МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное  
образовательное учреждение высшего образования  
«Самарский национальный исследовательский университет  
имени академика С.П. Королева»  
(Самарский университет)   
  
  
Институт информатики и кибернетики  
Кафедра программных систем  
  
Дисциплина  
**Практикум на ЭВМ  
  
  
  
ОТЧЕТ**по лабораторному практикуму  
  
Вариант №25

Студенты: Гребенщиков Д. А., Колбанов Д. О.  
Группа: 6301-020302D  
  
Преподаватель: Повова-Коварцева Д. А.  
  
Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Самара 2023

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#__RefHeading___Toc1862_1120461475)

[ХОД РАБОТЫ 5](#__RefHeading___Toc1864_1120461475)

[Часть 1. Проектирование базы данных 5](#__RefHeading___Toc1866_1120461475)

[Часть 2. Конструирование запросов 8](#__RefHeading___Toc1868_1120461475)

[Часть 3. Разработка интерфейса пользователя для работы с данными в БД. Создание форм. 11](#__RefHeading___Toc1870_1120461475)

[Тема 4. Конструирование отчета 15](#__RefHeading___Toc1872_1120461475)

[ПРИЛОЖЕНИЕ А 17](#__RefHeading___Toc1874_1120461475)

[ПРИЛОЖЕНИЕ B 22](#__RefHeading___Toc1876_1120461475)

[ПРИЛОЖЕНИЕ C 24](#__RefHeading___Toc1878_1120461475)

[ПРИЛОЖЕНИЕ D 35](#__RefHeading___Toc1880_1120461475)

# ВВЕДЕНИЕ

Проект «Дипломный проект»

Студенты высших учебных заведений на последнем курсе сдают госэкзамены (количество варьируется в зависимости от вуза), пишут и защищают дипломную работу. При написании дипломной работы выбирают тему дипломной работы и руководителя.

Необходимо спроектировать базу данных ДИПЛОМНЫЙ ПРОЕКТ, информация которой будет использоваться для хранения и поиска данных о научных темах, которые предложены студентам-дипломникам, успеваемости студентов и др.

В БД должна храниться информация:

* о СТУДЕНТАХ: номер зачетной книжки, Ф.И.О. студента, факультет, группа;
* ТЕМАХ: код преподавателя, тема дипломной работы;
* ОТМЕТКАХ: номер зачетной книжки; оценка, полученная на госэкзамене, оценка, полученная на защите дипломной работы;
* ПРЕПОДАВАТЕЛЯХ: код преподавателя, Ф.И.О. преподавателя, степень, звание, кафедра, телефон, e-mail.

При проектировании БД необходимо учитывать следующее:

* преподаватель для руководства студентами-дипломниками предлагает несколько тем дипломных работ. Тема дипломной работы может быть предложена только одним преподавателем;
* студент выбирает одну тему дипломной работы. Тема может быть выбрана только одним студентом;
* студент получает одну отметку. Отметка соответствует одному студенту.

Кроме того, следует учесть:

* преподаватель не обязательно предлагает тему дипломной работы (он может не иметь научной степени или научного звания, необходимых для руководителя дипломной работы). Каждая тема обязательно предлагается преподавателем для написания дипломной работы;
* каждый студент обязательно выбирает тему для написания дипломной работы. Тема не обязательно выбирается студентом;
* каждый студент обязательно получает отметку. Каждая отметка обязательно соответствует студенту.

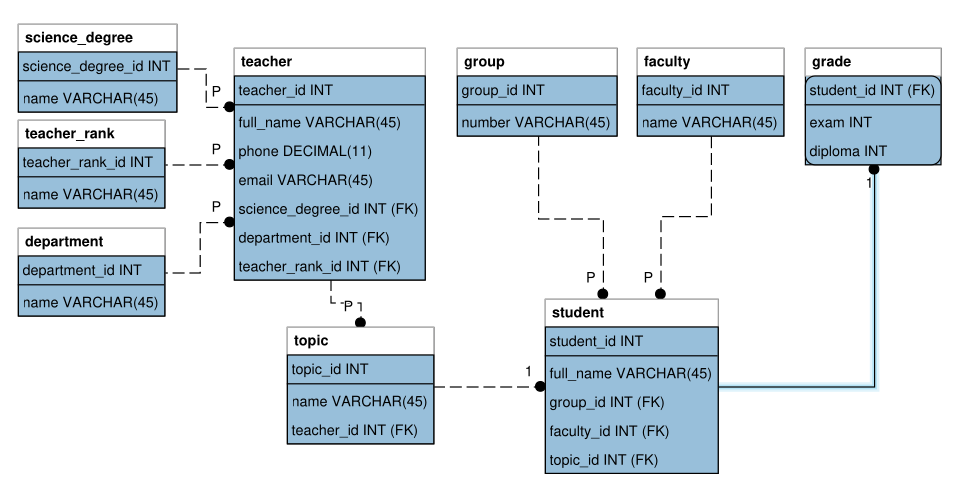
# ХОД РАБОТЫ

## Часть 1. Проектирование базы данных

1. Разработать ER-модель предметной области, описанной в проекте в соответствии с вариантом задания. Каждую сущность охарактеризовать набором атрибутов, при необходимости добавить атрибуты в сущности. Установить связи между сущностями;
2. Создать в выбранной СУБД разработанную БД. Наполнить таблицы записями (не менее 15 записей для каждой таблицы).

Ход выполнения работы:

1. Физическая модель

Рисунок 1.1 – Физическая модель

Домены атрибутов сущностей:

1. Сущность Преподаватель:

* teacher\_id – int(pk);
* full\_name – varchar(45);
* phone – decimal(11);
* email – varchar(45);
* science\_degree\_id – int(fk);
* department\_id – int(fk);
* teacher\_rank\_id – int(fk).

1. Сущность science\_degree:

* science\_degree\_id – int(pk);
* name – varchar(45).

1. Сущность teacher\_rank:

* teacher\_rank\_id – int(fk);
* name – varchar(45).

1. Сущность department:

* department\_id – int(pk);
* name – varchar(45).

1. Сущность topic:

* topic\_id – int(pk);
* name – varchar(45);
* teacher\_id – int(fk).

1. Сущность group:

* group\_id – int(pk);
* number – varchar(45).

1. Сущность faculty:

* faculty\_id – int(pk);
* name – varchar(45).

1. Сущность grade:

* student\_id – int(pk)(fk);
* exam – int;
* diploma – int.

1. Сущность student:

* student\_id – int(pk);
* full\_name – varchar(45);
* group\_id – int(fk);
* faculty\_id – int(fk);
* topic\_id – int(fk).

Скрипт создания и наполнения БД в СУБД приведен в приложении А.

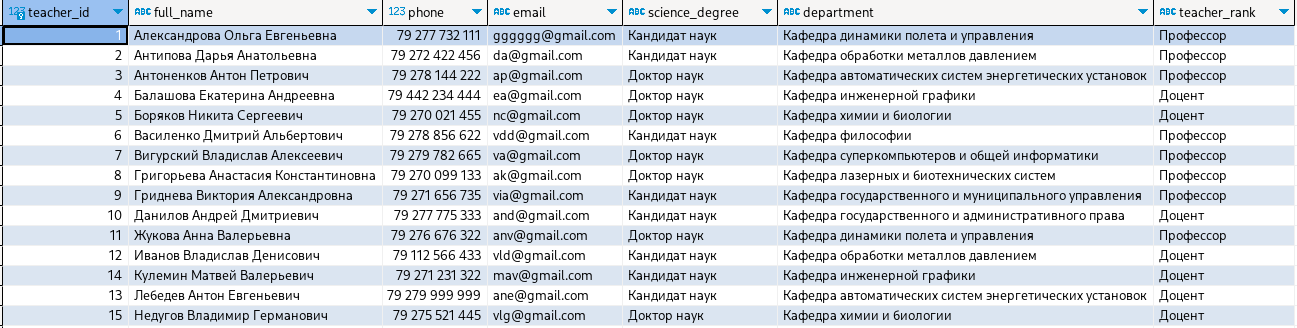
## Часть 2. Конструирование запросов

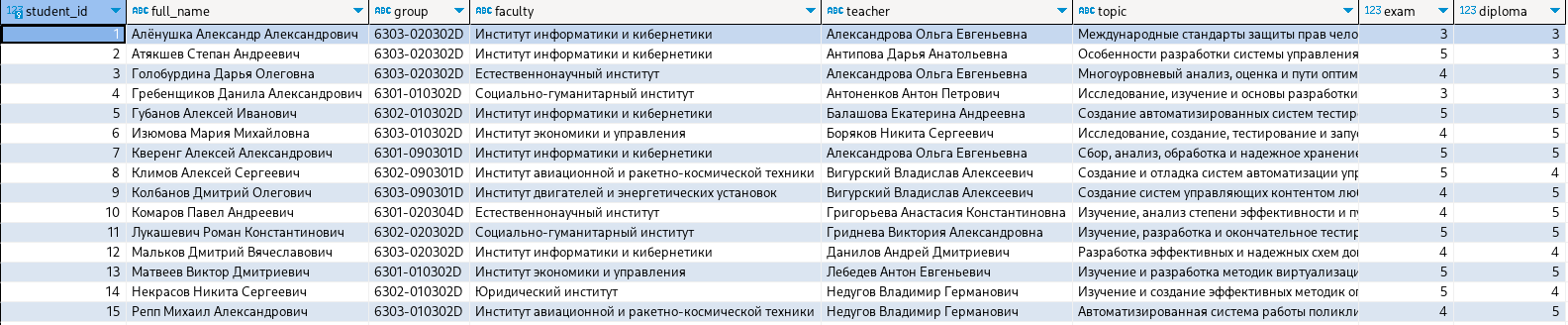
1. Сформулировать и сконструировать в выбранной СУБД запросы к БД, которые будут необходимы для предметной области (в соответствии с вариантом задания):

* Запрос на выборку избранных полей таблицы, с использованием синонима (алиаса) и сортировкой записей (ORDER BY);
* Запрос с использованием сортировки (ORDER BY) и группировки (GROUP BY);
* Запрос с использованием предложения DISTINCT;
* Запрос с использованием операций сравнения;
* Запросы для предикатов: IN, BETWEEN, LIKE, IS NULL;
* Запросы с использованием агрегатных функций (COUNT, SUM, AVG, MAX, MIN), производящие обобщенную групповую обработку значений полей (используя ключевые фразы GROUP BY и HAVING);
* Запрос на выборку данных из двух связанных таблиц. Выбрать несколько полей, по которым сортируется вывод;
* Многотабличный запрос с использованием внутреннего и внешнего соединения;
* Многотабличный запрос с использованием оператора UNION.

1. Создайте SQL команды для модификации данных (INSERT, UPDATE, DELETE).

Результаты запросов:

Рисунок 2.1 – Результат запроса с использованием необходимых операторов

Рисунок 2.2 – Результат запроса с использованием необходимых операторов

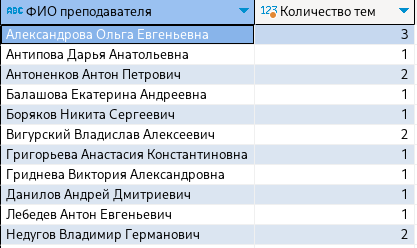


Рисунок 2.3 – Результат запроса с использованием необходимых операторов



Рисунок 2.4 – Результат запроса с использованием необходимых операторов



Рисунок 2.5 – Результат запроса с использованием необходимых операторов



Рисунок 2.6 – Результат запроса с использованием необходимых операторов

Код запросов приведен в приложение Б.

## Часть 3. Разработка интерфейса пользователя для работы с данными в БД. Создание форм.

Изучите теоретические основы разработки форм, рассмотренные в методических указаниях или других источниках.

1. Создайте необходимые формы для ввода информации в базу данных (созданной на предыдущих лабораторных работах), согласно предметной области своего варианта;
2. Проверьте работу форм (введите, измените и удалите около 10 записей в каждой форме);
3. Проверьте правильность работы обеспечения целостности данных;
4. Обдумайте и создайте формы, которые, возможно, будут полезными для будущих пользователей вашей БД.

Ход выполнения работы:

Для разработки интерфейса пользователя для работы с базой данных использовались: html, css, javascript, react.js для frontend и express.js для backend. На backend части были реализованы все http запросы, необходимые для взаимодействия с базой данных MySql. На frontend части с помощью библиотеки axios поступают get запросы на сервер, после чего происходит отображение данных на странице. Для добавления, обновления или удаления данных реализуются post, put и delete запросы соответственно.

Демонстрация разработанного сайта:

1. На рисунке 3.1 представлен вывод всех преподавателей.

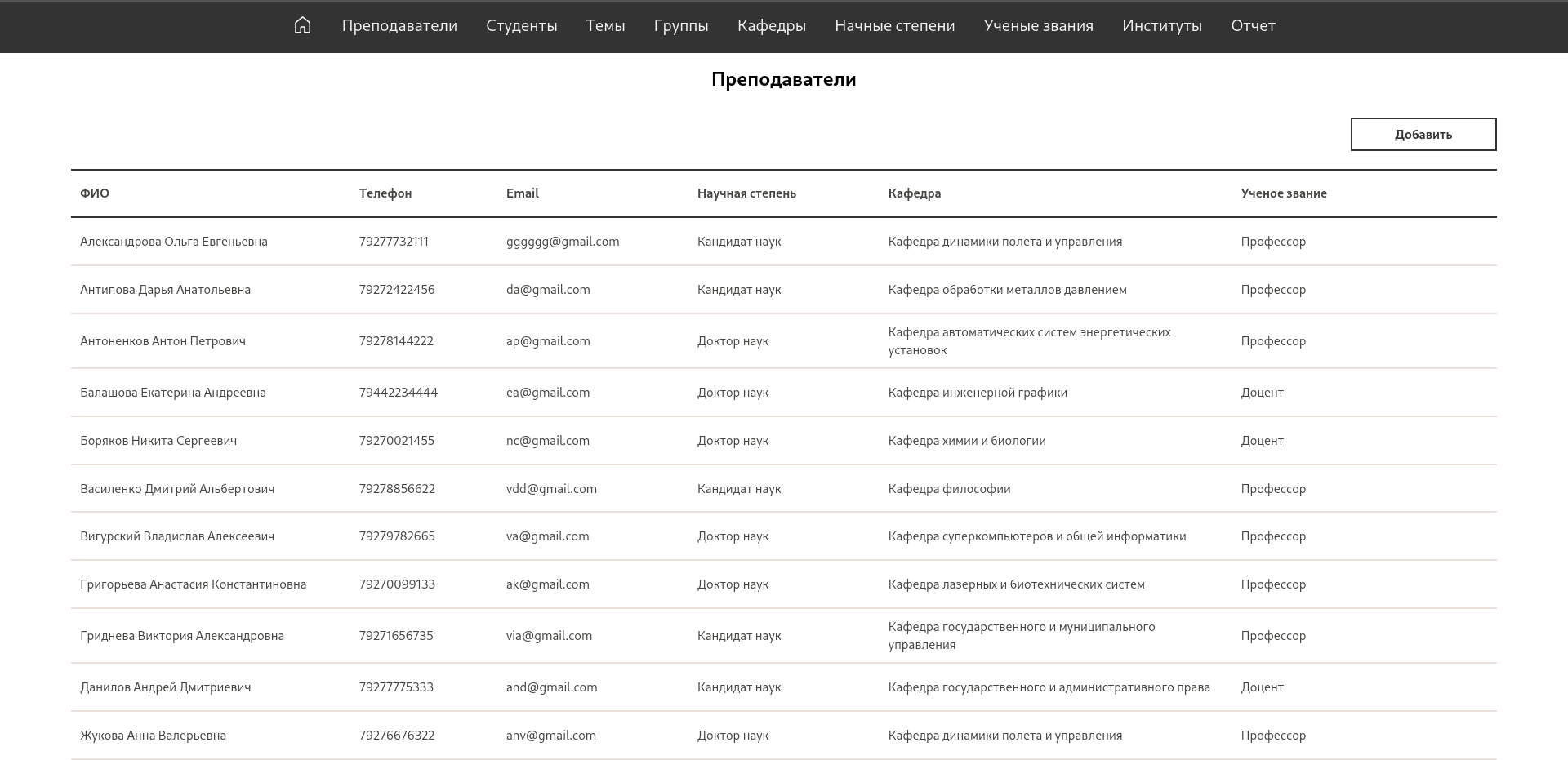


Рисунок 3.1 - Вывод всех преподавателей

1. На рисунке 3.2 представлен вывод всех студентов.

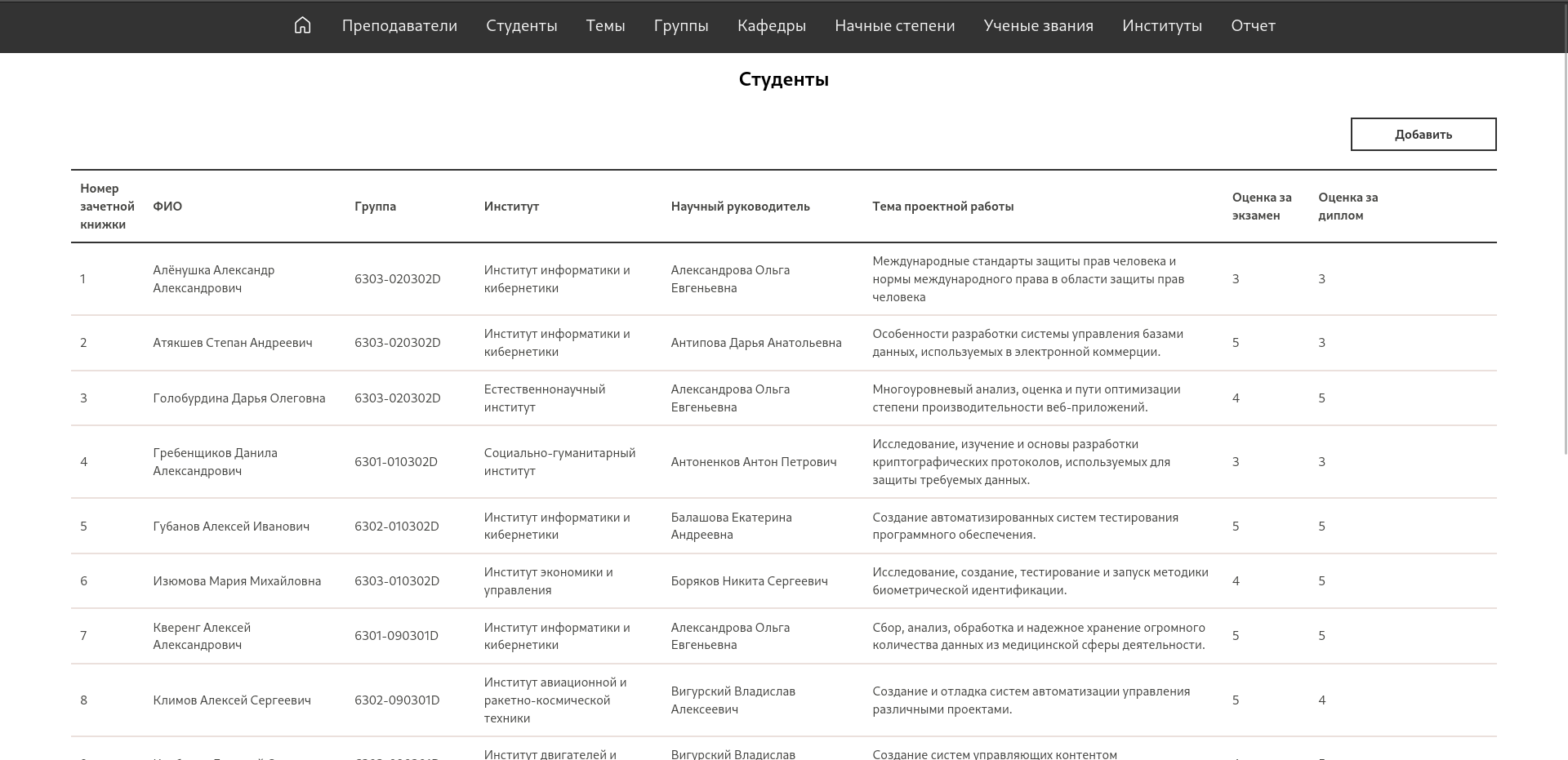


Рисунок 3.2 - Вывод всех студентов

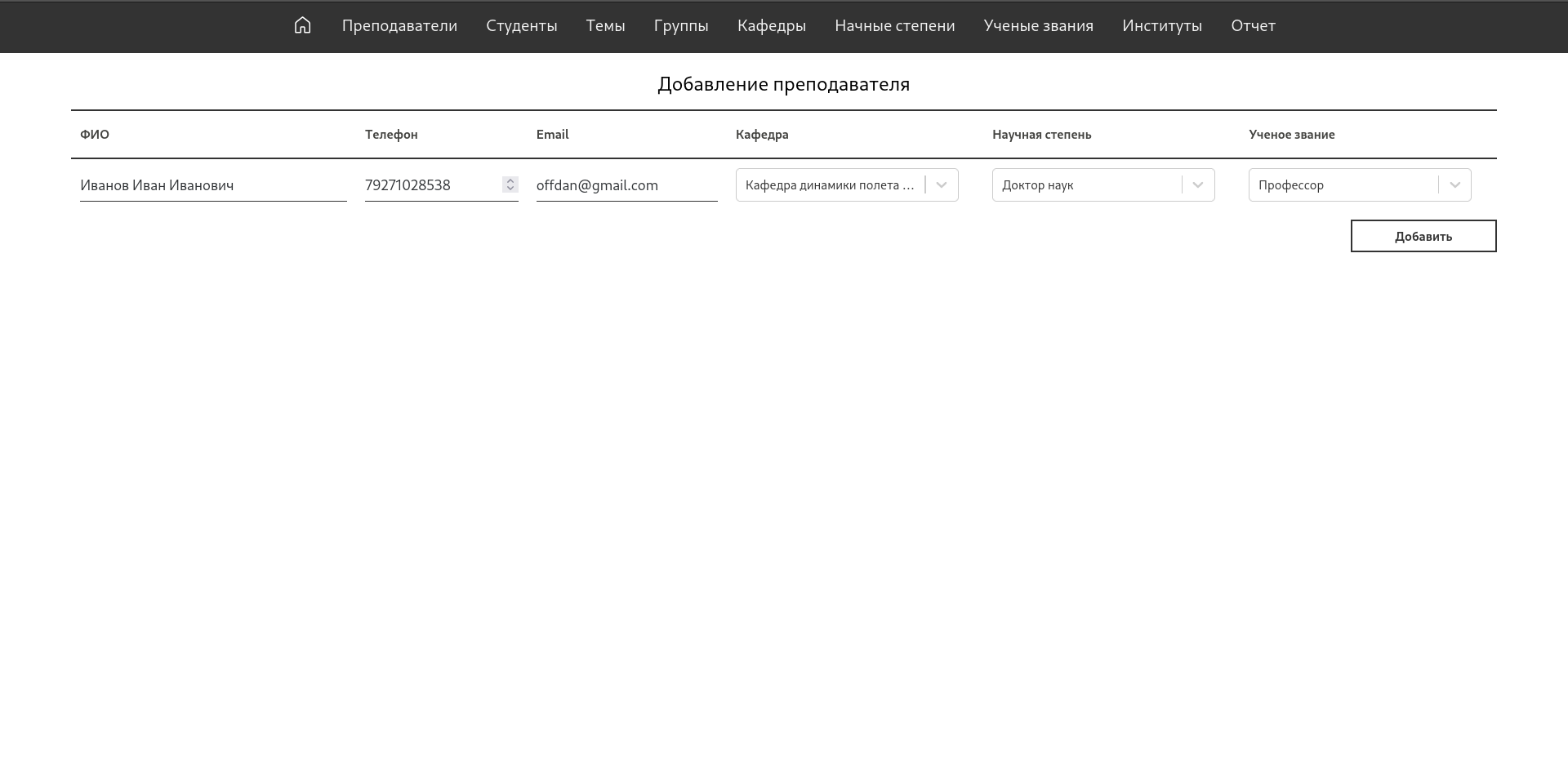
1. На рисунке 3.3 представлена форма добавления преподавателя.

Рисунок 3.3 - Форма добавления преподавателя

1. На рисунке 3.4 представлена форма добавления студента.

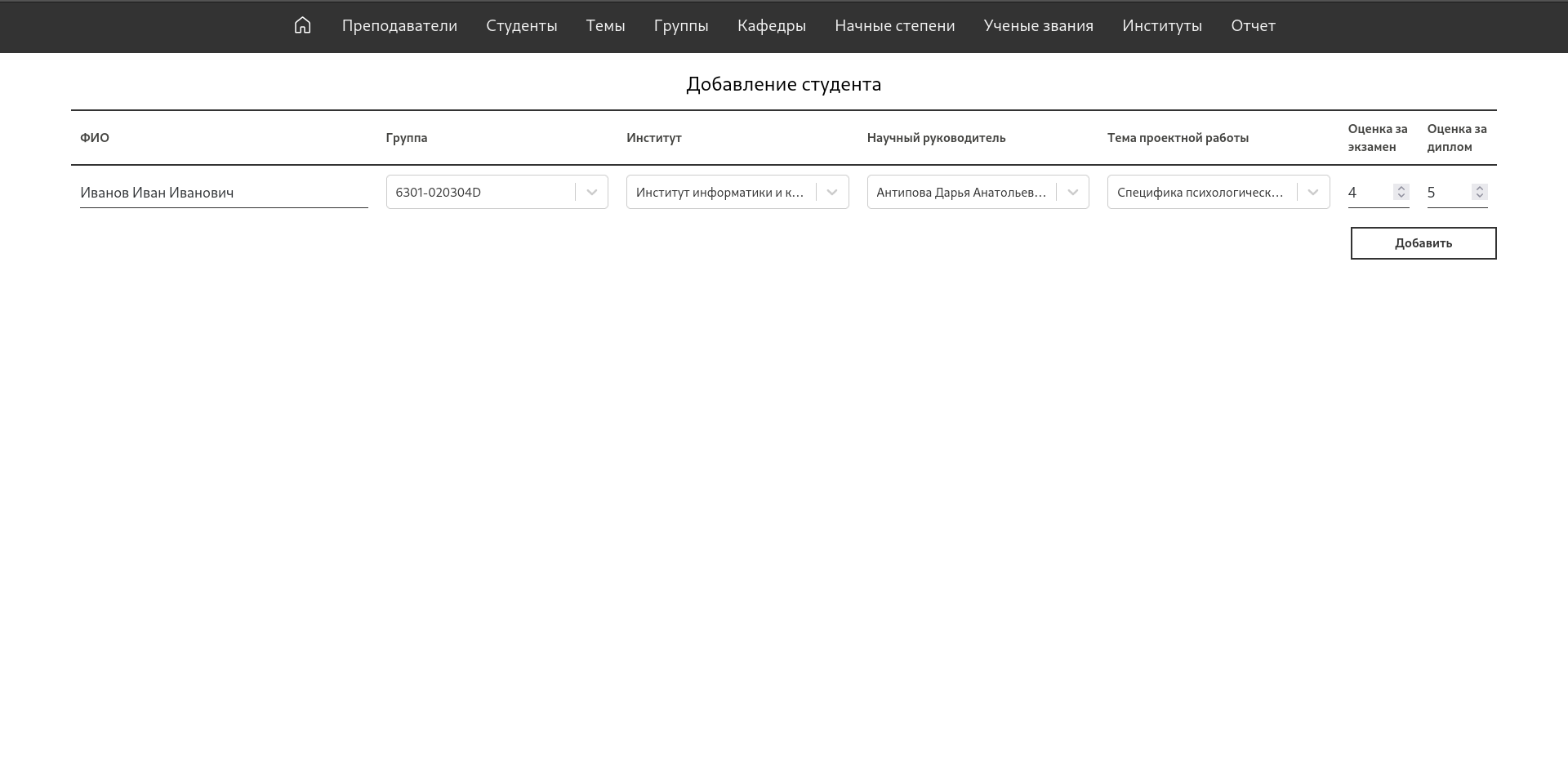


Рисунок 3.4 - Форма добавления студента

1. На рисунке 3.5 представлена обновления студента.

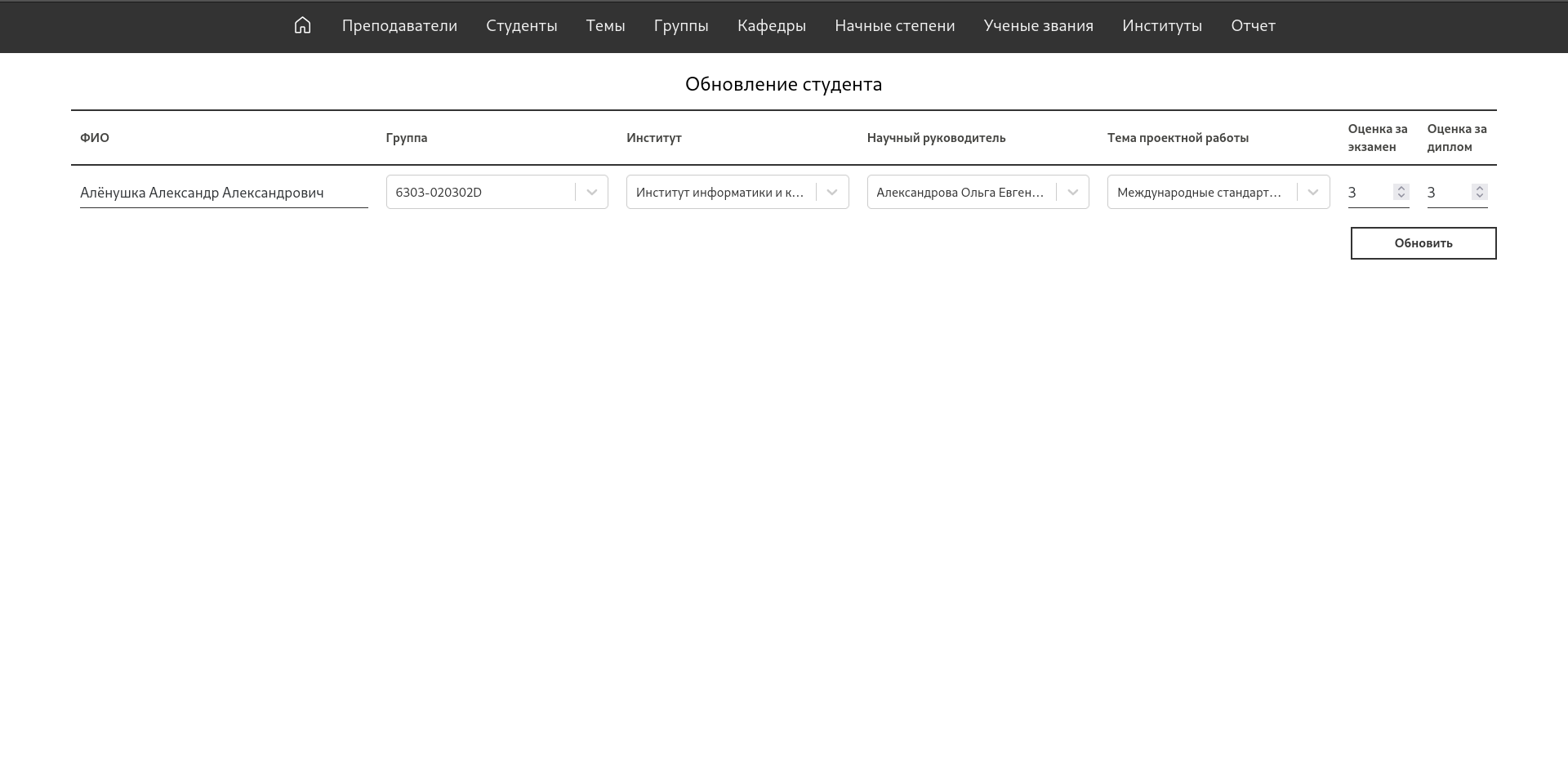


Рисунок 3.5 – Форма обновления студента

1. На рисунке 3.6 представлена форма добавления темы.

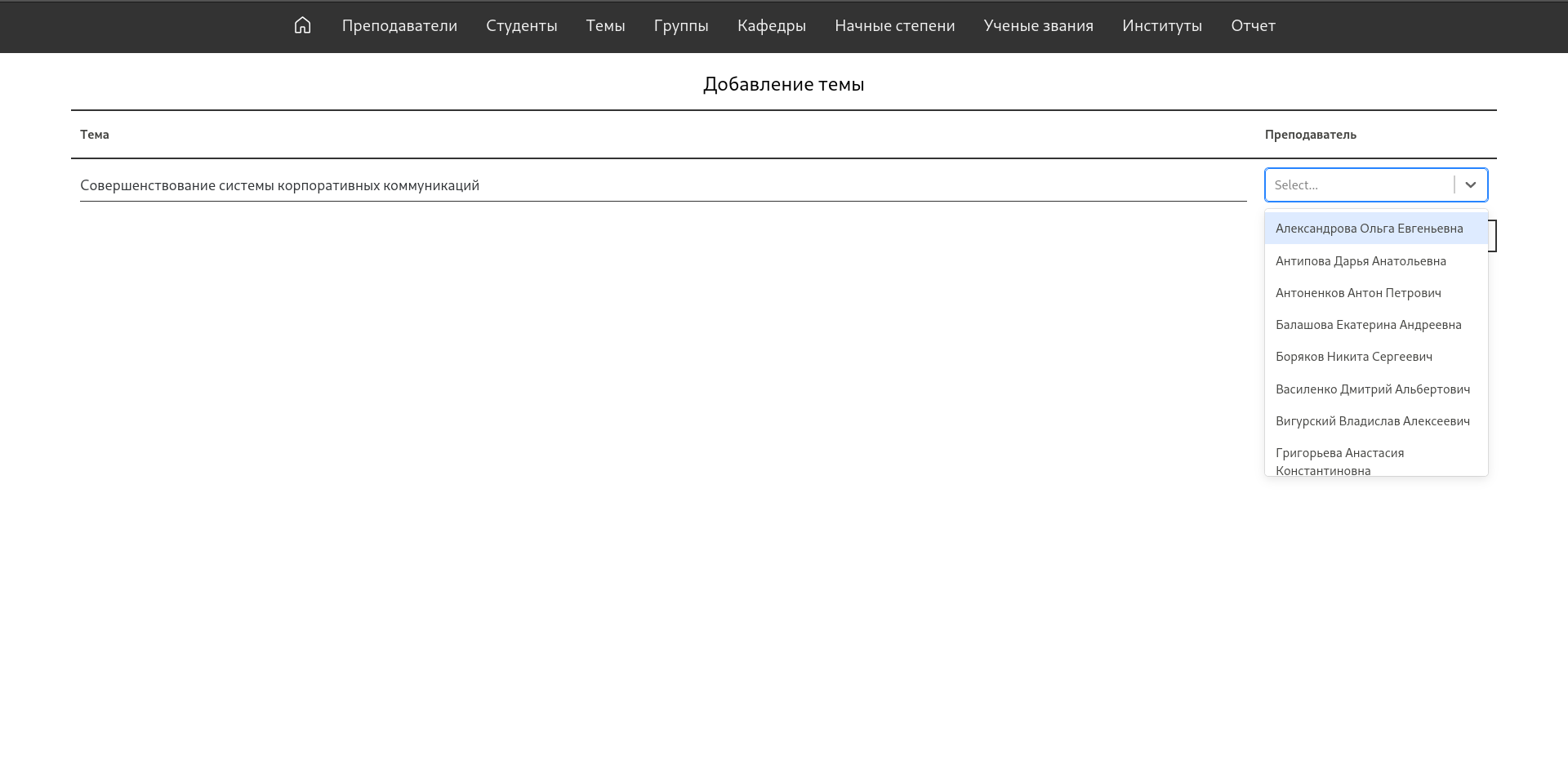


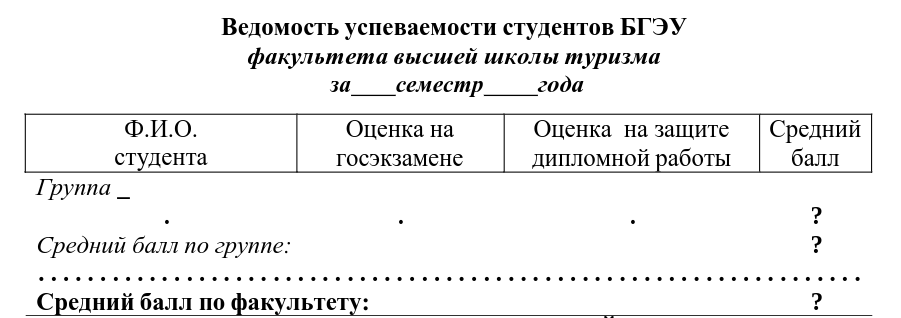
Рисунок 3.6 – Форма добавления темы

Код страниц приведен в приложение C.

## Тема 4. Конструирование отчета

Сконструировать отчет (объект базы данных), вид которого следует выбрать из раздела "Отчеты, выводимые на основе базы данных"(находится в конце сборника заданий). В отчете произвести вычисления в строках и подвести частные и общие итоги (если они указаны).

Вид отчеты:



Визуальное представление вывода отчета:

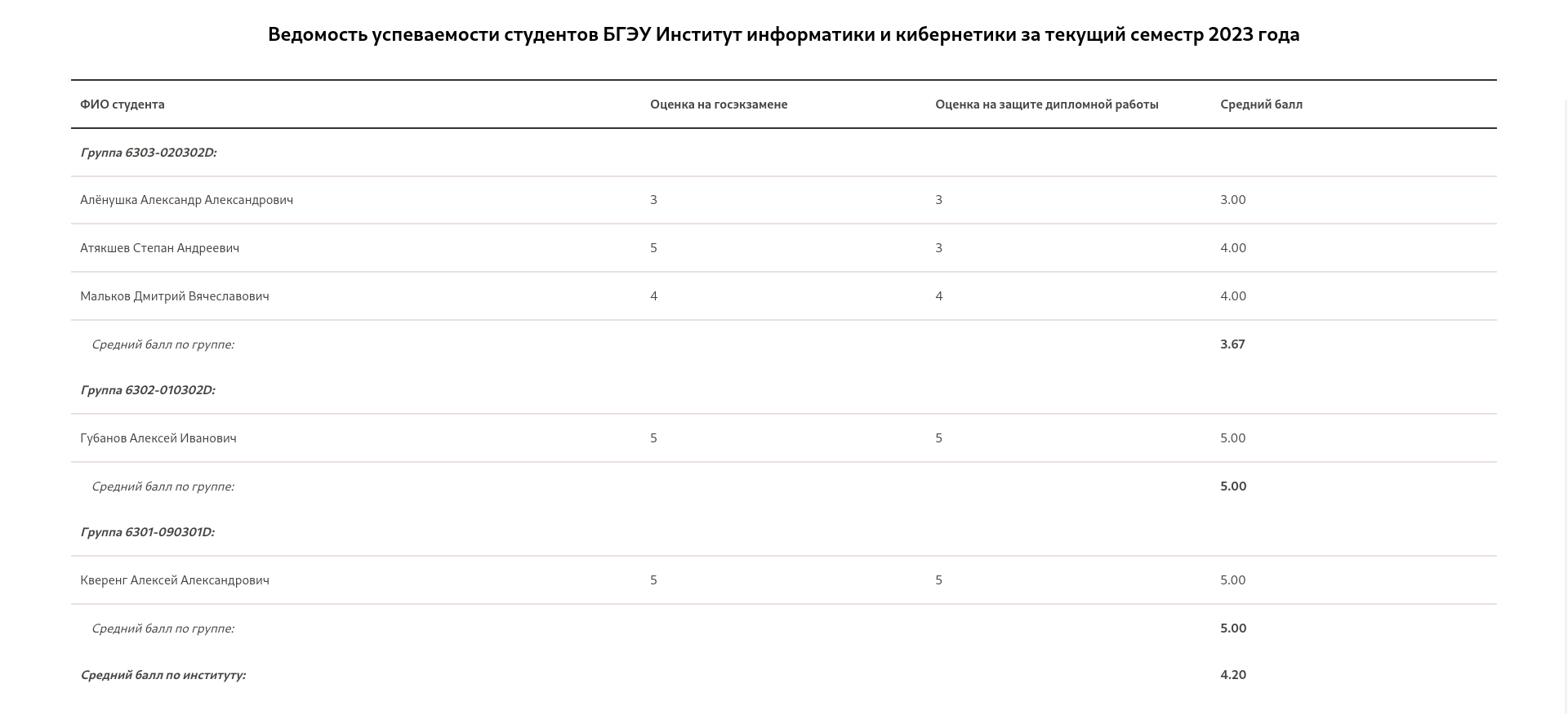
1. На рисунке 4.2 представлен результат вывода отчета.

Рисунок 4.2 - Результат вывода отчета в файл pdf

1. На рисунке 4.2 представлен результат вывода отчета в файл pdf.

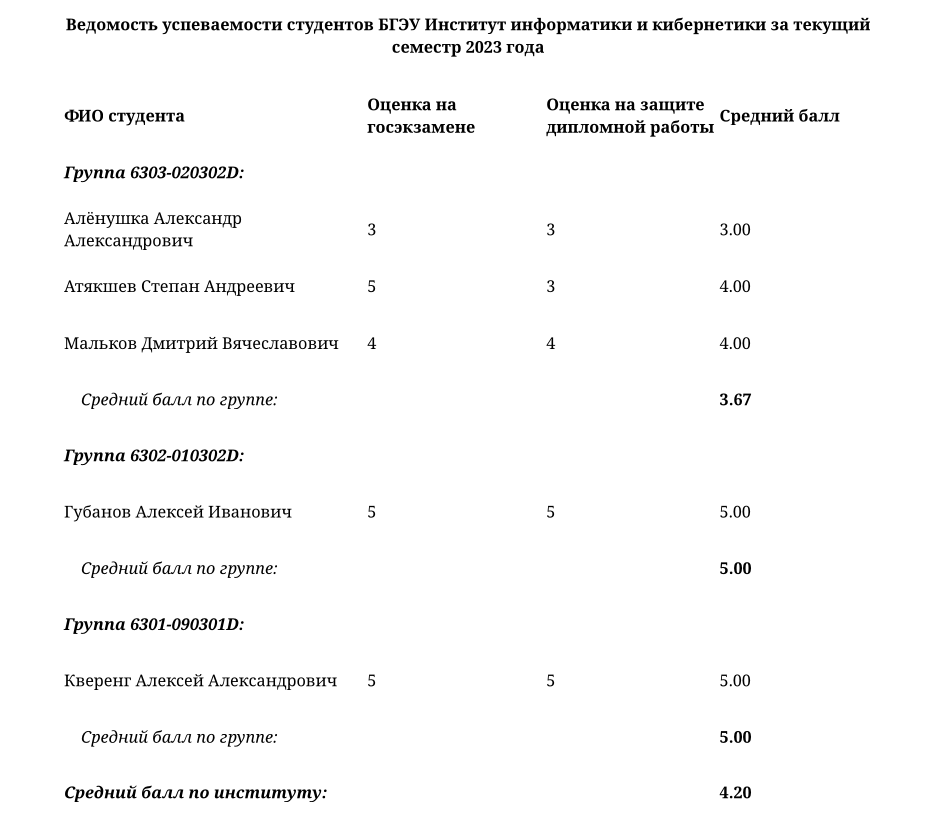


Рисунок 4.2 - Результат вывода отчета в файл pdf

Код страницы выполнения операции приведён в приложении D.

# ПРИЛОЖЕНИЕ А

CREATE SCHEMA IF NOT EXISTS `graduation\_project` DEFAULT CHARACTER SET utf8 ;

USE `graduation\_project` ;

CREATE TABLE IF NOT EXISTS `graduation\_project`.`group` (

`group\_id` INT NOT NULL AUTO\_INCREMENT,

`number` VARCHAR(64) NOT NULL,

PRIMARY KEY (`group\_id`))

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `graduation\_project`.`faculty` (

`faculty\_id` INT NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(128) NOT NULL,

PRIMARY KEY (`faculty\_id`))

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `graduation\_project`.`science\_degree` (

`science\_degree\_id` INT NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(64) NOT NULL,

PRIMARY KEY (`science\_degree\_id`))

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `graduation\_project`.`department` (

`department\_id` INT NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(128) NOT NULL,

PRIMARY KEY (`department\_id`))

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `graduation\_project`.`teacher\_rank` (

`teacher\_rank\_id` INT NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(128) NOT NULL,

PRIMARY KEY (`teacher\_rank\_id`))

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `graduation\_project`.`teacher` (

`teacher\_id` INT NOT NULL AUTO\_INCREMENT,

`full\_name` VARCHAR(64) NOT NULL,

`phone` DECIMAL(11) NOT NULL,

`email` VARCHAR(64) NOT NULL,

`science\_degree\_id` INT NOT NULL,

`department\_id` INT NOT NULL,

`teacher\_rank\_id` INT NOT NULL,

PRIMARY KEY (`teacher\_id`),

INDEX `fk\_teacher\_science\_degree1\_idx` (`science\_degree\_id` ASC) VISIBLE,

INDEX `fk\_teacher\_department1\_idx` (`department\_id` ASC) VISIBLE,

INDEX `fk\_teacher\_teacher\_rank1\_idx` (`teacher\_rank\_id` ASC) VISIBLE,

CONSTRAINT `fk\_teacher\_science\_degree1`

FOREIGN KEY (`science\_degree\_id`)

REFERENCES `graduation\_project`.`science\_degree` (`science\_degree\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_teacher\_department1`

FOREIGN KEY (`department\_id`)

REFERENCES `graduation\_project`.`department` (`department\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_teacher\_teacher\_rank1`

FOREIGN KEY (`teacher\_rank\_id`)

REFERENCES `graduation\_project`.`teacher\_rank` (`teacher\_rank\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `graduation\_project`.`topic` (

`topic\_id` INT NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(128) NOT NULL,

`teacher\_id` INT NOT NULL,

PRIMARY KEY (`topic\_id`, `teacher\_id`),

INDEX `fk\_topic\_teacher1\_idx` (`teacher\_id` ASC) VISIBLE,

CONSTRAINT `fk\_topic\_teacher1`

FOREIGN KEY (`teacher\_id`)

REFERENCES `graduation\_project`.`teacher` (`teacher\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `graduation\_project`.`student` (

`student\_id` INT NOT NULL AUTO\_INCREMENT,

`full\_name` VARCHAR(64) NOT NULL,

`group\_id` INT NOT NULL,

`faculty\_id` INT NOT NULL,

`topic\_id` INT,

PRIMARY KEY (`student\_id`, `topic\_id`),

UNIQUE INDEX `student\_id\_UNIQUE` (`student\_id` ASC) VISIBLE,

INDEX `fk\_student\_group\_idx` (`group\_id` ASC) VISIBLE,

INDEX `fk\_student\_faculty1\_idx` (`faculty\_id` ASC) VISIBLE,

INDEX `fk\_student\_topic1\_idx` (`topic\_id` ASC) VISIBLE,

CONSTRAINT `fk\_student\_group`

FOREIGN KEY (`group\_id`)

REFERENCES `graduation\_project`.`group` (`group\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_student\_faculty1`

FOREIGN KEY (`faculty\_id`)

REFERENCES `graduation\_project`.`faculty` (`faculty\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_student\_topic1`

FOREIGN KEY (`topic\_id`)

REFERENCES `graduation\_project`.`topic` (`topic\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `graduation\_project`.`grade` (

`grade\_id` INT NOT NULL AUTO\_INCREMENT,

`exam` INT,

`diploma` INT,

`student\_id` INT NOT NULL,

PRIMARY KEY (`grade\_id`, `student\_id`),

INDEX `fk\_grade\_student1\_idx` (`student\_id` ASC) VISIBLE,

CONSTRAINT `fk\_grade\_student1`

FOREIGN KEY (`student\_id`)

REFERENCES `graduation\_project`.`student` (`student\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

INSERT INTO science\_degree (name) VALUES

('Кандидат наук'),

('Доктор наук');

INSERT INTO teacher\_rank (name) VALUES

('Доцент'),

('Профессор');

INSERT INTO graduation\_project.group (number) VALUES

('6301-020302D'),

('6302-020302D'),

('6303-020302D'),

('6301-010302D'),

('6302-010302D'),

('6303-010302D'),

('6301-090301D'),

('6302-090301D'),

('6303-090301D');

INSERT INTO faculty (name) VALUES

('Институт авиационной и ракетно-космической техники'),

('Институт двигателей и энергетических установок'),

('Естественнонаучный институт'),

('Cоциально-гуманитарный институт'),

('Институт информатики и кибернетики'),

('Институт экономики и управления'),

('Юридический институт');

INSERT INTO department (name) VALUES

('Кафедра динамики полета и систем управления'),

('Кафедра обработки металлов давлением'),

('Кафедра автоматических систем энергетических установок'),

('Кафедра инженерной графики'),

('Кафедра химии'),

('Кафедра философии'),

('Кафедра суперкомпьютеров и общей информатики'),

('Кафедра лазерных и биотехнических систем'),

('Кафедра государственного и муниципального управления'),

('Кафедра государственного и административного права');

INSERT INTO teacher (full\_name, phone, email, science\_degree\_id, department\_id, teacher\_rank\_id) VALUES

('Александрова Ольга Евгеньевна', 79277732111, 'oe@gmail.com', 1, 1, 2),

('Антипова Дарья Анатольевна', 79272422456, 'da@gmail.com', 1, 2, 2),

('Антоненков Антон Петрович', 79278144222, 'ap@gmail.com', 2, 3, 2),

('Балашова Екатерина Андреевна', 79442234444, 'ea@gmail.com', 2, 4, 1),

('Боряков Никита Сергеевич', 79270021455, 'nc@gmail.com', 2, 5, 1),

('Василенко Дмитрий Альбертович', 79278856622, 'vdd@gmail.com', 1, 6, 2),

('Вигурский Владислав Алексеевич', 79279782665, 'va@gmail.com', 2, 7, 2),

('Григорьева Анастасия Константиновна', 79270099133, 'ak@gmail.com', 2, 8, 2),

('Гриднева Виктория Александровна', 79271656735, 'via@gmail.com', 1, 9, 2),

('Данилов Андрей Дмитриевич', 79277775333, 'and@gmail.com', 1, 10, 1),

('Жукова Анна Валерьевна', 79276676322, 'anv@gmail.com', 2, 1, 2),

('Иванов Владислав Денисович', 79112566433, 'vld@gmail.com', 1, 2, 1),

('Лебедев Антон Евгеньевич', 79279999999, 'ane@gmail.com', 1, 3, 1),

('Кулемин Матвей Валерьевич', 79271231322, 'mav@gmail.com', 1, 4, 1),

('Недугов Владимир Германович', 79275521445, 'vlg@gmail.com', 2, 5, 1);

INSERT INTO topic (name, teacher\_id) VALUES

('Многоуровневый анализ, оценка и пути оптимизации степени производительности веб-приложений.', 1),

('Основы и особенности разработки системы управления базами данных, используемых в электронной коммерции.', 1),

('Разработка, тестирование и отладка архитектуры систем облачных вычислений для единичного предприятия', 3),

('Исследование, изучение и основы разработки криптографических протоколов, используемых для защиты требуемых данных.', 3),

('Создание автоматизированных систем тестирования программного обеспечения.', 4),

('Исследование, создание, тестирование и запуск методики биометрической идентификации.', 5),

('Сбор, анализ, обработка и надежное хранение огромного количества данных из медицинской сферы деятельности.', 7),

('Создание и отладка систем автоматизации управления различными проектами.', 7),

('Создание систем управляющих контентом любых/заданных веб-сайтов.', 7),

('Изучение, анализ степени эффективности и пути эффективной оптимизации функционирование заданных компьютерных сетей.', 8),

('Изучение, разработка и окончательное тестирование алгоритмов для осуществления компьютерного распознавания лиц (зрения/видения).', 9),

('Разработка эффективных и надежных схем документооборота для единичного крупного предприятия или организации.', 10),

('Изучение и разработка методик виртуализации заданных серверов.', 13),

('Изучение и создание эффективных методик оптимизации потребления электроэнергии в компьютерных сетях и системах.', 15),

('Автоматизированная система работы поликлиники ', 15);

INSERT INTO student (full\_name, group\_id, faculty\_id, topic\_id) VALUES

('Алёнушка Александр Александрович', 1, 1, 1),

('Атякшев Степан Андреевич', 2, 2, 2),

('Голобурдина Дарья Олеговна', 3, 3, 3),

('Гребенщиков Данила Александрович', 4, 4, 4),

('Губанов Алексей Иванович', 5, 5, 5),

('Изюмова Мария Михайловна', 6, 6, 6),

('Кверенг Алексей Александрович', 7, 7, 7),

('Климов Алексей Сергеевич', 8, 1, 8),

('Колбанов Дмитрий Олегович', 9, 2, 9),

('Комаров Павел Андреевич', 1, 3, 10),

('Лукашевич Роман Константинович', 2, 4, 11),

('Мальков Дмитрий Вячеславович', 3, 5, 12),

('Матвеев Виктор Дмитриевич', 4, 6, 13),

('Некрасов Никита Сергеевич', 5, 7, 14),

('Репп Михаил Александрович', 6, 1, 15);

INSERT INTO grade (exam, diploma, student\_id) VALUES

(5, 5, 1),

(4, 5, 2),

(4, 5, 3),

(3, 3, 4),

(5, 5, 5),

(4, 5, 6),

(5, 5, 7),

(5, 4, 8),

(4, 5, 9),

(4, 5, 10),

(5, 5, 11),

(4, 4, 12),

(5, 5, 13),

(5, 4, 14),

(4, 5, 15);

# ПРИЛОЖЕНИЕ B

const db = require("./db");

class Controller {

async createTeacher(req, res) {

const {

full\_name,

phone,

email,

science\_degree\_id,

department\_id,

teacher\_rank\_id,

} = req.body;

await db.query(

`INSERT INTO teacher (full\_name, phone, email, science\_degree\_id, department\_id, teacher\_rank\_id) VALUES

(?, ?, ?, ?, ?, ?)`,

[

full\_name,

phone,

email,

science\_degree\_id,

department\_id,

teacher\_rank\_id,

],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async updateTeacher(req, res) {

const id = +req.params.id;

const {

full\_name,

phone,

email,

science\_degree\_id,

department\_id,

teacher\_rank\_id,

} = req.body;

await db.query(

`UPDATE teacher SET

full\_name = ?, phone = ?, email = ?, science\_degree\_id = ?, department\_id = ?, teacher\_rank\_id = ?

WHERE teacher\_id = ?`,

[

full\_name,

phone,

email,

science\_degree\_id,

department\_id,

teacher\_rank\_id,

id,

],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async deleteTeacher(req, res) {

const id = +req.params.id;

await db.query(

`DELETE FROM topic WHERE teacher\_id = ?`,

[id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

db.query(

`DELETE FROM teacher WHERE teacher\_id = ?`,

[id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

);

}

async getTeachers(req, res) {

await db.query(

`SELECT teacher\_id, full\_name, phone, email, science\_degree.name science\_degree, department.name department, teacher\_rank.name teacher\_rank

FROM teacher

JOIN science\_degree USING(science\_degree\_id)

JOIN department USING(department\_id)

JOIN teacher\_rank USING(teacher\_rank\_id)

ORDER BY full\_name`,

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async getTeacherRaw(req, res) {

const id = +req.params.id;

await db.query(

`SELECT \* FROM teacher

WHERE teacher\_id = ?`,

[id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async createStudent(req, res) {

const { full\_name, group\_id, faculty\_id, topic\_id, exam, diploma } =

req.body;

await db.query(

`INSERT INTO student (full\_name, group\_id, faculty\_id, topic\_id) VALUES

(?, ?, ?, ?)`,

[full\_name, group\_id, faculty\_id, topic\_id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

db.query(

`INSERT INTO grade (exam, diploma, student\_id) VALUES

(?, ?, ?)`,

[exam, diploma, result.insertId],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

);

}

async updateStudent(req, res) {

const id = +req.params.id;

const { full\_name, group\_id, faculty\_id, topic\_id, exam, diploma } =

req.body;

await db.query(

`UPDATE student SET

full\_name = ?, group\_id = ?, faculty\_id = ?, topic\_id = ?

WHERE student\_id = ?`,

[full\_name, group\_id, faculty\_id, topic\_id, id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

db.query(

`UPDATE grade SET

exam = ?, diploma = ?

WHERE student\_id = ?`,

[exam, diploma, id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

);

}

async deleteStudent(req, res) {

const id = +req.params.id;

db.query(

`DELETE FROM grade WHERE student\_id = ?`,

[id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

db.query(

`DELETE FROM student WHERE student\_id = ?`,

[id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

);

}

async getStudents(req, res) {

await db.query(

`SELECT student\_id, student.full\_name, \`group\`.number 'group', faculty.name faculty, teacher.full\_name 'teacher', topic.name topic, grade.exam, grade.diploma

FROM student

JOIN \`group\` USING(group\_id)

JOIN faculty USING(faculty\_id)

JOIN topic USING(topic\_id)

JOIN teacher USING(teacher\_id)

JOIN grade USING(student\_id)

ORDER BY student.full\_name`,

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async getStudent(req, res) {

const id = +req.params.id;

await db.query(

`SELECT student\_id, student.full\_name, \`group\`.number 'group', faculty.name faculty, teacher.full\_name 'teacher', topic.name topic, grade.exam, grade.diploma

FROM student

JOIN \`group\` USING(group\_id)

JOIN faculty USING(faculty\_id)

JOIN topic USING(topic\_id)

JOIN teacher USING(teacher\_id)

JOIN grade USING(student\_id)

WHERE student\_id = ?`,

[id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async getStudentRaw(req, res) {

const id = +req.params.id;

await db.query(

`SELECT student\_id, student.full\_name, group\_id, faculty\_id, teacher\_id, topic\_id, grade.exam, grade.diploma

FROM student

JOIN topic USING(topic\_id)

JOIN teacher USING(teacher\_id)

JOIN grade USING(student\_id)

WHERE student\_id = ?`,

[id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async createTopic(req, res) {

const { name, teacher\_id } = req.body;

await db.query(

`INSERT INTO topic (name, teacher\_id) VALUES

(?, ?)`,

[name, teacher\_id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async updateTopic(req, res) {

const id = +req.params.id;

const { name, teacher\_id } = req.body;

await db.query(

`UPDATE topic SET

name = ?, teacher\_id = ?

WHERE topic\_id = ?`,

[name, teacher\_id, id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async deleteTopic(req, res) {

const id = +req.params.id;

await db.query(

`DELETE FROM topic WHERE topic\_id = ?`,

[id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async getTopics(req, res) {

await db.query(

`SELECT topic\_id, name, teacher.full\_name

FROM topic

JOIN teacher USING(teacher\_id)

ORDER BY full\_name`,

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async getTopicRaw(req, res) {

const id = +req.params.id;

await db.query(

`SELECT \* FROM topic

WHERE topic\_id = ?`,

[id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async getTopic(req, res) {

const id = req.params.id;

await db.query(

`SELECT topic\_id, name, teacher.full\_name

FROM topic

JOIN teacher USING(teacher\_id)

WHERE topic\_id = ?`,

[id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async getFreeTeacherTopics(req, res) {

const id = req.params.id;

await db.query(

`SELECT topic.topic\_id, topic.name

FROM topic

LEFT JOIN student USING(topic\_id)

WHERE teacher\_id = ? AND student.topic\_id IS NULL`,

[id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async createDepartment(req, res) {

const { name } = req.body;

await db.query(

`INSERT INTO department (name) VALUES

(?)`,

[name],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async updateDepartment(req, res) {

const id = +req.params.id;

const { name } = req.body;

await db.query(

`UPDATE department SET

name = ?

WHERE department\_id = ?`,

[name, id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async deleteDepartment(req, res) {

const id = +req.params.id;

await db.query(

`DELETE FROM department WHERE department\_id = ?`,

[id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async getDepartments(req, res) {

await db.query(`SELECT \* FROM department`, (err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

});

}

async getDepartmentRaw(req, res) {

const id = +req.params.id;

await db.query(

`SELECT \* FROM department

WHERE department\_id = ?`,

[id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async createFaculty(req, res) {

const { name } = req.body;

await db.query(

`INSERT INTO faculty (name) VALUES

(?)`,

[name],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async updateFaculty(req, res) {

const id = +req.params.id;

const { name } = req.body;

await db.query(

`UPDATE faculty SET

name = ?

WHERE faculty\_id = ?`,

[name, id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async deleteFaculty(req, res) {

const id = +req.params.id;

await db.query(

`DELETE FROM faculty WHERE faculty\_id = ?`,

[id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async getFaculties(req, res) {

await db.query(`SELECT \* FROM faculty`, (err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

});

}

async getFacultyRaw(req, res) {

const id = +req.params.id;

await db.query(

`SELECT \* FROM faculty

WHERE faculty\_id = ?`,

[id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async createGroup(req, res) {

const { number } = req.body;

await db.query(

`INSERT INTO \`group\` (number) VALUES

(?)`,

[number],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async updateGroup(req, res) {

const id = +req.params.id;

const { number } = req.body;

await db.query(

`UPDATE \`group\` SET

number = ?

WHERE group\_id = ?`,

[number, id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async deleteGroup(req, res) {

const id = +req.params.id;

await db.query(

`DELETE FROM \`group\` WHERE group\_id = ?`,

[id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async getGroups(req, res) {

await db.query(`SELECT \* FROM \`group\``, (err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

});

}

async getGroupRaw(req, res) {

const id = +req.params.id;

await db.query(

`SELECT \* FROM \`group\`

WHERE group\_id = ?`,

[id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async createScienceDegree(req, res) {

const { name } = req.body;

await db.query(

`INSERT INTO science\_degree (name) VALUES

(?)`,

[name],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async updateScienceDegree(req, res) {

const id = +req.params.id;

const { name } = req.body;

await db.query(

`UPDATE science\_degree SET

name = ?

WHERE science\_degree\_id = ?`,

[name, id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async deleteScienceDegree(req, res) {

const id = +req.params.id;

await db.query(

`DELETE FROM science\_degree WHERE science\_degree\_id = ?`,

[id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async getScienceDegrees(req, res) {

await db.query(`SELECT \* FROM science\_degree`, (err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

});

}

async getScienceDegreeRaw(req, res) {

const id = +req.params.id;

await db.query(

`SELECT \* FROM science\_degree

WHERE science\_degree\_id = ?`,

[id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async createTeacherRank(req, res) {

const { name } = req.body;

await db.query(

`INSERT INTO teacher\_rank (name) VALUES

(?)`,

[name],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async updateTeacherRank(req, res) {

const id = +req.params.id;

const { name } = req.body;

await db.query(

`UPDATE teacher\_rank SET

name = ?

WHERE teacher\_rank\_id = ?`,

[name, id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async deleteTeacherRank(req, res) {

const id = +req.params.id;

await db.query(

`DELETE FROM teacher\_rank WHERE teacher\_rank\_id = ?`,

[id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

async getTeacherRanks(req, res) {

await db.query(`SELECT \* FROM teacher\_rank`, (err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

});

}

async getTeacherRankRaw(req, res) {

const id = +req.params.id;

await db.query(

`SELECT \* FROM teacher\_rank

WHERE teacher\_rank\_id = ?`,

[id],

(err, result, field) => {

if (err) {

console.log(err);

return res.status(500).json({ message: "Ошибка" });

}

res.json(result);

}

);

}

}

module.exports = new Controller();

# ПРИЛОЖЕНИЕ C

Страница вывода информации о преподавателях

import React, { Component } from "react";

import { withSnackbar } from "react-simple-snackbar";

import DataService from "../../services/DataService.js";

import { Link } from "react-router-dom";

class TeacherTable extends Component {

constructor(params) {

super(params);

this.state = {

teachers: [],

};

}

componentDidMount() {

this.retrieve();

}

retrieve() {

DataService.getTeachers()

.then((res) => {

this.setState({

teachers: res.data,

});

})

.catch((err) => {

console.log(err);

});

}

render() {

const { teachers } = this.state;

return (

<>

<h1 className="table\_\_title">Преподаватели</h1>

<Link className="add-button" to={"/create-teacher"}>

Добавить

</Link>

<table className="table">

<thead>

<tr>

<th width="19%">ФИО</th>

<th width="10%">Телефон</th>

<th width="13%">Email</th>

<th width="13%">Научная степень</th>

<th width="24%">Кафедра</th>

<th width="11%">Ученое звание</th>

<th width="3.5%"></th>

<th width="3.5%"></th>

</tr>

</thead>

<tbody>

{teachers.map((teacher) => (

<tr

key={teacher.teacher\_id}

className="table\_\_body-row"

>

<td>{teacher.full\_name}</td>

<td>{teacher.phone}</td>

<td>{teacher.email}</td>

<td>{teacher.science\_degree}</td>

<td>{teacher.department}</td>

<td>{teacher.teacher\_rank}</td>

<td>

<Link

className="remove-button"

to={`/update-teacher/${teacher.teacher\_id}`}

>

<svg

xmlns="http://www.w3.org/2000/svg"

x="0px"

y="0px"

width="20"

height="20"

viewBox="0 0 30 30"

>

<path d="M 22.828125 3 C 22.316375 3 21.804562 3.1954375 21.414062 3.5859375 L 19 6 L 24 11 L 26.414062 8.5859375 C 27.195062 7.8049375 27.195062 6.5388125 26.414062 5.7578125 L 24.242188 3.5859375 C 23.851688 3.1954375 23.339875 3 22.828125 3 z M 17 8 L 5.2597656 19.740234 C 5.2597656 19.740234 6.1775313 19.658 6.5195312 20 C 6.8615312 20.342 6.58 22.58 7 23 C 7.42 23.42 9.6438906 23.124359 9.9628906 23.443359 C 10.281891 23.762359 10.259766 24.740234 10.259766 24.740234 L 22 13 L 17 8 z M 4 23 L 3.0566406 25.671875 A 1 1 0 0 0 3 26 A 1 1 0 0 0 4 27 A 1 1 0 0 0 4.328125 26.943359 A 1 1 0 0 0 4.3378906 26.939453 L 4.3632812 26.931641 A 1 1 0 0 0 4.3691406 26.927734 L 7 26 L 5.5 24.5 L 4 23 z"></path>

</svg>

</Link>

</td>

<td>

<Link

className="remove-button"

onClick={(e) => {

DataService.deleteTeacher(

teacher.teacher\_id

)

.then(() => {

this.props.openSnackbar(

"Успешно",

5000

);

this.retrieve();

})

.catch(() => {

this.props.openSnackbar(

"Невозможно",

5000

);

});

}}

>

<svg

width="16"

height="16"

viewBox="0 0 16 16"

>

<path

fillRule="evenodd"

clipRule="evenodd"

d="M4.11 2.697L2.698 4.11 6.586 8l-3.89 3.89 1.415 1.413L8 9.414l3.89 3.89 1.413-1.415L9.414 8l3.89-3.89-1.415-1.413L8 6.586l-3.89-3.89z"

></path>

</svg>

</Link>

</td>

</tr>

))}

</tbody>

</table>

</>

);

}

}

export default withSnackbar(TeacherTable);

# ПРИЛОЖЕНИЕ D

import React, { PureComponent } from "react";

import DataService from "../services/DataService.js";

import { Link } from "react-router-dom";

import printJS from "print-js";

import Select from "react-select";

export class MyReport extends PureComponent {

constructor(params) {

super(params);

this.state = {

isVisible: "hidden",

students: [],

groups: [],

faculty\_id: null,

faculty\_name: "",

options\_faculty: [],

};

}

componentDidMount() {

this.retrieve();

}

retrieve() {

DataService.getFaculties()

.then((res) => {

this.setState({

options\_faculty: res.data.map((item) => {

return {

value: item.faculty\_id,

label: item.name,

};

}),

});

})

.catch((err) => {

console.log(err);

});

DataService.getStudents()

.then((res) => {

this.setState({

students: res.data,

});

})

.catch((err) => {

console.log(err);

});

DataService.getGroups()

.then((res) => {

this.setState({

groups: res.data,

});

})

.catch((err) => {

console.log(err);

});

}

updateReport() {}

reportBody() {

let students = this.state.students;

let groups = this.state.groups;

students = students.filter(

(item) => item.faculty === this.state.faculty\_name

);

groups = groups.filter((group) =>

students.find((student) => student.group === group.number)

);

return (

<>

{groups.map((group) => {

const groupStudent = students.filter(

(item) => group.number === item.group

);

return (

<>

<tr>

<td width="40%">

<b>

<em>Группа {group.number}:</em>

</b>

</td>

<td width="20%"></td>

<td width="20%"></td>

<td width="20%"></td>

</tr>

{groupStudent.map((student) => {

return (

<tr className="table\_\_body-row">

<td width="40%">{student.full\_name}</td>

<td width="20%">{student.exam}</td>

<td width="20%">{student.diploma}</td>

<td width="20%">

{(

(student.exam +

student.diploma) /

2

).toFixed(2)}

</td>

</tr>

);

})}

<tr>

<td width="40%">

<em>

&nbsp;&nbsp;&nbsp;&nbsp;Средний балл по

группе:

</em>

</td>

<td width="20%"></td>

<td width="20%"></td>

<td width="20%">

<b>

{(

groupStudent.reduce(

(accum, item) => {

return (

accum +

(item.exam +

item.diploma) /

2

);

},

0

) / groupStudent.length

).toFixed(2)}

</b>

</td>

</tr>

</>

);

})}

<tr>

<td width="40%">

<b>

<em>Средний балл по институту:</em>

</b>

</td>

<td width="20%"></td>

<td width="20%"></td>

<td width="20%">

<b>

{(

students.reduce((accum, item) => {

return (

accum + (item.exam + item.diploma) / 2

);

}, 0) / students.length

).toFixed(2)}

</b>

</td>

</tr>

</>

);

}

render() {

return (

<>

<h1 className="table\_\_title">Отчет</h1>

<div

style={{

display: "flex",

}}

>

<form>

<label

style={{

fontSize: 16 + "px",

}}

>

Выберите институт:

</label>

<Select

className="selector"

options={this.state.options\_faculty}

value={

this.state.options\_faculty.find((val) => {

return val.value === this.state.faculty\_id;

}) || ""

}

onChange={(option) => {

this.setState({

isVisible: "visible",

faculty\_id: option.value || null,

faculty\_name: option.label || "",

});

}}

/>

</form>

<Link

className="add-button"

onClick={() => {

printJS({

printable: "report",

type: "html",

});

}}

>

Печать

</Link>

</div>

<div

id="report"

style={{

visibility: this.state.isVisible,

}}

>

<br />

<table className="table">

<caption>

<h3 className="table\_\_title">

Ведомость успеваемости студентов БГЭУ{" "}

{this.state.faculty\_name} за текущий семестр

2023 года

</h3>

</caption>

<thead>

<tr>

<td width="40%">

<b>ФИО студента</b>

</td>

<td width="20%">

<b>Оценка на госэкзамене</b>

</td>

<td width="20%">

<b>Оценка на защите дипломной работы</b>

</td>

<td width="20%">

<b>Средний балл</b>

</td>

</tr>

</thead>

<tbody>{this.reportBody()}</tbody>

</table>

</div>

</>

);

}

}