## 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 4.1 - Сущность группы

package com.hescha.teacher\_workload\_accounting.entity;

import lombok.Data;

import javax.persistence.\*;

import java.util.ArrayList;

import java.util.List;

@Data

@Entity

@Table(name = "my\_group")

public class Group extends AbstractEntity {

private String name;

@ManyToOne

@JoinColumn(name = "department\_id")

private Department department;

@ManyToOne

@JoinColumn(name = "faculty\_id")

private Faculty faculty;

@ManyToOne

@JoinColumn(name = "training\_form\_id")

private TrainingForm trainingForm;

@ManyToOne

@JoinColumn(name = "speciality\_id")

private Speciality speciality;

private String course;

private String studentCount;

private String groupCount;

@OneToMany(fetch = FetchType.LAZY, mappedBy = "group", cascade = CascadeType.REMOVE)

private List<TableRow> tableRows = new ArrayList<>();

public Group() {

}

public Group(String name, TrainingForm trainingForm) {

this.name = name;

this.trainingForm = trainingForm;

}

public Group(String name, Department department, Faculty faculty,

TrainingForm trainingForm, Speciality speciality,

String course, String studentCount, String groupCount) {

this.name = name;

this.department = department;

this.faculty = faculty;

this.trainingForm = trainingForm;

this.speciality = speciality;

this.course = course;

this.studentCount = studentCount;

this.groupCount = groupCount; }

@Override

public String toString() {

return

name +

", " + department +

", " + faculty +

", " + trainingForm +

", " + speciality +

", курс: " + course +

", студентов: " + studentCount +

", групп: " + groupCount

;

}

}

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.

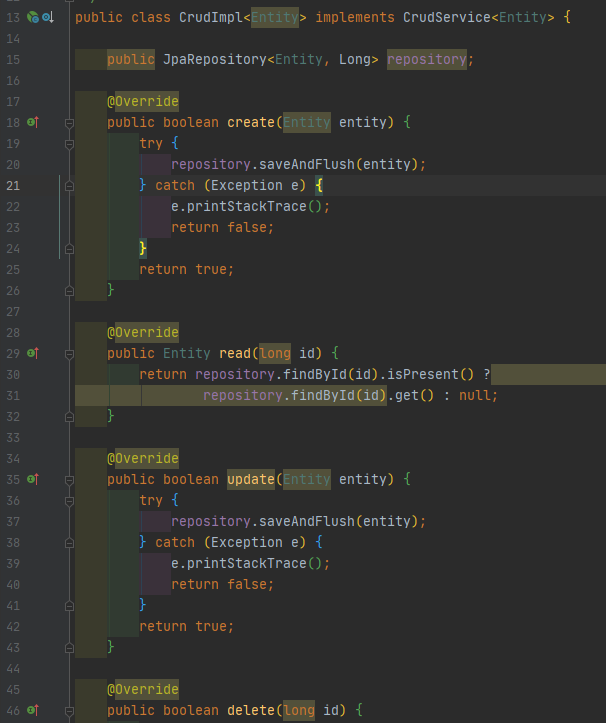


Рисунок 4.2 – Реализация интерфейса

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.

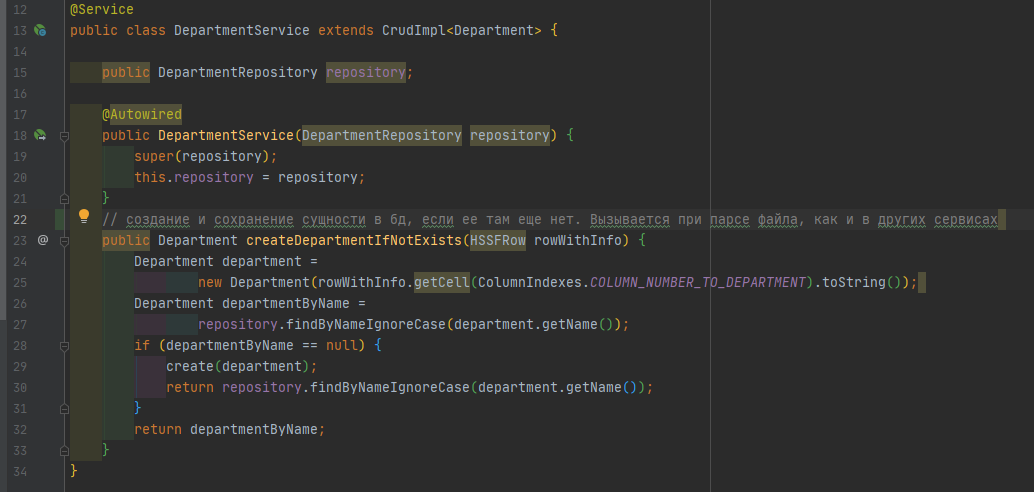


Рисунок 4.3 – Сервис доступа к таблице

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.

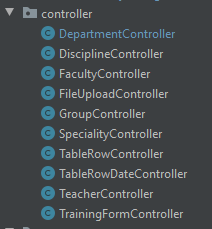


Рисунок 4.4 – Контроллеры страниц

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

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 4.2 - Контроллер факультета

package com.hescha.teacher\_workload\_accounting.controller;

import com.hescha.teacher\_workload\_accounting.entity.Department;

import com.hescha.teacher\_workload\_accounting.entity.Faculty;

import com.hescha.teacher\_workload\_accounting.entity.Group;

import com.hescha.teacher\_workload\_accounting.entity.TableRow;

import com.hescha.teacher\_workload\_accounting.service.FacultyService;

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 java.util.ArrayList;

import java.util.List;

@Controller

@RequestMapping("/facultys")

public class FacultyController {

@Autowired

private FacultyService service;

@GetMapping

public String getList(Model model) {

model.addAttribute("list", service.repository.findAll());

return "facultys";

}

@RequestMapping(path = {"/edit", "/edit/{id}"})

public String edit(Model model, @PathVariable(name = "id", required =

false) Long id) {

if (id != null) {

Faculty entity = service.read(id);

model.addAttribute("entity", entity);

} else {

model.addAttribute("entity", new Faculty());

}

return "facultys-add-edit";

}

@RequestMapping(path = "/delete/{id}")

public String delete(Model model, @PathVariable("id") Long id) {

service.delete(id);

return "redirect:/facultys";

}

@RequestMapping(path = "/create", method = RequestMethod.POST)

public String createOrUpdate(Faculty entity) {

service.create(entity);

return "redirect:/facultys";

}

@RequestMapping(path = "/showWorkload/{id}")

public String showWorkload(Model model, @PathVariable("id") Long id) {

Faculty entity = service.read(id);

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

for(Group g:entity.getGroups()){

list.addAll(g.getTableRows());

}

model.addAttribute("list", list);

model.addAttribute("message","По выбранному факультету: "+entity.getName());

return "tableRows";

}

}

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.

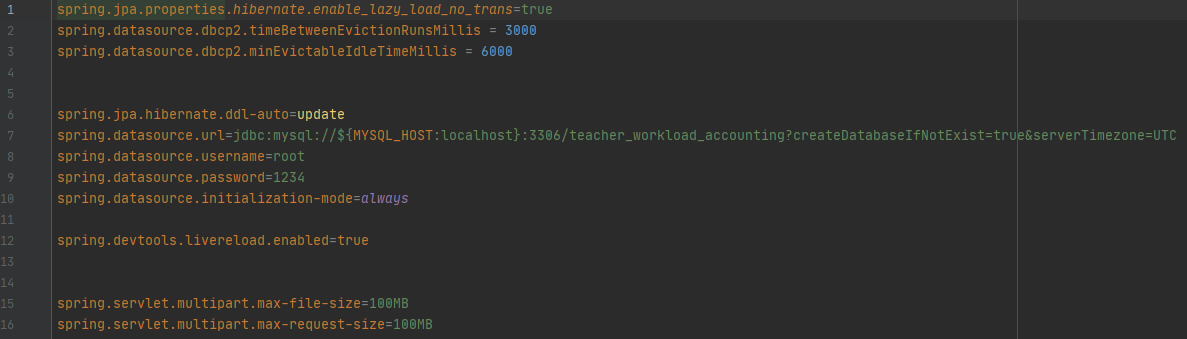


Рисунок 4.5 – Файл настройщик приложений

Проделав все эти операции, мы имеем следующую диаграмму классов



Рисунок 4.6 – Диаграмма классов

## 4.2 Разработка интерфейса программного продукта

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 тарифов будут использоваться строки листинга 4.3.

Листинг 4.3 - Шаблон для отображения списка преподавателей

<section class="list">

<div class="container my-2">

<div class="card">

<div class="row">

<div class="card-body col">

<h2 class="text-center"> Кафедры</h2>

<div th:switch="${list}" class="container my-5">

<div class="col-md-12">

<h2 th:case="null">Записей не найдено</h2>

<div th:case="\*" class="overScroll">

<table class="table table-striped table-responsive-md">

<thead>

<tr>

<th>ФИО</th>

</tr>

</thead>

<tbody>

<tr th:each="entity : ${list}">

<td th:text="${entity.name}"></td>

<td class="widthLastCol ">

<a th:href="@{/teachers/showWorkload/{id}(id=${entity.id})}" class="myButton ">

Нагрузка выбранного

</a>

</td>

<td class="widthLastCol ">

<a th:href="@{/teachers/edit/{id}(id=${entity.id})}" class="myButton ">

<i class="fas fa-edit">Редактировать</i>

</a>

</td>

<td class="widthLastCol ">

<a th:href="@{/teachers/delete/{id}(id=${entity.id})}" class="myButton">

<i class="fas fa-trash">Удалить</i>

</a>

</td>

</tr>

</tbody>

</table>

</div>

<p class="d-flex justify-content-end">

<a href="/teachers/edit" class="myButton mx-2">

<i class="fas fa-plus"> Добавить запись </i></a>

</p>

</div>

</div>

</div>

</div>

</div>

</div>

</section>

Для отображения списка значений в выпадающем списке используется следующий код.

Листинг 4.4 - Отображение значений как выпадающий список

<select class="form-control" name="faculty\_id" id="" required>

<option th:if="${entity.faculty==null}" th:each="temp : ${faculties}"

th:value="${temp.id}" th:text="${temp.name}"

th:selected="${entity.faculty==temp}">

</option>

<option th:if="${entity.faculty!=null}" th:each="temp : ${faculties}"

th:value="${temp.id}" th:text="${temp.name}"

th:selected="${entity.faculty.id==temp.id}">

</option>

</select>

После создания всех шаблонов 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 заноситься информация о нагрузке.

Она довольно большая (поскольку содержит множество колонок разных видов работ), поэтому имеет горизонтальную полосу прокрутки.

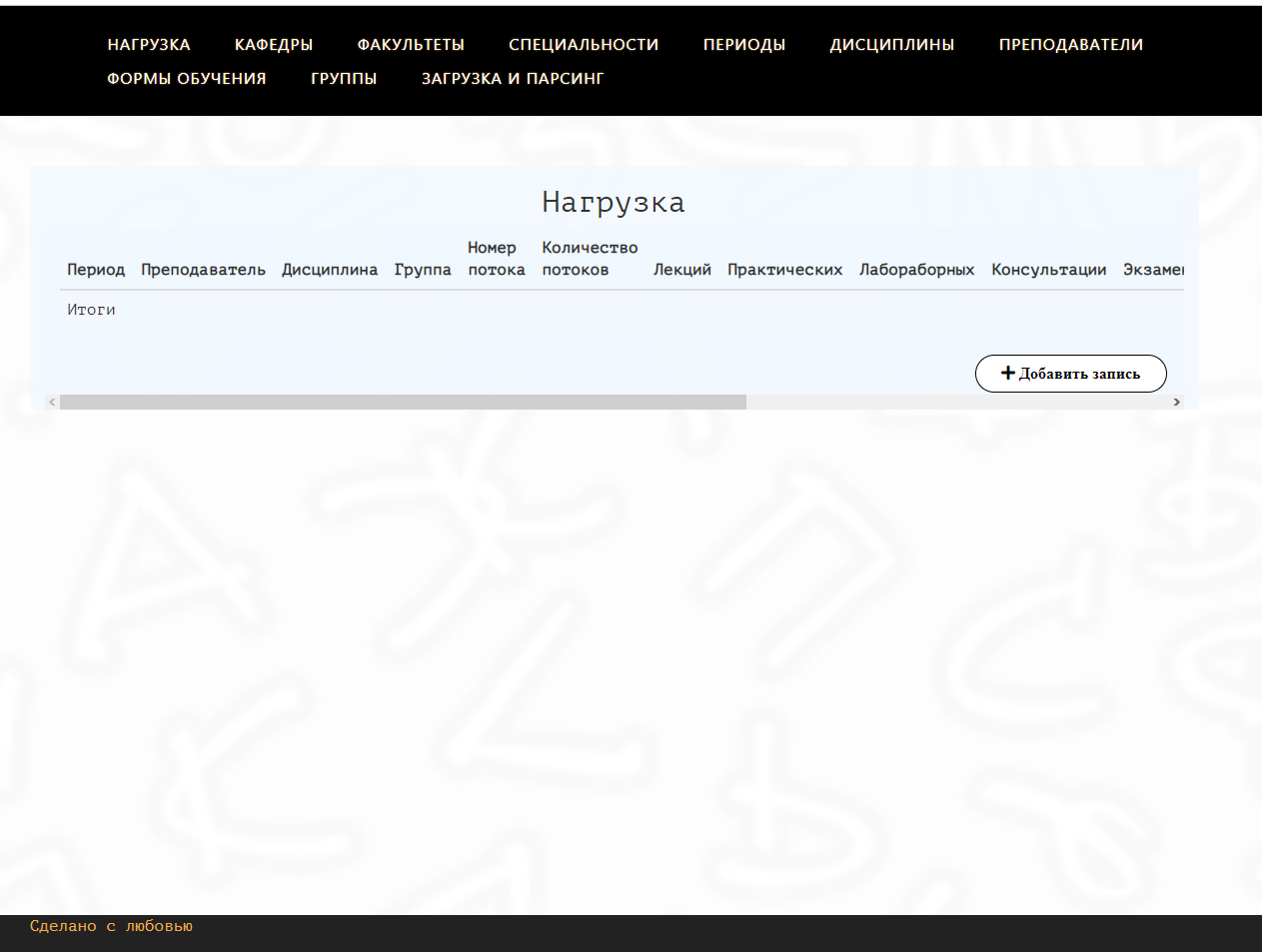


Рисунок 4.7 – Страница нагрузки

Для добавления записи вручную, достаточно нажать кнопку добавления и будет осуществлен переход на страницу создания записи нагрузки. В ней из выпадающих списков выбираются преподаватель, период, группа, дисциплина и записываются данные о нагрузке по каждому типу.

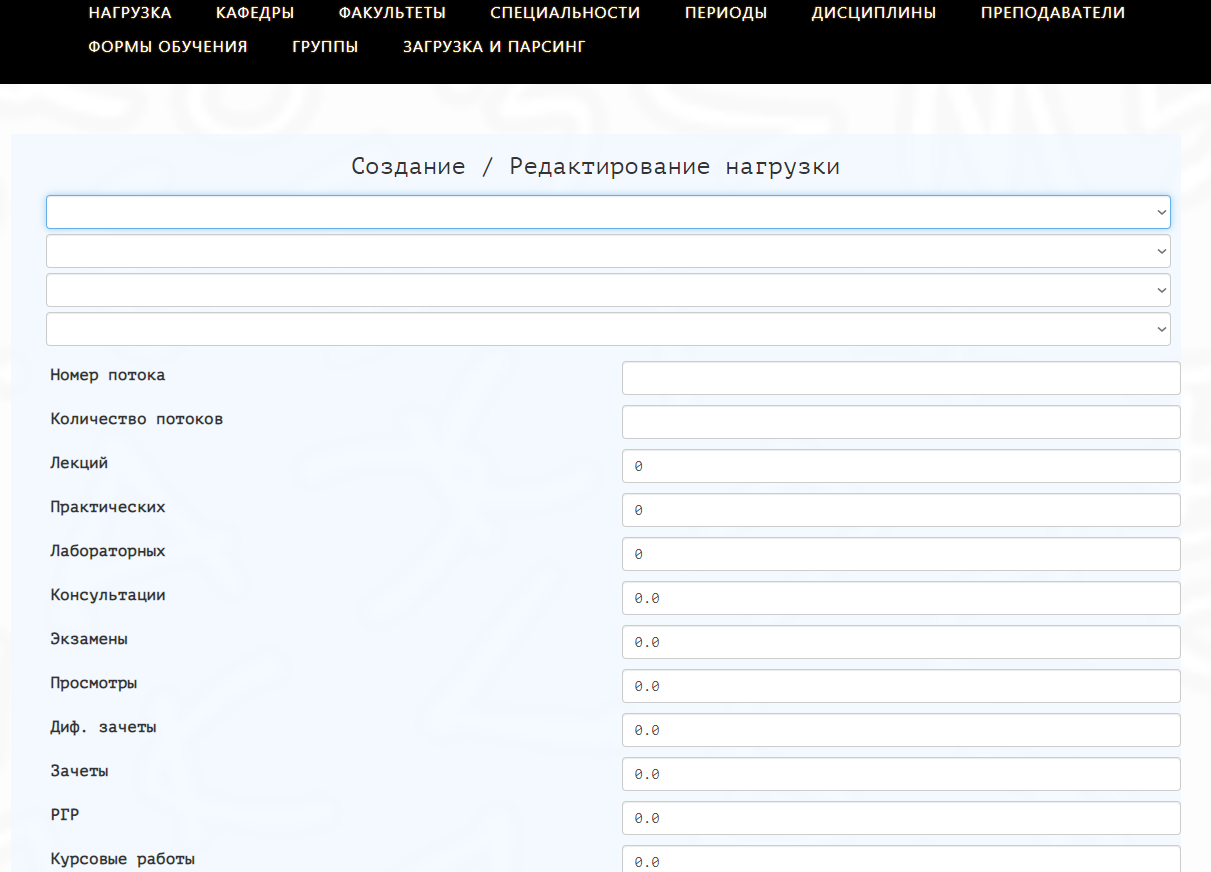


Рисунок 4.8 – Страница создания записи нагрузки

Поскольку база данных сейчас пуста, то и пусты выпадающие списки. Разумеется, сперва требуется заполнить данные, используемые в выпадающих списках. Заполнение данных для всех таблиц работает по одному и тому же принципу. Первым делом нужно выбрать в какую таблицу бд будут заноситься данные и перейти на страницу ее управления

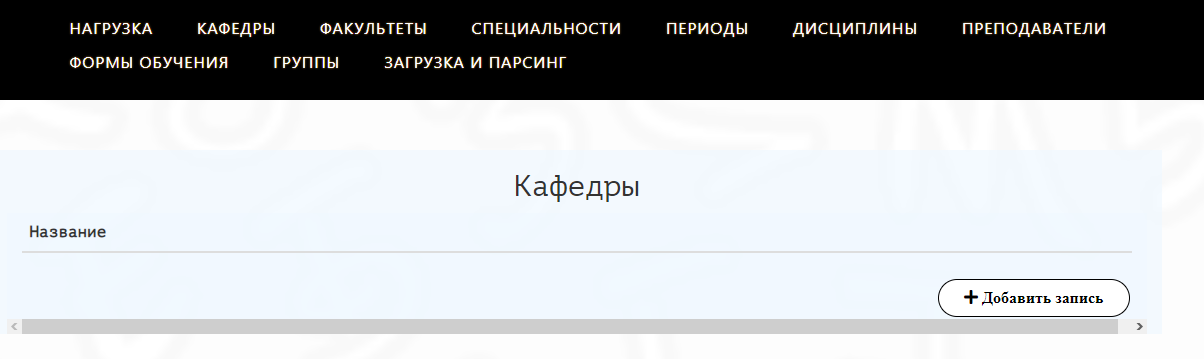


Рисунок 4.9 – Страница работы с кафедрами

После перехода следует нажать кнопку добавления записи и заполнить все поля

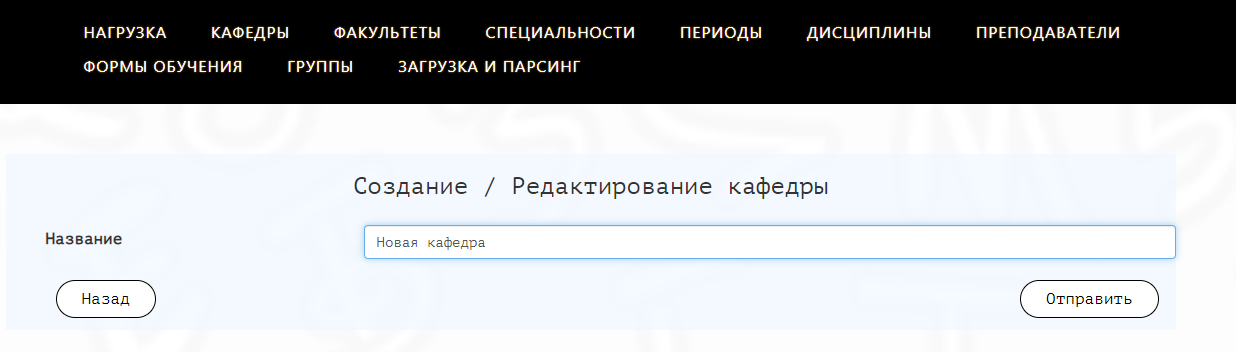


Рисунок 4.10 – Добавление новой кафедры

После заполнение нажимается кнопка отправить, и новая запись отображается в таблице.

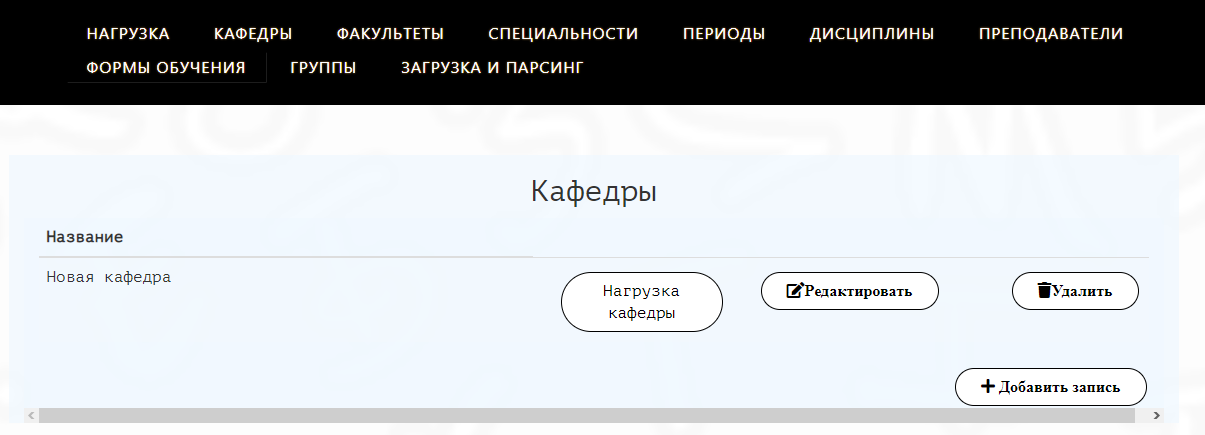


Рисунок 4.11 – Результат добавления записи

Но поскольку таких записей может быть много, и учитывая привычку работать в excel, была сделана отдельно страница для загрузки и парсинга данных.

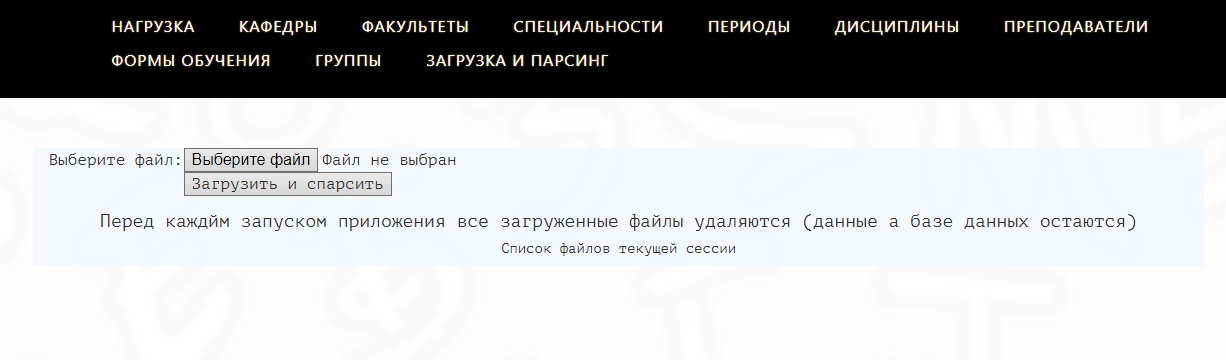


Рисунок 4.12 – Страница загрузки и парсинга данных

В ней выбирается файл со структурой нагрузки, отображенной ниже

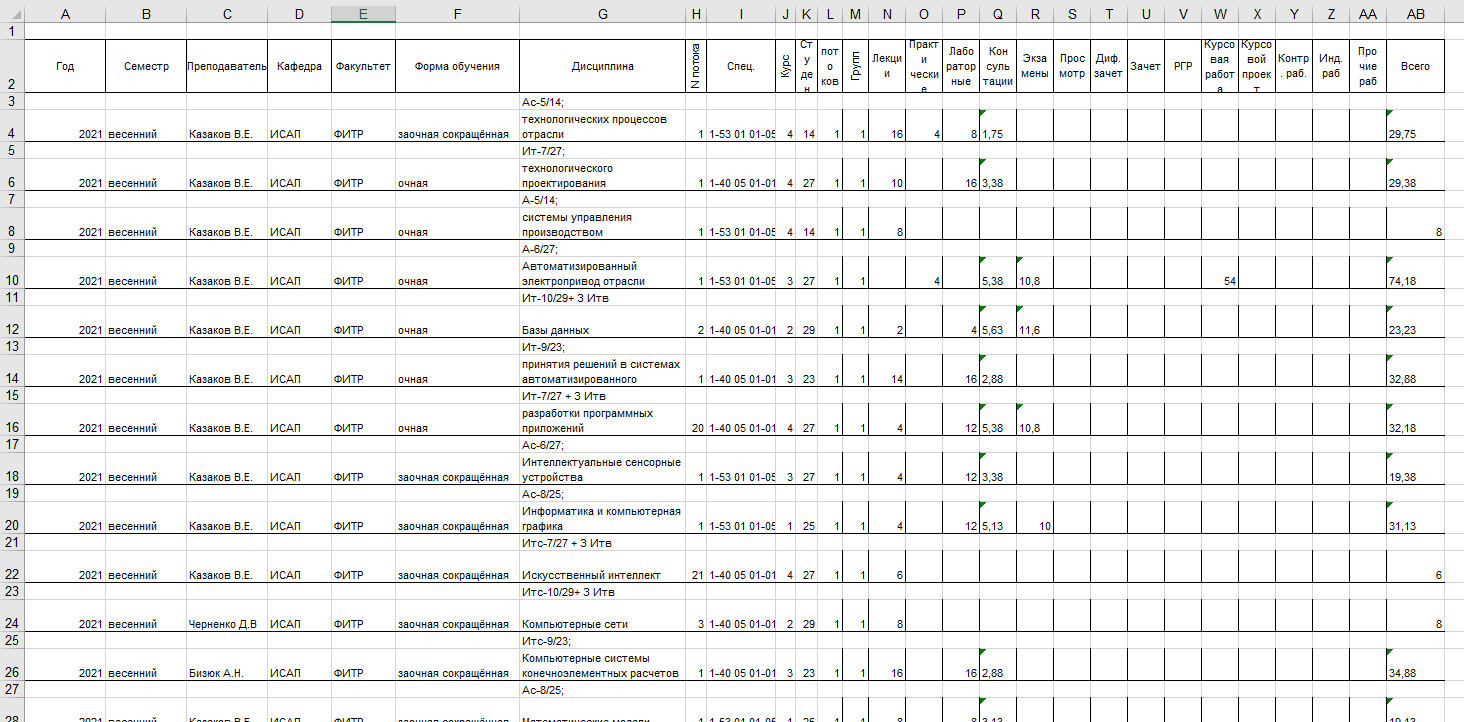


Рисунок 4.13 – Структура файла нагрузки для парсинга

Важно обратить внимание, что загружать можно файлы версии excel до 2003. Так что если у пользователя более новая версия, то ему остается только сохранить файл под старой версией, и можно будет его загружать на сайт

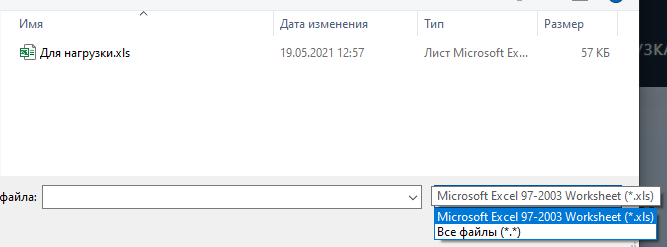


Рисунок 4.14 – Допустимая загрузка файла

После загрузки, сайт сам сразу же спарсит файл и сохранит в соответствующие таблицы все данные, которых еще не было в базе данных.

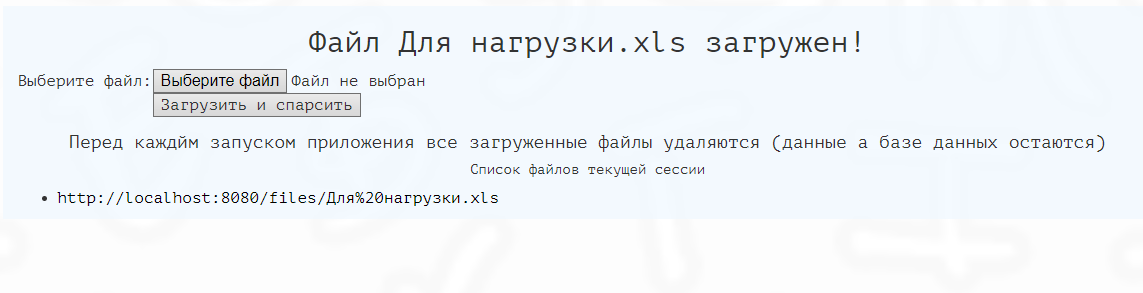


Рисунок 4.15 – Уведомление о загрузке данных

Теперь, при переходе на страницу нагрузки, можно видеть все загруженные данные.

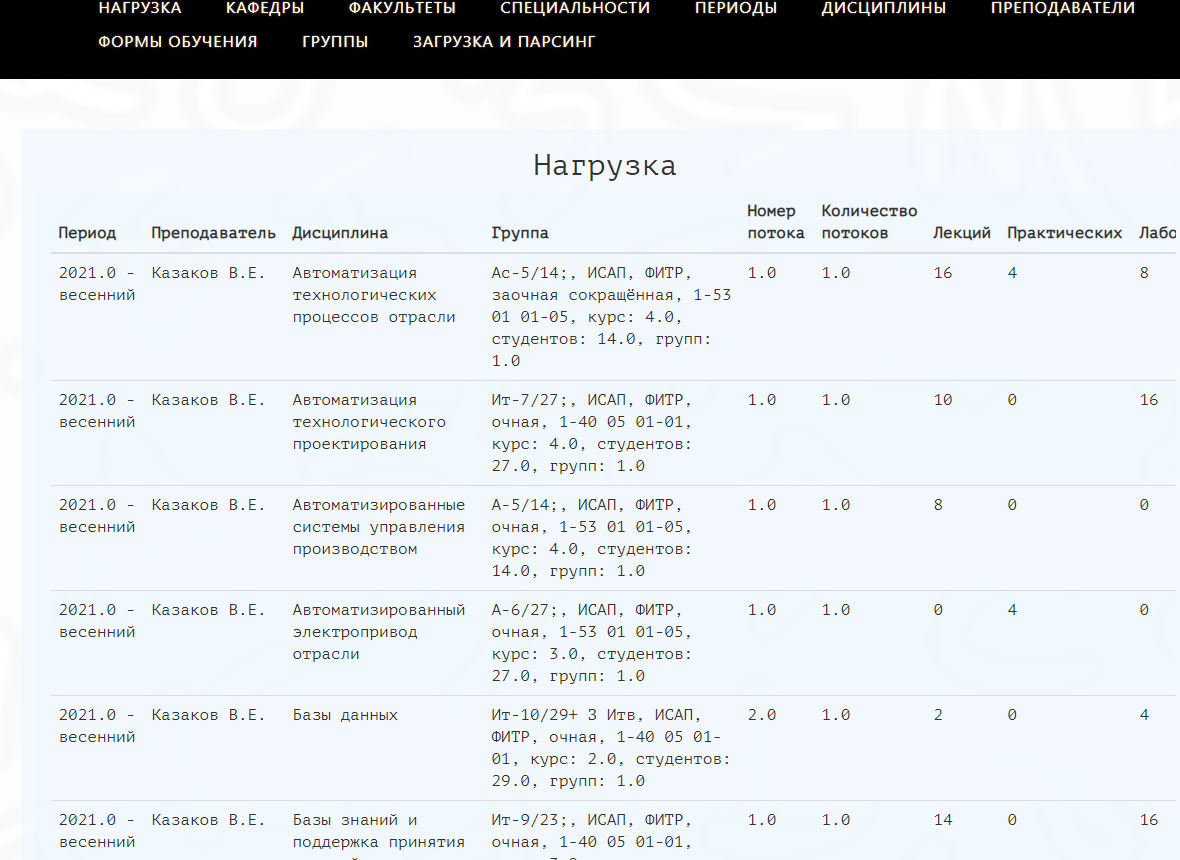


Рисунок 4.16 – Страница нагрузки после парсинга файла

В конце таблицы доступны итоговые значения по столбцам, что показано на рисунке 4.17.

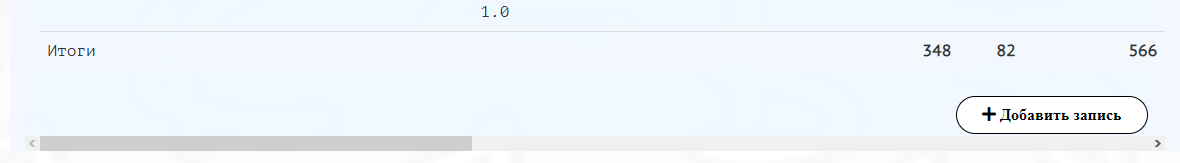


Рисунок 4.17 – Итоговые значения

Загруженные данные можно менять в любой из таблиц. Кроме того, для удобство сделана фильтрация нагрузки по таблицам. Т.е., к примеру, если мы хотим отфильтровать записи по какому-то преподавателю, то мы переходим на страницу преподавателей и жмем на кнопку нагрузки выбранного

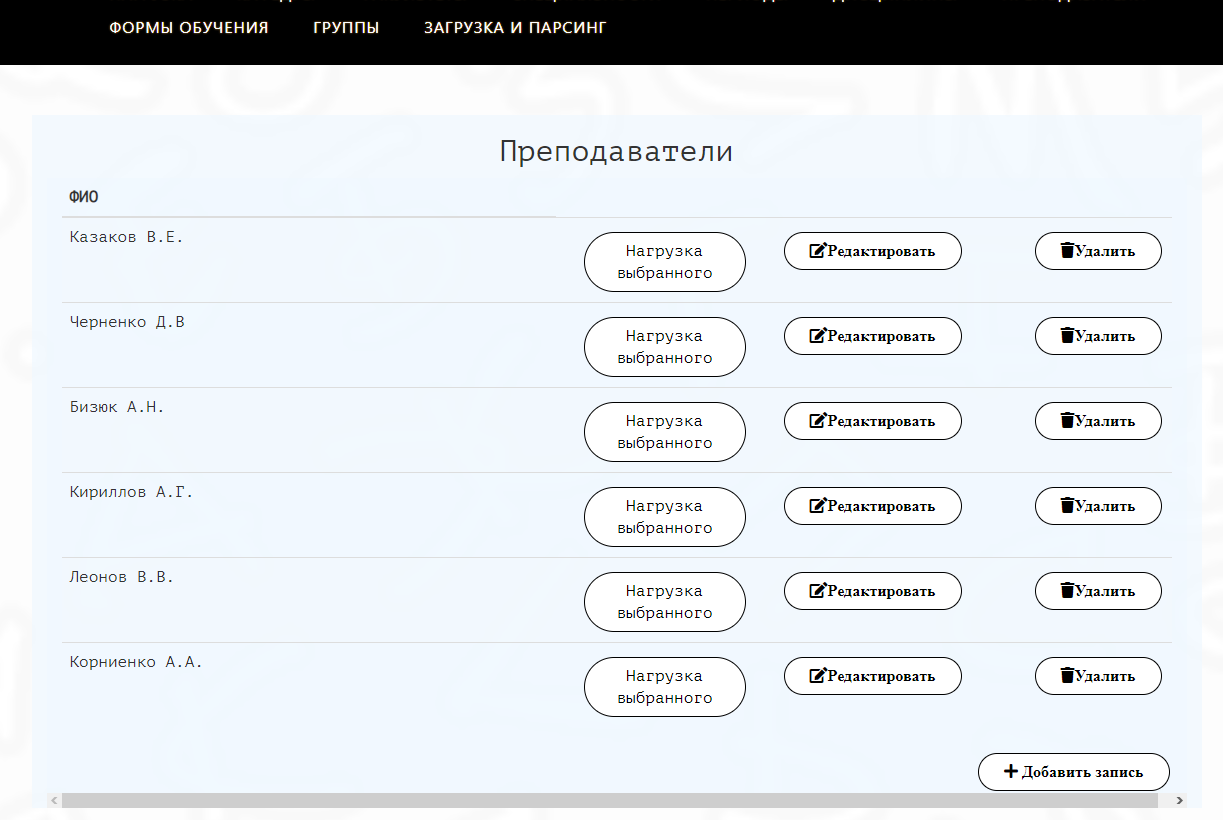


Рисунок 4.18 – Страница преподавателей после парсинга файла

После чего нас перебросит на страницу нагрузки с нагрузкой только выбранного преподавателя. Чтобы не забыть кого мы выбирали, под заголовком страницы пишется фио преподавателя. Для удобства также сделана кнопка возвращения к предыдущей странице

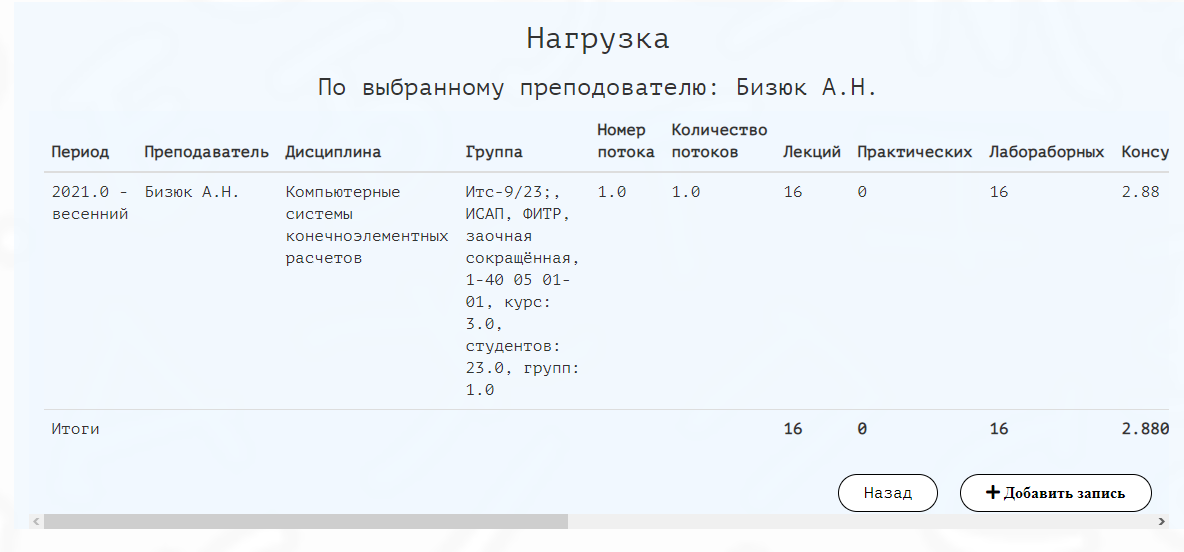


Рисунок 4.19 – Фильтрация по преподавателю

Как уже было сказано выше, фильтровать можно по большинству таблиц, в том числе и по группе

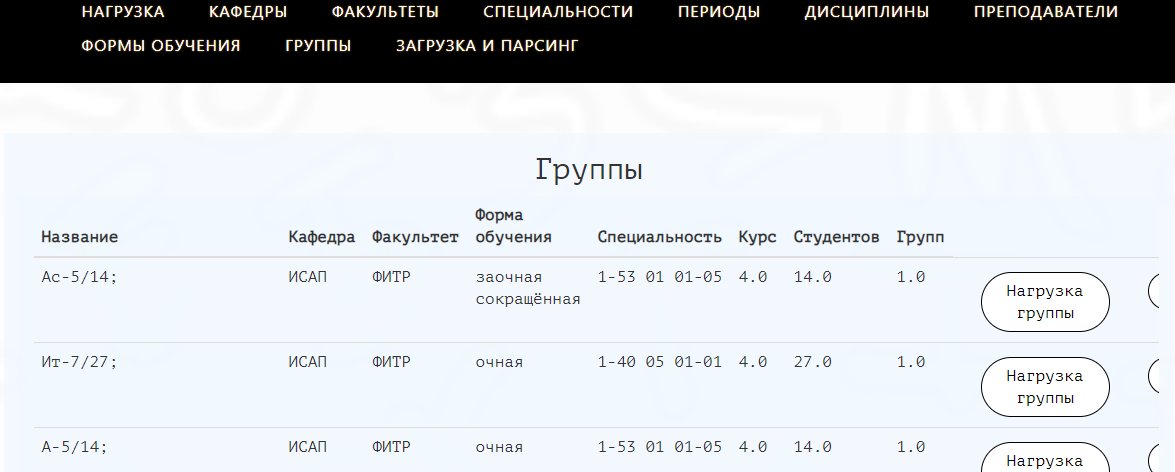


Рисунок 4.20 – Страница групп после парсинга

При выборе фильтрации по группе, будут отображены только те записи, которые включают данную группу

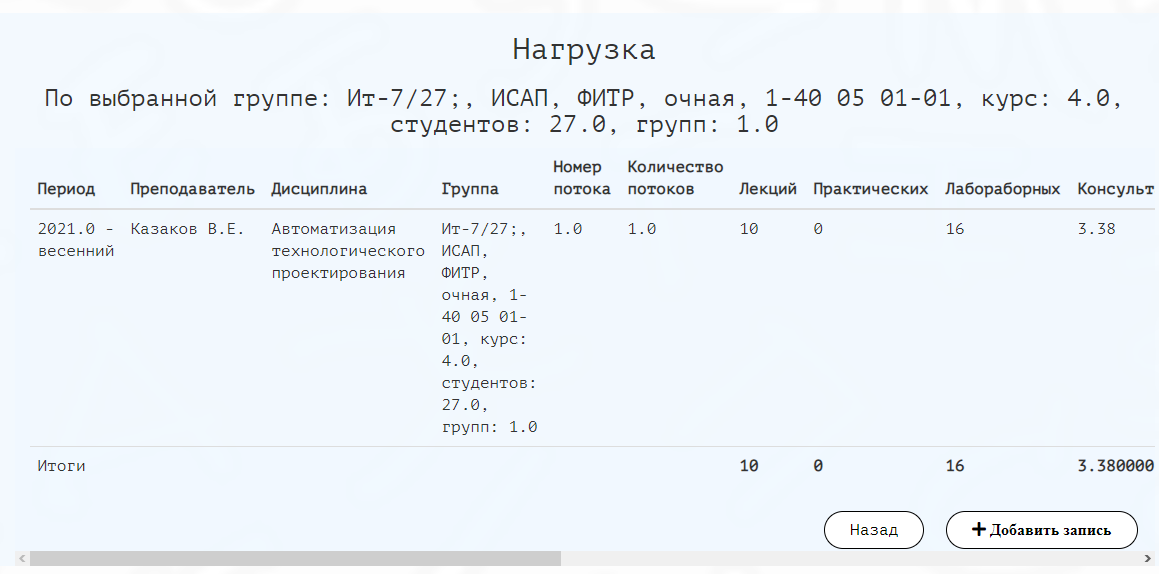


Рисунок 4.21 – Фильтрация по группе

## 4.3 Разработка алгоритмов реализации вариантов использования

При решении общей задачи возникали более мелкие, для решения которых необходимо было разрабатывать алгоритмы. Несколько примеров можно увидеть ниже:

1. Алгоритм сохранения в базе данных сущности, если ее еще там нет:

public Speciality createSpecialityIfNotExists(HSSFRow rowWithInfo) {

Speciality speciality =

new Speciality(rowWithInfo.getCell(ColumnIndexes.COLUMN\_NUMBER\_TO\_SPECIALITY).toString());

Speciality specialityByName =

repository.findByNameIgnoreCase(speciality.getName());

if (specialityByName == null) {

create(speciality);

return repository.findByNameIgnoreCase(speciality.getName());

}

return specialityByName;

}

1. Алгоритм парсинга загруженного файла построчно:

public void parseFileByPath(String filePath) {

System.out.println("start parsing file: "+filePath);

HSSFWorkbook wb = ExcelReader.readWorkbook(filePath);

if (wb != null) {

System.out.println("file opened ");

HSSFSheet sheet = wb.getSheetAt(0);

Iterator rowIter = sheet.rowIterator();

while (rowIter.hasNext()) {

HSSFRow row = (HSSFRow) rowIter.next();

if (isHeaderRow(row)) {

continue;

}

if (isRowEnded(row)) {

break;

}

saveParseDataToDatabase(row, (HSSFRow) rowIter.next());

}

} else{

System.out.println("workbook is null");

}

System.out.println("end parsing.");

}

private boolean isRowEnded(HSSFRow row) {

return row.getCell(ColumnIndexes.COLUMN\_NUMBER\_TO\_SPECIALITY) != null

&& (row.getCell(ColumnIndexes.COLUMN\_NUMBER\_TO\_TOTAL).toString().equals("Итого")

|| row.getCell(ColumnIndexes.COLUMN\_NUMBER\_TO\_GROUP\_NAME).toString().equals(""));

}

private boolean isHeaderRow(HSSFRow row) {

return row.getCell(ColumnIndexes.COLUMN\_NUMBER\_TO\_YEAR) != null

&& row.getCell(ColumnIndexes.COLUMN\_NUMBER\_TO\_YEAR).toString().equals(

"Год");

}

private void saveParseDataToDatabase(HSSFRow rowWithGroupName,

HSSFRow rowWithInfo) {

TableRowDate tableRowDate =

tableRowDateService.createTableRowDateIfNotExists(rowWithInfo);

Teacher teacher = teacherService.createTeacherIfNotExists(rowWithInfo);

Department department =

departmentService.createDepartmentIfNotExists(rowWithInfo);

Faculty faculty = facultyService.createFacultyIfNotExists(rowWithInfo);

TrainingForm trainingForm =

trainingFormService.createTrainingFormIfNotExists(rowWithInfo);

Speciality speciality =

specialityService.createSpecialityIfNotExists(rowWithInfo);

Group group = groupService.createGroupIfNotExists(rowWithGroupName,

rowWithInfo, department, trainingForm, faculty, speciality);

Discipline discipline =

disciplineService.createDisciplineIfNotExists(rowWithInfo);

tableRowService.createTableRowIfNotExists(rowWithInfo, tableRowDate,

teacher, group, discipline);

}

## 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 [13].

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 группы (таблица 4.2).

Таблица 4.2 - Тестирование добавления группы

|  |  |  |  |
| --- | --- | --- | --- |
| Краткое описание | Предварительные условия | Шаги для воспроизведения | Ожидаемый результат |
| Тестирование добавления группы | Запустить программу  Добавить информацию о специальностях, кафедре, факультете и форме обучения | Перейти на страницу добавления группы. Выбрать данные из выпадающего списка и заполнить поля. Отправить данные. | После сохранения, новая группа выводится на странице вместе с другими существующими |

1. Тестирование изменения группы (таблица 4.3).

Таблица 4.3 - Тестирование изменения группы

|  |  |  |  |
| --- | --- | --- | --- |
| Краткое описание | Предварительные условия | Шаги для воспроизведения | Ожидаемый результат |
| Тестирование изменение группы | Запустить программу. Добавить группу по предыдущему тесту | Перейти на страницу групп. Выбрать из списка группу для редактирования. Изменить данные и отправить. | В таблице групп отображаются новые данные |