Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

Югорский государственный университет

Институт (НОЦ) технических систем и информационных технологий

КУРСОВОЙ ПРОЕКТ

по дисциплине: «Разработка программного обеспечения»

(направление подготовки 09.03.04 Программная инженерия)

на тему: Информационная система «Электронный журнал»

Группа 1541б

Студент: Чернавский А.Ю. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Руководитель: Гусаков А. В. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Отметка о защите: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

Ханты-Мансийск 2017 г.

# Содержание

[Содержание 2](#_Toc501606213)

[Введение 3](#_Toc501606214)

[1. Методологии разработки 6](#_Toc501606215)

[2. Средства разработки 8](#_Toc501606216)

[3. Описание системы 11](#_Toc501606217)

[3.1 Функциональные блоки 11](#_Toc501606218)

[3.2 Описание алгоритма 12](#_Toc501606219)

[3.2.1 Сохранение и изменение данных в БД 15](#_Toc501606220)

[3.2.2 Поиск данных 17](#_Toc501606221)

[3.2.3 Удаление данных 18](#_Toc501606222)

[3.3 Структура БД 19](#_Toc501606223)

[3.4 Интерфейс системы 21](#_Toc501606224)

[4. Тестирование 26](#_Toc501606225)

[4.1 Unit тесты 27](#_Toc501606226)

[Заключение 29](#_Toc501606227)

[Список использованных источников 30](#_Toc501606228)

[Приложение А 31](#_Toc501606229)

[Приложение Б 32](#_Toc501606230)

[Приложение В 34](#_Toc501606231)

[Приложение Г 42](#_Toc501606232)

# Введение

В высших учебных заведениях (и не только в высших) всегда будет актуальным вопрос об автоматизации системы учета посещаемости студентов (сотрудников), а также переносе бумажных журналов в электронный, наглядное оформление статистики, избавление от лишней бумажной работы. В этих условиях возрастает роль обработки информации в электронном виде, а также повышаются требования к автоматизированным информационным системам учета посещаемости «Электронный журнал».

Внедрение автоматизированных технологий системы «Электронного журнала» позволяет повысить эффективность управления организацией за счет обеспечения руководителей, преподавателей, сотрудников учебной части полной, оперативной и достоверной информацией.

Поскольку система посещаемости не претерпевает сильных изменений от группы к группе, то можно создать унифицированный инструмент для большинства пользователей с распределением ролей преподавателей, студентов.

В данном курсовом проекте рассматривается Югорский государственный университет как объект для разработки системы по ведению учета посещаемости «Электронный журнал».

В Югорском государственном университете внедрены различные системы, такие системы, как:

* Корпоративная информационная система класса ERP Галактика 9.1. «Управление учебным процессом».
* Многочисленные разработанные внутренние сервисы (Информационно-справочная система, Сервисы научной библиотеки и т.д.).

Данные сервисы могли бы предоставлять информацию для ИС «Электронный журнал», поскольку необходима унифицированная система учета посещаемости, которая бы интегрировала в себя необходимые данные с описанных сервисов и позволяла бы вести учет посещаемости, а также собирать различную статистику посещаемости.

С каждым днем всё большее количество систем разрабатываются в виде веб-сервисов по той причине, что они имеют ряд преимуществ перед остальными. Во-первых, доступ к веб-сервису не требует установки на компьютер, достаточно лишь наличие подключения в интернет. Во-вторых, работать можно под любой операционной системой (Windows, MacOS, Linux и др.). Необходим только установленный браузер последних версий. В-третьих, веб-системы фактически не требовательны к ресурсам пользователей и не предъявляют никаких требований к аппаратной платформе пользователя.

Также веб-системы позволяют пользователям быть по-настоящему мобильными: они позволяют работать, сохранять результаты своей работы и, в случае необходимости, иметь к ним доступ с любого компьютера, где имеется доступ в Интернет.

В связи с этим можно сказать, что разрабатываемая система должна быть создана именно в виде веб-сервиса, что позволит вовлечь максимальное число обучающихся университета и преподавателей университета, поскольку является мобильной и легковесной, что актуально для при учете посещаемости во время занятий.

Целью данного курсового проекта является разработка ИС «Электронный журнал», позволяющей вести учет посещаемости студентов Югорского государственного университета с максимальной вовлеченностью всех преподавателей образовательного учреждения.

Для достижения данной цели необходимо решить следующие задачи:

* Определение методологии разработки;
* Определение паттерна для разработки системы;
* Выбор средств для разработки системы;
* Реализация ролевой политики системы;
* Реализация модуля администратора для управления ролевой политикой системы
* Реализация модуля преподавателя для учета посещаемости студентов
* Реализация модуля студента для отображения статистики
* Проведение тестирования с целью проверки соответствию требованиям.

# Методологии разработки

В качестве методологии разработки была выбрана итеративная модель.[1]

Итеративная модель жизненного цикла не требует для начала полной спецификации требований. Вместо этого, создание начинается с реализации части функционала, становящейся базой для определения дальнейших требований. Этот процесс повторяется.

В несколько упрощенном виде, итеративная модель состоит из четырех основных стадий, которые повторяются в каждой из итераций:

* определение и анализ требований;
* дизайн и проектирование – согласно требованиями;
* разработка и тестирование – кодирование, интеграция и тестирование нового компонента;
* фаза ревью – оценка, пересмотр текущих требований и предложения дополнений к ним.

Разработка велась с учетом данных стадий. Вначале были получены и проанализированы требования к приложению, затем разрабатывался дизайн пользовательского интерфейса. После завершения данной стадии велась работа над итеративной реализацией функционала (связь с БД, работа с БД).

В процессе разработки, также, использовался шаблон проектирования MVC.

Model-view-controller (MVC) - схема использования нескольких [шаблонов проектирования](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), с помощью которых модель приложения, [пользовательский интерфейс](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81) и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные.

Основная цель применения этой концепции состоит в отделении [бизнес-логики](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%B0) модели от её визуализации. За счёт такого разделения повышается возможность [повторного использования](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B2%D1%82%D0%BE%D1%80%D0%BD%D0%BE%D0%B5_%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5).

Концепция MVC позволяет разделить данные (модель), представление и обработку действий (производимую контроллером) пользователя на три отдельных компонента:

Модель – предоставляет данные и методы работы с этими данными; реагирует на запросы, изменяя своё состояние; не содержит информации, как эти данные можно визуализировать.

Представление - отвечает за визуализацию информации.

Контроллер – обеспечивает связь между пользователем и системой: контролирует ввод данных пользователем и использует модель и представление для реализации необходимой реакции. [2]

Данный шаблон проектирования в полной мере можно проследить по структуре проекта.

Пакеты Entity и Services содержат классы-модели и классы-сервисы (реализующие бизнес логику приложения) соответственно предоставляющие данные и методы работы с этими данными.

Созданы классы контроллеры для взаимодействия с пользователем.

В качестве архитектурного стиля использован стиль REST для взаимодействия компонентов распределенной системы.[3]

Визуализация польщовательской информации в компоненте представления было сформировано средствами HTML и специальными библиотеками:

- использованы файлы стилей CSS, а также подключена библиотека

Materialized, позволяющая создать простой, минималистичный и приятный польщовательский интерфейс.

- использована технология JavaScript с использованием js библиотеки jQuery и некоторых средств js Materialized (такие как datePicker для выбора дат), основанных на данной библиотеке, а также технология ajax для обращения к серверу.

# Средства разработки

В качестве средств разработки выбран язык программирования Java и [интегрированная среда разработки программного обеспечения](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8) IntelliJ IDEA.

Java - строго типизированный [объектно-ориентированный язык программирования](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), разработанный компанией [Sun Microsystems](https://ru.wikipedia.org/wiki/Sun_Microsystems). Приложения Java обычно [транслируются](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D1%8F%D1%82%D0%BE%D1%80) в специальный [байт-код](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82-%D0%BA%D0%BE%D0%B4), поэтому они могут работать на любой компьютерной архитектуре, с помощью [виртуальной Java-машины](https://ru.wikipedia.org/wiki/Java_Virtual_Machine).[4]

IntelliJ IDEA - [интегрированная среда разработки программного обеспечения](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8) на многих языках программирования, в частности [Java](https://ru.wikipedia.org/wiki/Java), [JavaScript](https://ru.wikipedia.org/wiki/JavaScript), [Python](https://ru.wikipedia.org/wiki/Python) и включающая поддержку всех последних технологий и фреймворков, разработанная компанией [JetBrains](https://ru.wikipedia.org/wiki/JetBrains).[5]

Приложение разрабатывалось на платформе Java EE.

Java EE (Enterprise Edition) представляет собой широко используемую платформу, содержащую набор взаимосвязанных технологий, которые существенно сокращают стоимость и сложность разработки, развертывания многоуровневых серверных приложений, а также управления ими. Платформа Java EE основана на платформе Java SE и предоставляет набор интерфейсов API (интерфейсов разработки приложений) для разработки и запуска портируемых, надежных, масштабируемых и безопасных серверных приложений.

Java EE в числе прочих содержит следующие компоненты:

* Enterprise JavaBeans (EJB): управляемая серверная архитектура компонентов, используемая для инкапсуляции бизнес-логики приложения. Технология EJB позволяет осуществлять быструю и упрощенную разработку распределенных, транзакционных, безопасных и переносимых приложений, основанных на технологии Java.
* Интерфейс API сохранения состояния Java (Java Persistence API, JPA): инфраструктура, позволяющая разработчикам управлять данными с помощью объектно-реляционного сопоставления (ORM) в приложениях, созданных на платформе Java.

В процессе разработки был использован фреймворк Hibernate.

Hibernate - библиотека для языка программирования [Java](https://ru.wikipedia.org/wiki/Java), предназначенная для решения задач объектно-реляционного от отображения. Целью Hibernate является освобождение разработчика от значительного объёма сравнительно низкоуровневого программирования при работе в объектно-ориентированных средствах в реляционной базе данных. Библиотека не только решает задачу связи классов Java с таблицами базы данных, но и также предоставляет средства для автоматической генерации и обновления набора таблиц, построения запросов и обработки полученных данных и может значительно уменьшить время разработки, которое обычно тратится на ручное написание [SQL](https://ru.wikipedia.org/wiki/SQL)- и [JDBC](https://ru.wikipedia.org/wiki/JDBC)-кода.[6]

В качестве средства для работы с БД использована стандартная среда для работы с базой данных MySQL – MySQL Workbench CE 6.3.

MySQL Workbench - инструмент для визуального проектирования [баз данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), интегрирующий проектирование, моделирование, создание и эксплуатацию БД в единое бесшовное окружение для системы баз данных MySQL, предоставляет разработчику следующие возможности:

* Позволяет наглядно представить модель базы данных в графическом виде.
* Наглядный и функциональный механизм установки связей между таблицами, в том числе «многие ко многим» с созданием таблицы связей.
* Reverse Engineering — восстановление структуры таблиц из уже существующей на сервере БД
* Удобный редактор SQL запросов, позволяющий сразу же отправлять их серверу и получать ответ в виде таблицы.
* Возможность редактирования данных в таблице в визуальном режиме.

Для работы с клиенсткой частью представления был использован удобный текстовый редактор SublimeText3.

Для разработки прототипов пользовательского интерфейса была использовано программное обеспечение для разработки прототипов WireFrameSketcher.

Для тестирования запросов ситсемы REST в системы было использовано программное обеспечение тестирования запросов POSTMAN

В качестве системы контроля версий был выбран веб-хостинг GitHub и клиент GIT для настольных компьютеров Source Tree.

GitHub — крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки. Основан на системе контроля версий Git и разработан на Ruby on Rails.

Сервис абсолютно бесплатен для проектов с открытым исходным кодом.[7]

Source Tree предоставляет удобный интерфейс для работы с репозиторием.

# Описание системы

## 3.1 Функциональные блоки

Разрабатываемую систему учета посещаемости можно разбить на следующие функциональные блоки (Рисунок 1).

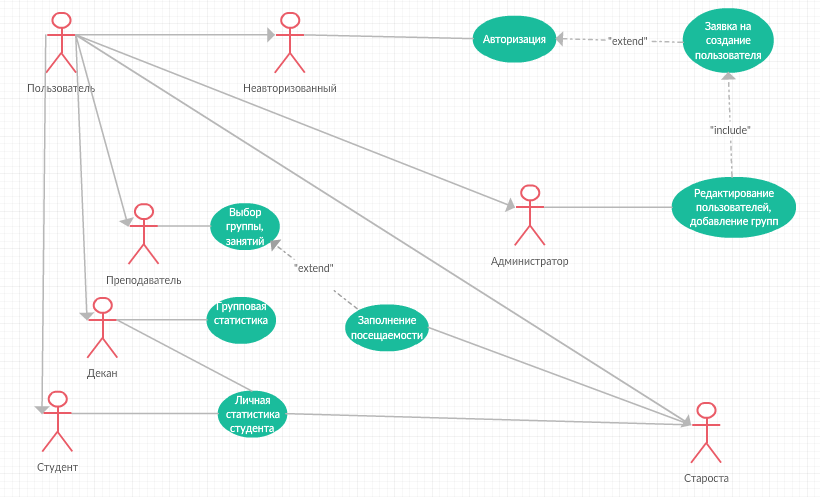


Рисунок 1. Диаграмма вариантов использования

В системе определены роли пользователей. Также, определены прецеденты пользователей, основным из коорых будет заполнение посещаемости, доступный старосте и преподавателю группы.

## 3.2 Описание алгоритма

Система состоит из классов пяти типов: классы-сущности, классы-dto для передачи информации в компонент представления, классы-контроллеры, классы-сервисы, интерфейсы-репозитории.

Также, файлы пользоватлеьского представления (структура – html, стили отображения – css, логика отображения, анимация – js файлы).

С помощью возможностей Hibernate сгенерированы классы-сущностей с таблицы БД. В них описаны стандартные методы get, set, equals, toString, конструктор и пр. при помощи плагина, используещего библиотеку Lombok для форматирования, структуризации, улучшения читаемости и отделения постоянной избыточной части кода от основной, определяющей логику (Рисунок 2).

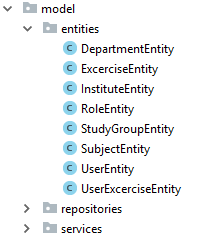


Рисунок 2 – Entity

Код основной сущности пользователя, для примера, вынесен в приложение А.

Для каждого класса сущности реализованы классы-сервисы на основе интерфейсов-репозиториев, которые наследются от репозитория CRUD (Create, Read, Update, Delete), определяющего основные методы работы с сущностью.

Сервисы необходимы для взаимодействия с БД. В данных классах описаны методы save (create/update), delete, find (read) (Рисунок 3).

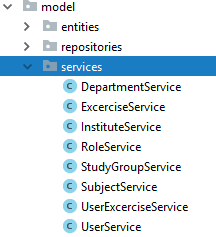


Рисунок 3 - Services

Код сервиса для сущности польщователя вынесен в приложение Б

Созданы классы контроллеры для взаимодействия с пользователем. В них содержатся методы-обработчики событий и отрисовка форм.

На данном этапе разработки реализованы классы для администратора и преподавателя (Рисунок 4).



Рисунок 4 - Controller

Код контроллеров вынесед в приложение В.

К файлам польщовательского интерфейса можно отнести файлы html, css, js, а также шрифты библиотеки materialized (Рисунок 5)

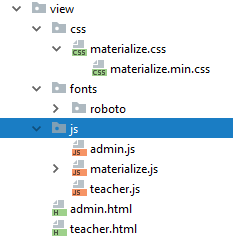


Рисунок 5 - Файлы представления пользователя

Система использует три основных алгоритма: сохранение и изменение данных в БД, поиск данных в БД, удаление данных из БД.

### 3.2.1 Сохранение и изменение данных в БД

Алгоритм сохранения изменеия данных в БД представлен на блок схеме ниже. (Рисунок 6)

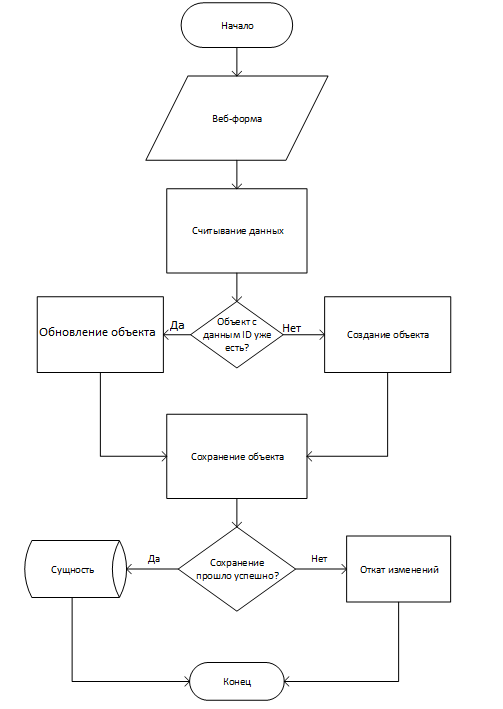


Рисунок 6. Блок схема сохранения

Данный алгоритм выполняет следующие действия:

1. считывает данные с веб-формы;
2. проверяет есть ли объект с данным ID;
3. создает объект класса на основе полученных данных;
4. с помощью метода save() созданный объект сохраняется (обновляется) в БД;
5. если сохранение прошло успешно выводит сохраненный объект.

### 3.2.2 Поиск данных

Алгоритм поиска данных представлен на блок схеме ниже. (Рисунок 3.2.2.1)



Рисунок 3.2.2.1. Блок схема

Данный алгоритм выполняет следующие действия:

1. считывает данные с веб-формы;
2. проверяет есть ли объект с данным ID;
3. с помощью метода findById() получает объект из БД;
4. если поиск прошел успешно выводит полученный объект.

### 3.2.3 Удаление данных

Алгоритм удаления данных представлен на блок схеме ниже. (Рисунок 3.2.3.1)



Рисунок 3.2.3.1. Блок схема

Данный алгоритм выполняет следующие действия:

1. считывает данные с веб-формы;
2. проверяет есть ли объект с данным ID;
3. с помощью метода delete() удаляет объект из БД;

## 3.3 Структура БД

Характеристика отношений и взаимосвязь между основными группами пользователей – преподавателей и студентов – следующая:

(Характеристика связей, прототип БД системы)

- У одного преподавателя могут проходить занятия у различных групп.

- Один преподаватель может вести несколько занятий у одной группы по различным учебным дисциплинам.

- По одной учебной дисциплине разные виды занятий могут вести различные преподаватели. Как правило, имеется основной, который отвечает полностью за дисциплину.

- У группы проводят занятия различные преподаватели.

- Пользователь как студент, в рамках учебного процесса, закреплен за одной группой.

- Пользователь может иметь несколько ролей в системе, например, как студент-магистрант и как преподаватель.

Для хранения данных была спроектирована модель БД, физическая структура БД приведена на Рисунке 7.

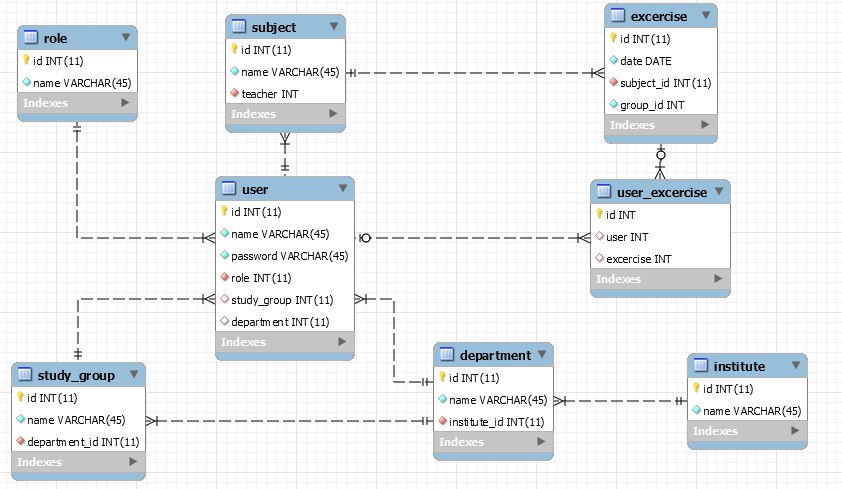


Рисунок 7. Физическая структура БД

Составлено 8 таблиц для хранения данных о пользователях и данных учета посещаемости. Описание ключевых таблиц БД приведено в Таблице 1.

|  |  |
| --- | --- |
| Таблица 1. Описание ключевых таблиц БД | |
| Наименование таблицы | Описание |
| user | Таблица хранит данные о пользователе (студент/преподаватель): имя, пароль, роль в системе, группу (для студентов), кафедру (для преподавателей) |
| Exercise | Таблица хранит данные о занятиях: дата занятия, предмет, группа |
| User\_excercise | Таблица «многие ко многим» сопоставляет данные о студентах и занятиях: хранит ссылку на студента и на занятие (фактически, учет посещаемости) |

## 3.4 Интерфейс системы

Приложение имеет минималистичный интерфейс простой интерфейс.

На данном этапе разработаны представления для двух ролей – администратора и преподавателя.

Для разработки интерфейса были составлены прототипы, представленные на рисунках 8 и 9.

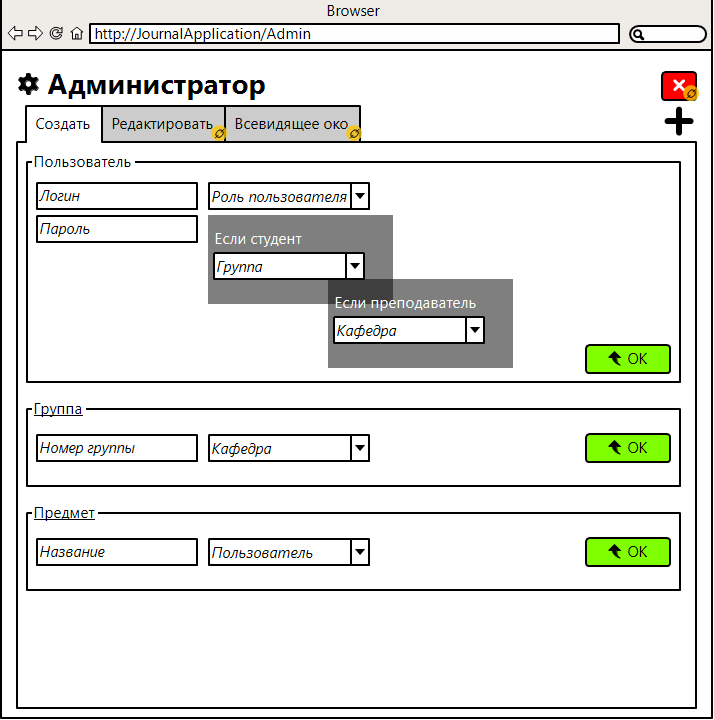


Рисунок 8 – Прототип страницы администратора

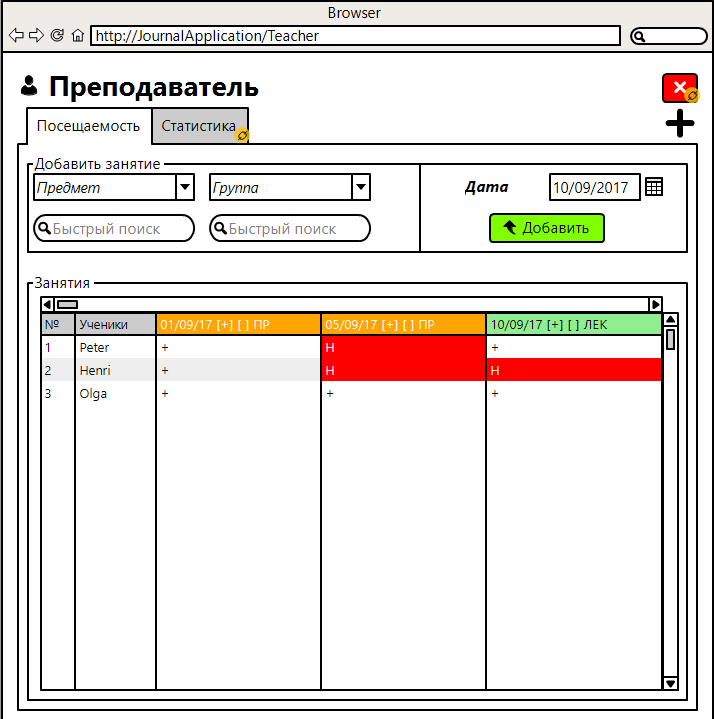


Рисунок 9 – Прототип страницы преподавателя

Форма администратора представлена на рисунке 10.

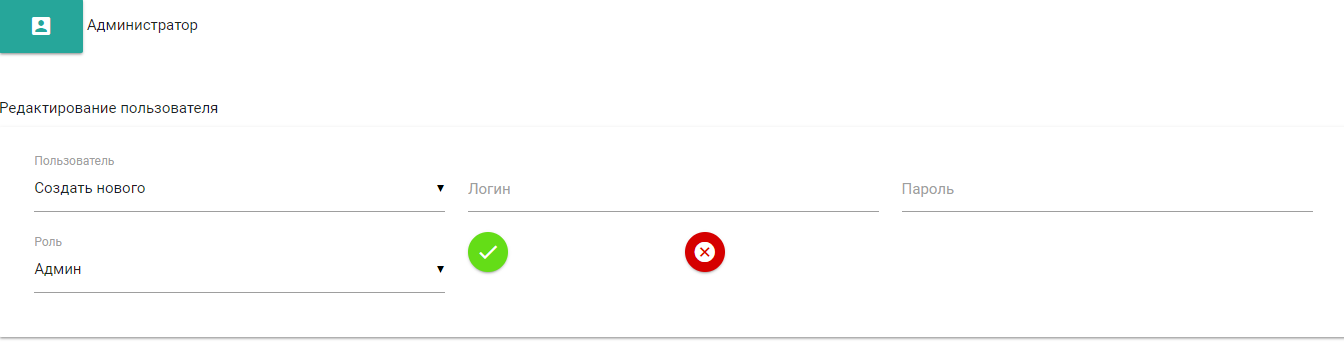


Рисунок 10 – Форма администратора

При первоначальной загрузке формы выбраны опции для создания пользователя. Загружены существующие пользователи в первый выпадающий список и роли в список ниже.

При создании пользователя указываем имя (оно е явлется в данном случае логином), пароль и роль в системе.

Если роль Администратор (админ), то польщователя можно сохранить.

При выборе роли преподавателя, старосты или студента появится дополнительный список: для студентов – выбор группы, для преподавателя – выбор кафедры.

Создание студента представлено на рисунке 11.

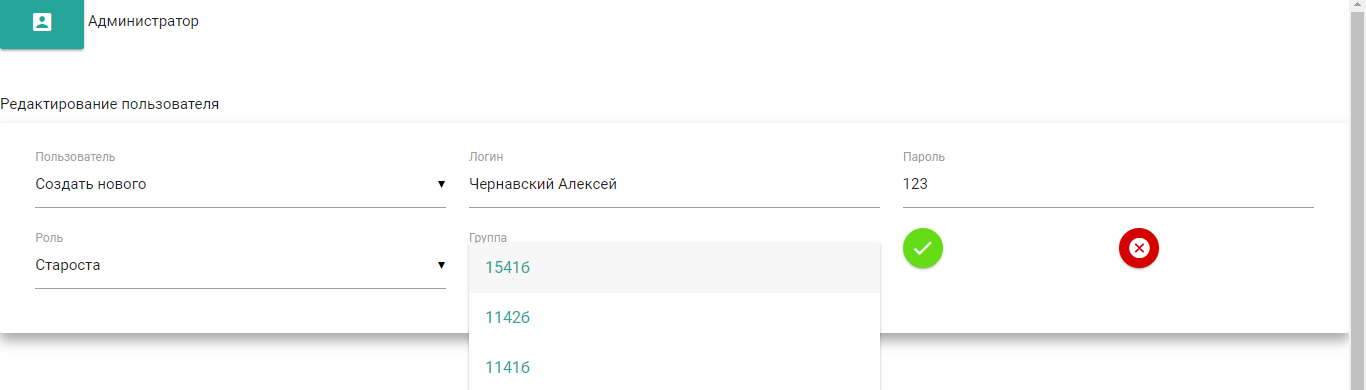


Рисунок 11 – создание пользователя

Для создания нужно подтвердить изменения зеленой кнопкой на польщовательском интерфейсе.

После создания данного пользователя можно выбрать в первом выпадающем списке и изменить его параметры (обновление) либо удалить нажатием на красную кнопку.

Перейдем к функционалу формы преподавателя, представленной на рисунке 12.

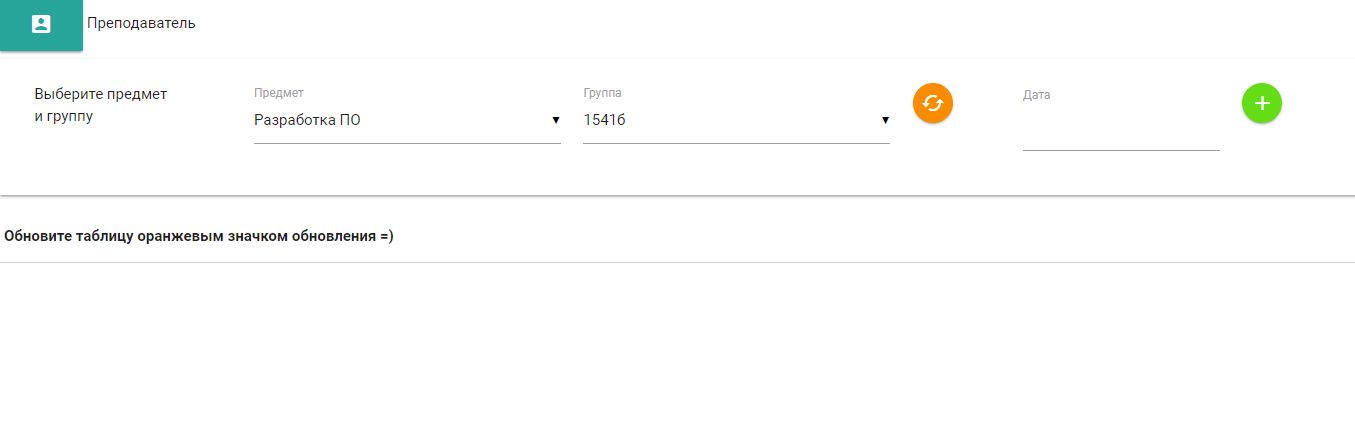


Рисунок 12 – форма преподавателя

Для преподавателя предполагается отображение его предметов в списке предметов и список групп, у которых проходит данное занятие. После это, на основе выбранных данных формируется таблица посещаемости по датам со списком студентов (Рисунок 13).

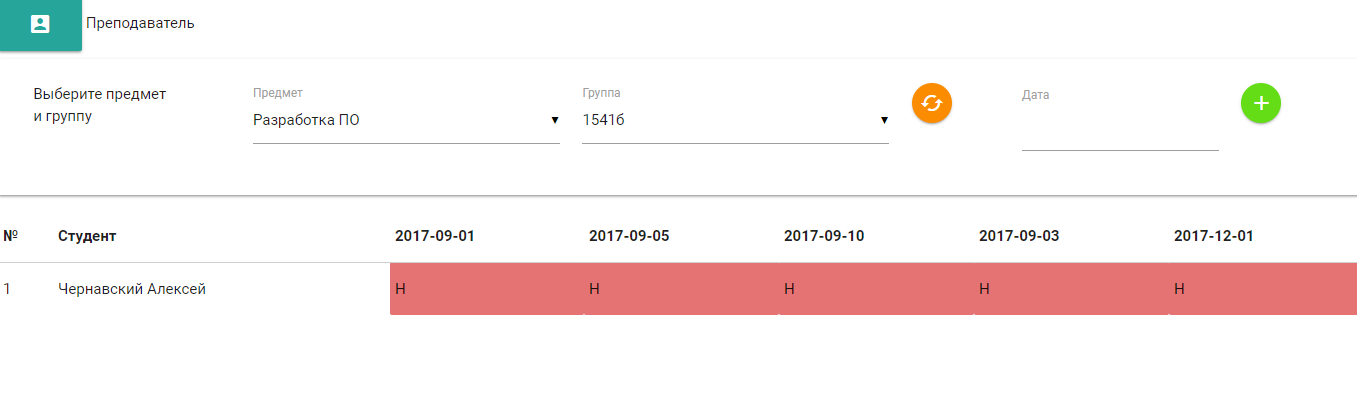


Рисунок 13 – таблица посещаемости

Поскольку судентов в нашей системе небыло и мы добавили 1 студента, в занятиях у него автоматически проставились пропуски.

Преподаватель может добавить занятие через выбор даты в поле справа и нажатия зеленой кнопки для добавления (рисунок 14).

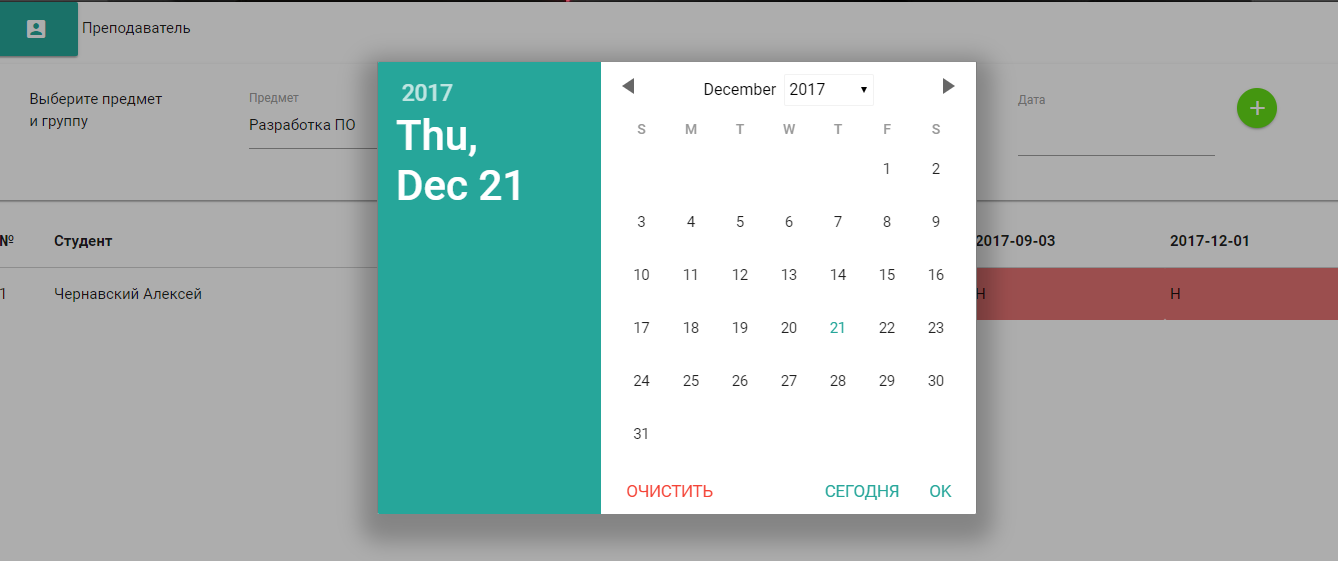


Рисунок 14 – добавление даты

Отмечать посещаемость преподаватель может при помощи мыши, щелкая на нужные поля пересечения студента и занятия для изменения типа отметки посещаемости (рисунок 15).

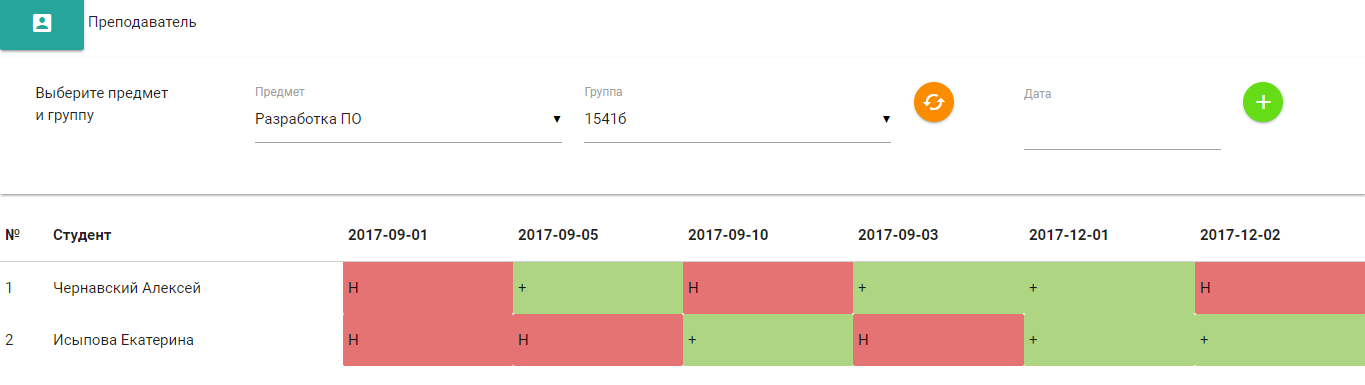


Рисунок 15 – учет посещаемости

# Тестирование

Тестирование программного обеспечения - процесс исследования, испытания [программного продукта](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), имеющий две различные цели:

* продемонстрировать разработчикам и заказчикам, что программа соответствует требованиям;
* выявить ситуации, в которых поведение программы является неправильным, нежелательным или не соответствующим спецификации.

Было проведено тестирование unit-тестами.

## 4.1 Unit тесты

Модульное тестирование ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) unit testing) — процесс в [программировании](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), позволяющий проверить на корректность единицы [исходного кода](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4), наборы из одного или более программных модулей вместе с соответствующими управляющими данными, процедурами использования и обработки.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к [регрессии](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%81%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Для тестирования был использован метод Arrange-Act-Assert (AAA). Суть его заключается в том, чтобы в модульном тестечетко определить предусловия (иициализация тестовых данных, предварительные установки), действия (то, что тестируется) и постусловия (что должно быть в результате выполненных действий).

Были написан unit тест для СЕРВИСА ПОЛЬЗОВАТЕЛЯ, поскольку остальные имеют подобные методы.

Пример теста метода findById(), в результате которого этот метод должен получить объект класса UserEntity приведен на рисунке 16. Исходный код приведен в Приложении Г.

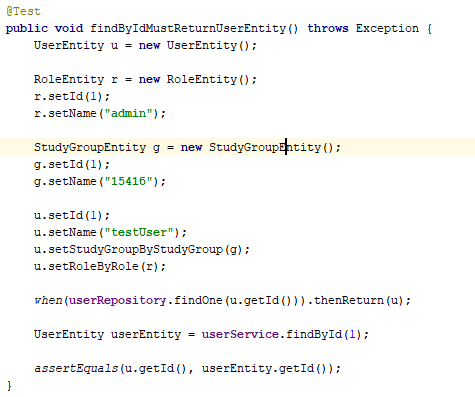


Рисунок 16 - unit тест

# Заключение

В ходе выполнения курсового проекта была спроектирована и разработана информационная система «Электронный журнал» c базовым функционалом, позволяющим работать с БД.

Были изучены теоретические концепции и разработки, связанные с заданием курсовой работы.

В процессе выполнения первой части курсового проекта была формализована предметная область, проведен анализ моделируемой системы, сформированы требования.

В процессе выполнения второй части курсового проекта был реализован пользовательский интерфейс системы и реализован основной функционал.

Выполнение курсового проекта помогло закрепить материал и навыки, полученные на лекциях и лабораторных работах, помогло систематизировать знания по проектированию и разработке ПО и научится применять их к конкретным задачам.

# Список использованных источников

1. Итеративная модель (Iterative model) [Электронный ресурс]. - Режим доступа: <http://qalight.com.ua/baza-znaniy/iterativnaya-model-iterative-model/>
2. Model-view-controller [Электронный ресурс]. - Режим доступа: <https://ru.wikipedia.org/wiki/Model-View-Controller>
3. REST-архитектура [Электронный ресурс]. - Режим доступа: https://ru.wikipedia.org/wiki/REST
4. Java [Электронный ресурс]. - Режим доступа: <https://ru.wikipedia.org/wiki/Java>
5. IntelliJ IDEA [Электронный ресурс]. - Режим доступа: <https://ru.wikipedia.org/wiki/IntelliJ_IDEA>
6. Hibernate [Электронный ресурс]. - Режим доступа: <https://ru.wikipedia.org/wiki/Hibernate_(%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0)>
7. .GitHub [Электронный ресурс]. - Режим доступа: https://ru.wikipedia.org/wiki/GitHub

# Приложение А

Листинг класса UserEntity

**package** ru.ugrasu.journal.model.entities;  
  
**import** lombok.\*;  
  
**import** javax.persistence.\*;  
**import** java.util.List;  
  
@Data  
@Entity  
@Table(name = **"user"**, schema = **"journal"**, catalog = **""**)  
**public class** UserEntity {  
 @Id@Column(name = **"id"**)  
 **private int id**;  
 @Basic@Column(name = **"name"**)  
 **private** String **name**;  
 @Basic@Column(name = **"password"**)  
 **private** String **password**;  
 @OneToMany(mappedBy = **"userByTeacher"**)  
 **private** List<SubjectEntity> **subjectsById**;  
 @ManyToOne@JoinColumn(name = **"role"**, referencedColumnName = **"id"**, nullable = **false**)  
 **private** RoleEntity **roleByRole**;  
 @ManyToOne@JoinColumn(name = **"study\_group"**, referencedColumnName = **"id"**)  
 **private** StudyGroupEntity **studyGroupByStudyGroup**;  
 @ManyToOne@JoinColumn(name = **"department"**, referencedColumnName = **"id"**)  
 **private** DepartmentEntity **departmentByDepartment**;  
 @OneToMany(mappedBy = **"userByUser"**)  
 **private** List<UserExcerciseEntity> **userExcercisesById**;  
  
}

# Приложение Б

Листинг класса UserService

**package** ru.ugrasu.journal.model.services;  
  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.stereotype.Component;  
**import** ru.ugrasu.journal.exception.NotFoundException;  
**import** ru.ugrasu.journal.model.entities.UserEntity;  
**import** ru.ugrasu.journal.model.repositories.UserExcerciseRepository;  
**import** ru.ugrasu.journal.model.repositories.UserRepository;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
@Component  
**public class** UserService {  
  
 @Autowired  
 **private** UserRepository **userRepository**;  
  
 **public** List<UserEntity> findAll() {  
 **return userRepository**.findAll();  
 }  
  
 **public** List<UserEntity> findUserByGroupId(**int** id) {  
 List<UserEntity> listUserEntity = findAll();  
 List<UserEntity> newListUserEntity = **new** ArrayList<>();  
  
 listUserEntity.forEach(userEntity -> {  
 *//Проверка роли (студент или староста)* String role = userEntity.getRoleByRole().getName();  
  
 **if** (role.equals(**"Староста"**) || role.equals(**"Студент"**)) {  
  
 *//Проверка группы (ТОЛЬКО У СТУДЕНТОВ)* **int** groupId = userEntity.getStudyGroupByStudyGroup().getId();  
  
 **if** (id == groupId) {  
 newListUserEntity.add(userEntity);  
 }  
 }  
 });  
  
 **return** newListUserEntity;  
 }  
  
 **public** UserEntity findById(**int** id) {  
 UserEntity userEntity = **userRepository**.findOne(id);  
  
 **if** (userEntity == **null**) {  
 **throw new** NotFoundException(**"Пользователь не найден"**);  
 }  
 **else** {  
 **return** userEntity;  
 }  
 }  
  
 **public void** save(UserEntity userEntity) {  
 **userRepository**.save(userEntity);  
 }  
  
 **public void** delete(UserEntity userEntity) {  
 **userRepository**.delete(userEntity);  
 }  
  
}

# Приложение В

Листинг класса AdminController

**package** ru.ugrasu.journal.controllers;  
  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.http.MediaType;  
**import** org.springframework.web.bind.annotation.PathVariable;  
**import** org.springframework.web.bind.annotation.RequestMapping;  
**import** org.springframework.web.bind.annotation.RequestMethod;  
**import** org.springframework.web.bind.annotation.RestController;  
**import** ru.ugrasu.journal.dto.DepartmentDto;  
**import** ru.ugrasu.journal.dto.RoleDto;  
**import** ru.ugrasu.journal.dto.StudyGroupDto;  
**import** ru.ugrasu.journal.dto.UserDto;  
**import** ru.ugrasu.journal.exception.NotFoundException;  
**import** ru.ugrasu.journal.model.entities.DepartmentEntity;  
**import** ru.ugrasu.journal.model.entities.RoleEntity;  
**import** ru.ugrasu.journal.model.entities.StudyGroupEntity;  
**import** ru.ugrasu.journal.model.entities.UserEntity;  
**import** ru.ugrasu.journal.model.services.\*;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
**import static** org.springframework.http.MediaType.***APPLICATION\_JSON\_UTF8\_VALUE***;  
**import static** org.springframework.web.bind.annotation.RequestMethod.***GET***;  
  
@RestController  
@RequestMapping(value = **"admin"**)  
**public class** AdminController {  
  
 @Autowired  
 **private** UserService **userService**;  
  
 @Autowired  
 **private** RoleService **roleService**;  
  
 @Autowired  
 **private** DepartmentService **departmentService**;  
  
 @Autowired  
 **private** StudyGroupService **studyGroupService**;  
  
 @Autowired  
 **private** SubjectService **subjectService**;  
  
 @Autowired  
 **private** ExcerciseService **excerciseService**;  
  
 @Autowired  
 **private** UserExcerciseService **userExcerciseService**;  
  
 @RequestMapping(value = **"/refrashUser"**, produces = ***APPLICATION\_JSON\_UTF8\_VALUE***, method = ***GET***)  
 **public** List<UserDto> findAllUser() {  
 System.***out***.println(**"AdminController - findAllUser"**);  
  
 List<UserDto> listUserDto = **new** ArrayList<>();  
 List<UserEntity> listUserEntity = **userService**.findAll();  
  
 **if** (listUserEntity == **null**) {  
 **throw new** NotFoundException(**"Нет пользователей"**);  
 }  
 **else** {  
 listUserEntity.forEach(userEntity -> {  
 UserDto userDto = **new** UserDto();  
 userDto.setId(userEntity.getId());  
 userDto.setName(userEntity.getName());  
 userDto.setPassword(userEntity.getPassword());  
  
 *//Если роль преподаватель - добавляем кафедру в ДТО  
 //Если роль Староста или студент - добавляем группу в ДТО* RoleDto roleDto = **new** RoleDto();  
 roleDto.setId(userEntity.getRoleByRole().getId());  
 roleDto.setName(userEntity.getRoleByRole().getName());  
  
 userDto.setRoleByRole(roleDto);  
  
 String role = roleDto.getName();  
  
 **if** ((role.equals(**"Староста"**)) || (role.equals(**"Студент"**))) {  
 StudyGroupDto studyGroupDto = **new** StudyGroupDto();  
 studyGroupDto.setId(userEntity.getStudyGroupByStudyGroup().getId());  
 studyGroupDto.setName(userEntity.getStudyGroupByStudyGroup().getName());  
  
 userDto.setStudyGroupByStudyGroup(studyGroupDto);  
 }  
  
 **if** (role.equals(**"Учитель"**)) {  
 DepartmentDto departmentDto = **new** DepartmentDto();  
 departmentDto.setId(userEntity.getDepartmentByDepartment().getId());  
 departmentDto.setName(userEntity.getDepartmentByDepartment().getName());  
  
 userDto.setDepartmentByDepartment(departmentDto);  
 }  
  
 *//****TODO*** *//Предметы у преподавателей не указали!* listUserDto.add(userDto);  
 });  
  
 **return** listUserDto;  
 }  
 }  
  
 @RequestMapping(value = **"/refrashRole"**, produces = ***APPLICATION\_JSON\_UTF8\_VALUE***, method = ***GET***)  
 **public** List<RoleDto> findAllRole() {  
 System.***out***.println(**"AdminController - findAllRole"**);  
  
 List<RoleDto> listRoleDto = **new** ArrayList<>();  
 List<RoleEntity> listRoleEntity = **roleService**.findAll();  
  
 **if** (listRoleEntity == **null**) {  
 **throw new** NotFoundException(**"Нет ролей"**);  
 }  
 **else** {  
 listRoleEntity.forEach(roleEntity -> {  
 RoleDto roleDto = **new** RoleDto();  
 roleDto.setId(roleEntity.getId());  
 roleDto.setName(roleEntity.getName());  
  
 listRoleDto.add(roleDto);  
 });  
  
 **return** listRoleDto;  
 }  
 }  
  
 @RequestMapping(value = **"/refrashDepartment"**, produces = ***APPLICATION\_JSON\_UTF8\_VALUE***, method = ***GET***)  
 **public** List<DepartmentDto> findAllDepartment() {  
 System.***out***.println(**"AdminController - findAllDepartment"**);  
  
 List<DepartmentDto> listDepartmentDto = **new** ArrayList<>();  
 List<DepartmentEntity> listDepartmentEntity = **departmentService**.findAll();  
  
 **if** (listDepartmentEntity == **null**) {  
 **throw new** NotFoundException(**"Нет кафедр"**);  
 }  
 **else** {  
 listDepartmentEntity.forEach(departmentEntity -> {  
 DepartmentDto departmentDto = **new** DepartmentDto();  
 departmentDto.setId(departmentEntity.getId());  
 departmentDto.setName(departmentEntity.getName());  
  
 listDepartmentDto.add(departmentDto);  
 });  
  
 **return** listDepartmentDto;  
 }  
 }  
  
 @RequestMapping(value = **"/refrashStudyGroup"**, produces = ***APPLICATION\_JSON\_UTF8\_VALUE***, method = ***GET***)  
 **public** List<StudyGroupDto> findAllStudyGroup() {  
 System.***out***.println(**"AdminController - findAllStudyGroup"**);  
  
 List<StudyGroupDto> listStudyGroupDto = **new** ArrayList<>();  
 List<StudyGroupEntity> listStudyGroupEntity = **studyGroupService**.findAll();  
  
 **if** (listStudyGroupEntity == **null**) {  
 **throw new** NotFoundException(**"Нет групп"**);  
 }  
 **else** {  
 listStudyGroupEntity.forEach(studyGroupEntity -> {  
 StudyGroupDto studyGroupDto = **new** StudyGroupDto();  
 studyGroupDto.setId(studyGroupEntity.getId());  
 studyGroupDto.setName(studyGroupEntity.getName());  
  
 listStudyGroupDto.add(studyGroupDto);  
 });  
  
 **return** listStudyGroupDto;  
 }  
 }  
  
 @RequestMapping(value = **"/saveUser"**, consumes = MediaType.***APPLICATION\_FORM\_URLENCODED\_VALUE***, method = RequestMethod.***POST***)  
 **public void** saveUser(UserDto userDto) {  
  
 UserEntity userEntity = **new** UserEntity();  
 userEntity.setId(userDto.getId());  
 userEntity.setName(userDto.getName());  
 userEntity.setPassword(userDto.getPassword());  
  
 RoleEntity roleEntity = **roleService**.findById(userDto.getRoleId());  
 userEntity.setRoleByRole(roleEntity);  
  
 *//Если есть ссылка на группу (студент или староста)* **if** (userDto.getStudyGroupId() != 0) {  
 StudyGroupEntity studyGroupEntity = **studyGroupService**.findById(userDto.getStudyGroupId());  
 userEntity.setStudyGroupByStudyGroup(studyGroupEntity);  
 }  
  
 *//Учитель* **if** (userDto.getDepartmentId() != 0) {  
 DepartmentEntity departmentEntity = **departmentService**.findById(userDto.getDepartmentId());  
 userEntity.setDepartmentByDepartment(departmentEntity);  
 }  
  
 **userService**.save(userEntity);  
 }  
  
 @RequestMapping(value = **"/deleteUser/{id}"**, produces = ***APPLICATION\_JSON\_UTF8\_VALUE***, method = ***GET***)  
 **public void** deleteUser(@PathVariable(**"id"**) **int** id) {  
  
 UserEntity userEntity = **userService**.findById(id);  
  
 **if** (userEntity != **null**) {  
 *//Удаляем связи* userEntity.getSubjectsById().forEach(subjectEntity -> {  
  
 subjectEntity.getExcercisesById().forEach(excerciseEntity -> {  
  
 excerciseEntity.getUserExcercisesById().forEach(userExcerciseEntity -> {  
 **userExcerciseService**.delete(userExcerciseEntity);  
 });  
  
 **excerciseService**.delete(excerciseEntity);  
 });  
  
 **subjectService**.delete(subjectEntity);  
 });  
  
 userEntity.getUserExcercisesById().forEach(userExcerciseEntity -> {  
 **userExcerciseService**.delete(userExcerciseEntity);  
 });  
  
 **userService**.delete(userEntity);  
  
 }  
 }  
  
}

Листинг класса TeacherController

**package** ru.ugrasu.journal.controllers;  
  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.http.MediaType;  
**import** org.springframework.web.bind.annotation.PathVariable;  
**import** org.springframework.web.bind.annotation.RequestMapping;  
**import** org.springframework.web.bind.annotation.RequestMethod;  
**import** org.springframework.web.bind.annotation.RestController;  
**import** ru.ugrasu.journal.dto.\*;  
**import** ru.ugrasu.journal.exception.NotFoundException;  
**import** ru.ugrasu.journal.model.entities.\*;  
**import** ru.ugrasu.journal.model.services.\*;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
**import static** org.springframework.http.MediaType.***APPLICATION\_JSON\_UTF8\_VALUE***;  
**import static** org.springframework.web.bind.annotation.RequestMethod.***GET***;  
  
@RestController  
@RequestMapping(value = **"teacher"**)  
**public class** TeacherController {  
  
 @Autowired  
 **private** SubjectService **subjectService**;  
  
 @Autowired  
 **private** StudyGroupService **studyGroupService**;  
  
 @Autowired  
 **private** UserService **userService**;  
  
 @Autowired  
 **private** ExcerciseService **excerciseService**;  
  
 @Autowired  
 **private** UserExcerciseService **userExcerciseService**;  
  
 @RequestMapping(value = **"/refrashSubject"**, produces = ***APPLICATION\_JSON\_UTF8\_VALUE***, method = ***GET***)  
 **public** List<SubjectDto> findAllSubject() {  
 System.***out***.println(**"TeacherController - findAllSubject"**);  
  
 List<SubjectDto> listSubjectDto = **new** ArrayList<>();  
 List<SubjectEntity> listSubjectEntity = **subjectService**.findAll();  
  
 **if** (listSubjectEntity == **null**) {  
 **throw new** NotFoundException(**"Нет предметов"**);  
 }  
 **else** {  
 listSubjectEntity.forEach(subjectEntity -> {  
 SubjectDto subjectDto = **new** SubjectDto();  
 subjectDto.setId(subjectEntity.getId());  
 subjectDto.setName(subjectEntity.getName());  
  
 listSubjectDto.add(subjectDto);  
 });  
  
 **return** listSubjectDto;  
 }  
 }  
  
 @RequestMapping(value = **"/refrashStudyGroup"**, produces = ***APPLICATION\_JSON\_UTF8\_VALUE***, method = ***GET***)  
 **public** List<StudyGroupDto> findAllStudyGroup() {  
 System.***out***.println(**"TeacherController - findAllStudyGroup"**);  
  
 List<StudyGroupDto> listStudyGroupDto = **new** ArrayList<>();  
 List<StudyGroupEntity> listStudyGroupEntity = **studyGroupService**.findAll();  
  
 **if** (listStudyGroupEntity == **null**) {  
 **throw new** NotFoundException(**"Нет групп"**);  
 }  
 **else** {  
 listStudyGroupEntity.forEach(studyGroupEntity -> {  
 StudyGroupDto studyGroupDto = **new** StudyGroupDto();  
 studyGroupDto.setId(studyGroupEntity.getId());  
 studyGroupDto.setName(studyGroupEntity.getName());  
  
 listStudyGroupDto.add(studyGroupDto);  
 });  
  
 **return** listStudyGroupDto;  
 }  
 }  
  
 @RequestMapping(value = **"/getStudentsByGroup/{id}"**, produces = ***APPLICATION\_JSON\_UTF8\_VALUE***, method = ***GET***)  
 **public** List<UserDto> findUserByGroupId(@PathVariable(**"id"**) **int** id) {  
 System.***out***.println(**"TeacherController - findUserByGroupId - "** + id);  
  
 List<UserDto> listUserDto = **new** ArrayList<>();  
 List<UserEntity> listUserEntity = **userService**.findUserByGroupId(id);  
  
 **if** (listUserEntity == **null**) {  
 **throw new** NotFoundException(**"Нет студентов"**);  
 }  
 **else** {  
 listUserEntity.forEach(userEntity -> {  
 UserDto userDto = **new** UserDto();  
 userDto.setId(userEntity.getId());  
 userDto.setName(userEntity.getName());  
  
 StudyGroupDto studyGroupDto = **new** StudyGroupDto();  
 studyGroupDto.setId(userEntity.getStudyGroupByStudyGroup().getId());  
 studyGroupDto.setName(userEntity.getStudyGroupByStudyGroup().getName());  
  
 userDto.setStudyGroupByStudyGroup(studyGroupDto);  
  
 *//Обработка промежуточной таблицы* List<UserExcerciseEntity> listUserExcerciseEntity = userEntity.getUserExcercisesById();  
 List<ExcerciseDto> listExcerciseDto = **new** ArrayList<>();  
  
 **if** (listUserExcerciseEntity != **null**) {  
  
 listUserExcerciseEntity.forEach(userExcerciseEntity -> {  
  
 ExcerciseDto excerciseDto = **new** ExcerciseDto();  
  
 excerciseDto.setId(userExcerciseEntity.getExcerciseByExcercise().getId());  
 excerciseDto.setDate(userExcerciseEntity.getExcerciseByExcercise().getDate());  
  
 listExcerciseDto.add(excerciseDto);  
 });  
 }  
  
 userDto.setExcercisesById(listExcerciseDto);  
  
 listUserDto.add(userDto);  
 });  
  
 **return** listUserDto;  
 }  
 }  
  
 @RequestMapping(value = **"/getExcercises/{idSubject}/{idGroup}"**,  
 produces = ***APPLICATION\_JSON\_UTF8\_VALUE***, method = ***GET***)  
 **public** List<ExcerciseDto> findExcercises(@PathVariable(**"idSubject"**) **int** idSubject,  
 @PathVariable(**"idGroup"**) **int** idGroup)  
 {  
 System.***out***.println(**"TeacherController - findExcercises - "** + idSubject + **" "** + idGroup);  
  
 List<ExcerciseEntity> listExcerciseEntity = **excerciseService**.findAll();  
 List<ExcerciseDto> listExcerciseDto = **new** ArrayList<>();  
  
 listExcerciseEntity.forEach(excerciseEntity -> {  
 ExcerciseDto excerciseDto = **new** ExcerciseDto();  
 excerciseDto.setId(excerciseEntity.getId());  
 excerciseDto.setDate(excerciseEntity.getDate());  
  
 *//Проверка соответствия группы и предмета* **int** subject = excerciseEntity.getSubjectBySubjectId().getId();  
 **int** group = excerciseEntity.getGroupId();  
  
 **if** ((subject == idSubject) && (group == idGroup)) {  
 listExcerciseDto.add(excerciseDto);  
 }  
 });  
  
 **return** listExcerciseDto;  
 }  
  
 @RequestMapping(value = **"/saveExcercise"**, consumes = MediaType.***APPLICATION\_FORM\_URLENCODED\_VALUE***, method = RequestMethod.***POST***)  
 **public void** saveExcercise(ExcerciseDto excerciseDto) {  
 System.***out***.println(**"TeacherController - saveExcercise"**);  
  
 ExcerciseEntity excerciseEntity = **new** ExcerciseEntity();  
  
 SubjectEntity subjectEntity = **subjectService**.findById(excerciseDto.getSubjectId());  
  
 excerciseEntity.setId(excerciseDto.getId());  
 excerciseEntity.setDate(excerciseDto.getDate());  
 excerciseEntity.setGroupId(excerciseDto.getGroupId());  
 excerciseEntity.setSubjectBySubjectId(subjectEntity);  
  
 **excerciseService**.save(excerciseEntity);  
 }  
  
 @RequestMapping(value = **"/saveUserExcercise"**, consumes = MediaType.***APPLICATION\_FORM\_URLENCODED\_VALUE***, method = RequestMethod.***POST***)  
 **public void** saveUserExcercise(UserExcerciseDto userExcerciseDto) {  
 System.***out***.println(**"TeacherController - saveUserExcercise"**);  
  
 **int** userId = userExcerciseDto.getUserId();  
 **int** excerciseId = userExcerciseDto.getExcerciseId();  
  
 UserExcerciseEntity userExcerciseEntity = **userExcerciseService**.findByUE(userId,excerciseId);  
  
 **if** (userExcerciseEntity == **null**) {  
 *//Создаем нового* userExcerciseEntity = **new** UserExcerciseEntity();  
 userExcerciseEntity.setId(0);  
  
 UserEntity userEntity = **userService**.findById(userId);  
 ExcerciseEntity excerciseEntity = **excerciseService**.findById(excerciseId);  
  
 userExcerciseEntity.setUserByUser(userEntity);  
 userExcerciseEntity.setExcerciseByExcercise(excerciseEntity);  
  
 **userExcerciseService**.save(userExcerciseEntity);  
 } **else** {  
 *//Удаляем запись из таблицы* **userExcerciseService**.delete(userExcerciseEntity);  
 }  
 }  
}

# Приложение Г

Листинг класса UserServiceTest

**package** ru.ugrasu.journal.UnitTest;  
  
**import** org.junit.Test;  
**import** org.mockito.InjectMocks;  
**import** org.mockito.Mock;  
**import** ru.ugrasu.journal.JournalApplicationTests;  
**import** ru.ugrasu.journal.exception.NotFoundException;  
**import** ru.ugrasu.journal.model.entities.RoleEntity;  
**import** ru.ugrasu.journal.model.entities.StudyGroupEntity;  
**import** ru.ugrasu.journal.model.entities.UserEntity;  
**import** ru.ugrasu.journal.model.repositories.UserRepository;  
**import** ru.ugrasu.journal.model.services.UserService;  
  
**import static** org.junit.Assert.*assertEquals*;  
**import static** org.mockito.Mockito.*when*;  
  
**public class** UserServiceTest **extends** JournalApplicationTests {  
  
 @Mock  
 **private** UserRepository **userRepository**;  
  
 @InjectMocks  
 **private** UserService **userService**;  
  
 @Test  
 **public void** findByIdMustReturnUserEntity() **throws** Exception {  
 UserEntity u = **new** UserEntity();  
  
 RoleEntity r = **new** RoleEntity();  
 r.setId(1);  
 r.setName(**"admin"**);  
  
 StudyGroupEntity g = **new** StudyGroupEntity();  
 g.setId(1);  
 g.setName(**"1541б"**);  
  
 u.setId(1);  
 u.setName(**"testUser"**);  
 u.setStudyGroupByStudyGroup(g);  
 u.setRoleByRole(r);  
  
 *when*(**userRepository**.findOne(u.getId())).thenReturn(u);  
  
 UserEntity userEntity = **userService**.findById(1);  
  
 *assertEquals*(u.getId(), userEntity.getId());  
 }  
  
 @Test(expected = NotFoundException.**class**)  
 **public void** findByIdMustReturnNotFoundException() **throws** NotFoundException {  
 *when*(**userRepository**.findOne(Integer.***MIN\_VALUE***)).thenReturn(**null**);  
 **userService**.findById(Integer.***MIN\_VALUE***);  
 }  
}