
 readAllToGeneration();  
 makeList();  
  
 model.addAttribute("list", lessons);  
 model.addAttribute("message1", "Шагов генерации - " + iter);  
 model.addAttribute("message2",  
 "Осталось предметов " + subjectTimeOnWeeks.size()  
 + " (" + subjectTimeOnWeeks + ")");  
 return "generation";  
 }  
  
 private void readAllToGeneration() {  
 lessons = new Lesson[5][11][7];  
 subjects = subjectService.readAll();  
 teachers = teacherService.readAll();  
 classes = classesService.readAll();  
 subjectTimeOnWeeks = classesSubjectTimeOnWeekService.readAll();  
 }  
  
  
 private void makeList() {  
 iter = 0;  
 while (iter < 1\_000\_000 && subjectTimeOnWeeks.size() > 0) {  
 iter++;  
  
 // получаем случайное требование  
 int randomLessonConditionId = rand.nextInt(subjectTimeOnWeeks.size());  
 ClassesSubjectTimeOnWeek subjectTimeOnWeek = subjectTimeOnWeeks.get(randomLessonConditionId);  
  
 // получаем случайный день недели  
 int randomDayOfWeekNumber = rand.nextInt(lessons.length);  
 Lesson[][] randomDay = lessons[randomDayOfWeekNumber];  
  
 // получаем все уроки класса  
 Lesson[] classLessons = randomDay[subjectTimeOnWeek.getClasses().getNumber() - 1];  
 // получаем случайный номер урока по счету выбранного класса в выбранный день  
 int lessonOnDayNumber = rand.nextInt(classLessons.length);  
 Lesson lesson = classLessons[lessonOnDayNumber];  
 // если место не свободно - пропустить  
 if (lesson != null) {  
 continue;  
 }  
  
 // если свободно - выбрать препода, проверить на совместимость  
 Teacher teacher = getTeacherBySubject(subjectTimeOnWeek);  
 // проверяем, нет ли в этот же день у других классов с этим же преподом урока в  
 // это же время  
 boolean canSetLesson = checkPossibleSetLesson(randomDay, lessonOnDayNumber, teacher);  
 if (!canSetLesson) {  
 continue;  
 }  
  
 //создаем урок  
 Lesson newLesson = createLesson(subjectTimeOnWeek, randomDayOfWeekNumber, lessonOnDayNumber,  
 teacher);  
  
 addLessonToArray(randomLessonConditionId, subjectTimeOnWeek, randomDayOfWeekNumber,  
 lessonOnDayNumber, newLesson);  
 }  
  
 }  
  
 private Teacher getTeacherBySubject(ClassesSubjectTimeOnWeek subjectTimeOnWeek) {  
 Set<Teacher> subjectTeachers = subjectTimeOnWeek.getSubject().getTeachers();  
 Object[] array = subjectTeachers.toArray();  
 Teacher teacher = (Teacher) array[rand.nextInt(array.length)];  
 return teacher;  
 }  
  
 private void addLessonToArray(int randomLessonConditionId,  
 ClassesSubjectTimeOnWeek subjectTimeOnWeek,  
 int randomDayOfWeekNumber, int lessonOnDayNumber, Lesson newLesson) {  
 lessons[randomDayOfWeekNumber][subjectTimeOnWeek.getClasses().getNumber()  
 - 1][lessonOnDayNumber] = newLesson;  
  
 subjectTimeOnWeek.setHoursInWeek(subjectTimeOnWeek.getHoursInWeek() - 1);  
 if (subjectTimeOnWeek.getHoursInWeek() == 0) {  
 subjectTimeOnWeeks.remove(randomLessonConditionId);  
 }  
 }  
  
 private Lesson createLesson(ClassesSubjectTimeOnWeek subjectTimeOnWeek, int randomDayOfWeekNumber,  
 int lessonOnDayNumber, Teacher teacher) {  
 Lesson newLesson = new Lesson();  
 newLesson.setClasses(subjectTimeOnWeek.getClasses());  
 newLesson.setTeacher(teacher);  
 newLesson.setDay(randomDayOfWeekNumber);  
 newLesson.setTime(lessonOnDayNumber);  
 newLesson.setSubject(subjectTimeOnWeek.getSubject());  
 return newLesson;  
 }  
  
 private boolean checkPossibleSetLesson(Lesson[][] randomDay, int lessonOnDayNumber,  
 Teacher teacher) {  
 for (Lesson[] lessonAllClassesInCurrentDay : randomDay) {  
 if (lessonAllClassesInCurrentDay[lessonOnDayNumber] != null  
 && lessonAllClassesInCurrentDay[lessonOnDayNumber].getTeacher().equals(teacher)) {  
 return false;  
 }  
 }  
 return true;  
 }  
  
  
}

package com.hescha.teacher\_workload\_accounting.entity;  
  
import java.util.Objects;  
  
import javax.persistence.CascadeType;  
import javax.persistence.Entity;  
import javax.persistence.JoinColumn;  
import javax.persistence.ManyToOne;  
  
import lombok.Data;  
  
@Data  
@Entity  
public class ClassesSubjectTimeOnWeek extends AbstractEntity {  
  
 @ManyToOne(cascade = CascadeType.*DETACH*)  
 @JoinColumn(name = "classes\_id")  
 private Classes classes;  
  
 @ManyToOne(cascade = CascadeType.*DETACH*)  
 @JoinColumn(name = "subject\_id")  
 private Subject subject;  
  
 private int hoursInWeek;  
  
 @Override  
 public String toString() {  
 return "класс " + classes.getNumber() + ", предмет " + subject.getName() + ", часов "  
 + hoursInWeek;  
 }  
  
 @Override  
 public boolean equals(Object obj) {  
 if (this == obj) {  
 return true;  
 }  
 if (!super.equals(obj)) {  
 return false;  
 }  
 if (getClass() != obj.getClass()) {  
 return false;  
 }  
 ClassesSubjectTimeOnWeek other = (ClassesSubjectTimeOnWeek) obj;  
 return Objects.*equals*(classes, other.classes) && hoursInWeek == other.hoursInWeek  
 && Objects.*equals*(subject, other.subject);  
 }  
  
 @Override  
 public int hashCode() {  
 final int prime = 31;  
 int result = super.hashCode();  
 result = prime \* result + Objects.*hash*(classes, hoursInWeek, subject);  
 return result;  
 }  
  
}

package com.hescha.teacher\_workload\_accounting.service;  
  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import com.hescha.teacher\_workload\_accounting.entity.ClassesSubjectTimeOnWeek;  
import com.hescha.teacher\_workload\_accounting.repository.ClassesSubjectTimeOnWeekRepository;  
  
  
@Service  
public class ClassesSubjectTimeOnWeekService extends CrudImpl<ClassesSubjectTimeOnWeek> {  
  
 public ClassesSubjectTimeOnWeekRepository repository;  
  
 @Autowired  
 public ClassesSubjectTimeOnWeekService(ClassesSubjectTimeOnWeekRepository repository) {  
 super(repository);  
 this.repository = repository;  
 }  
  
}

<!DOCTYPE html>  
<html lang="en" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <title>Генерация</title>  
 <div th:replace="~{commons :: head}"></div>  
</head>  
<body>  
  
<div th:replace="~{commons :: nav}"></div>  
  
  
<section>  
 <div class="container">  
 <div class="row">  
 <div class="col">  
 <h3 th:text="${message1}"></h3>  
 <h3 th:text="${message2}"></h3>  
 <div id="accordion">  
 <div class="card" th:each="arr2, i:${list}">  
 <div class="card-header">  
 <h5 class="mb-0">  
 <button class="btn btn-link" data-toggle="collapse"  
 th:data-target="'#collapseOne'+${i.index}" aria-expanded="true"  
 th:aria-controls="'collapseOne'+${i.index}"  
 th:text="'День недели: ' + ${i.index+1}">Разворачиваемая панель #1  
 </button>  
 </h5>  
 </div>  
  
 <div th:id="'collapseOne'+${i.index}" class="collapse"  
 aria-labelledby="headingOne" data-parent="#accordion">  
 <div class="card-body">  
 <table class="table table-striped table-responsive-md"  
 th:each="arr:${arr2}">  
 <thead>  
 <tr>  
 <th>Класс</th>  
 <th>Номер урока</th>  
 <th>Предмет и учитель</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr th:each="entity : ${arr}">  
 <div th:if="${entity!=null}">  
 <td th:text="${entity.classes}"></td>  
 <td th:text="${entity.time+1}"></td>  
 <td  
 th:text="${entity.subject.name}+' ('+${entity.teacher.name}+')'"></td>  
 </div>  
 </tr>  
 </tbody>  
 </table>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
</section>  
  
</body>  
</html>