# МИНОБРНАУКИ РОССИИ

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ**

**по дисциплине «Введение в нереляционные базы данных» Тема: Электронный журнал успеваемости (Memcached)**

Студентки гр. 6383 Терещенко В.Н.

Лавренкова Е.Л.

Студент гр. 6382 Жук К.А.

Преподаватель Заславский М.М.

Санкт-Петербург

2019

# ЗАДАНИЕ

Студенты Терещенко В.Н., Лавренкова Е.Л. (группа 6383), Жук К.А. (группа 6382).

Тема проекта: Разработка приложения для электронного журнала успеваемости.

Исходные данные:

Необходимо реализовать приложение для электронного журнала успеваемости, использующее СУБД Memcached.

Содержание пояснительной записки:

«Содержание», «Введение», «Качественные требования к решению», «Сценарий использования», «Модель данных», «Разработанное приложение», «Заключение», «Список использованных источников», «Приложение».

Предполагаемый объем пояснительной записки: Не менее 10 страниц.

Дата выдачи задания:

Дата сдачи ИДЗ:

Дата защиты ИДЗ:

Студентки гр. 6383 Терещенко В.Н.

Лавренкова Е.Л.

Студент гр. 6382 Жук К.А.

Преподаватель Заславский М.М.

# АННОТАЦИЯ

Целью данного курса была разработка приложения в команде на одну из поставленных тем. Была выбрана тема создания приложения для хранения данных об успеваемости студентов на СУБД Memcached. В ходе работы были спроектированы сценарии использования для приложения, обосновано неудобство решения реализации данной темы на Memcached и реализована разработка самого приложения.

# SUMMARY

The purpose of this course was the development an application in a team on one of the set topics. Was chosen the topic of creating an application for storing data on student performance on the Memcached DBMS. In the course of the work, usage scenarios for the application were designed, the inconvenience of solving the implementation of this topic on Memcached was justified, and the development of the application itself was implemented. .

# СОДЕРЖАНИЕ

1. [Качественные требования к решению 5](#_bookmark0)
2. [Сценарии использования 5](#_bookmark1)
   1. [*Макеты пользовательского интерфейса 5*](#_bookmark2)
   2. [*Описание сценариев использования*](#_bookmark3) *10*
      1. [*Сценарий использования страницы «П*](#_bookmark4)*оиск студента» 10*
      2. [*Сценарий использования страницы*](#_bookmark5) *«Статистики» 10*
      3. [*Сценарий использования страницы «Добавление студента»*](#_bookmark6)*… 11*
      4. *Сценарий использования страницы «Экспорт базы»……………………..11*
3. [Модель данных 1](#_bookmark7)2
   1. [*NoSQL модель данных 1*](#_bookmark8)*2*
      1. [*Графическое представление 1*](#_bookmark9)*2*
      2. [*Подробное описание и расчёт объема 1*](#_bookmark10)*2*
      3. [*Примеры запросов 1*](#_bookmark11)*4*
   2. [*SQL модель данных 1*](#_bookmark12)*6*
      1. [*Графическое представление 1*](#_bookmark13)*6*
      2. [*Подробное описание и расчёт объема 1*](#_bookmark14)*6*
      3. [*Примеры запросов 1*](#_bookmark15)*8*
   3. [*Сравнение NoSQL и SQL моделей данных 1*](#_bookmark16)*8*
4. [Разработанное приложение](#_bookmark17) 20
   1. [*Краткое описание*](#_bookmark18) *20*
   2. [*Схема экранов приложения*](#_bookmark19) *20*
   3. [*Использованные технологии*](#_bookmark20) *21*
   4. [*Ссылка на приложение*](#_bookmark21) *21*

[Заключение](#_bookmark22) 22

[Список использованных источников](#_bookmark23) 23

Приложение А 24

Приложение Б…………………………………………………………………………………....25

**ВВЕДЕНИЕ**

Цель работы – создать приложение электронного журнала для хранения информации об успеваемости студентов.

Было решено разработать веб-приложение для просмотра списка студентов, редактирования, добавления и удаления информации, а также просмотра статистики.

# Качественные требования к решению

Требуется реализовать веб-приложение, использующее СУБД Memcached.

# Сценарии использования

# Макеты пользовательского интерфейса

# Макеты пользовательского интерфейса представлены на рис. 1-8.

****

Рисунок 1 – Главная страница



Рисунок 2 – Страница «Добавление студента»

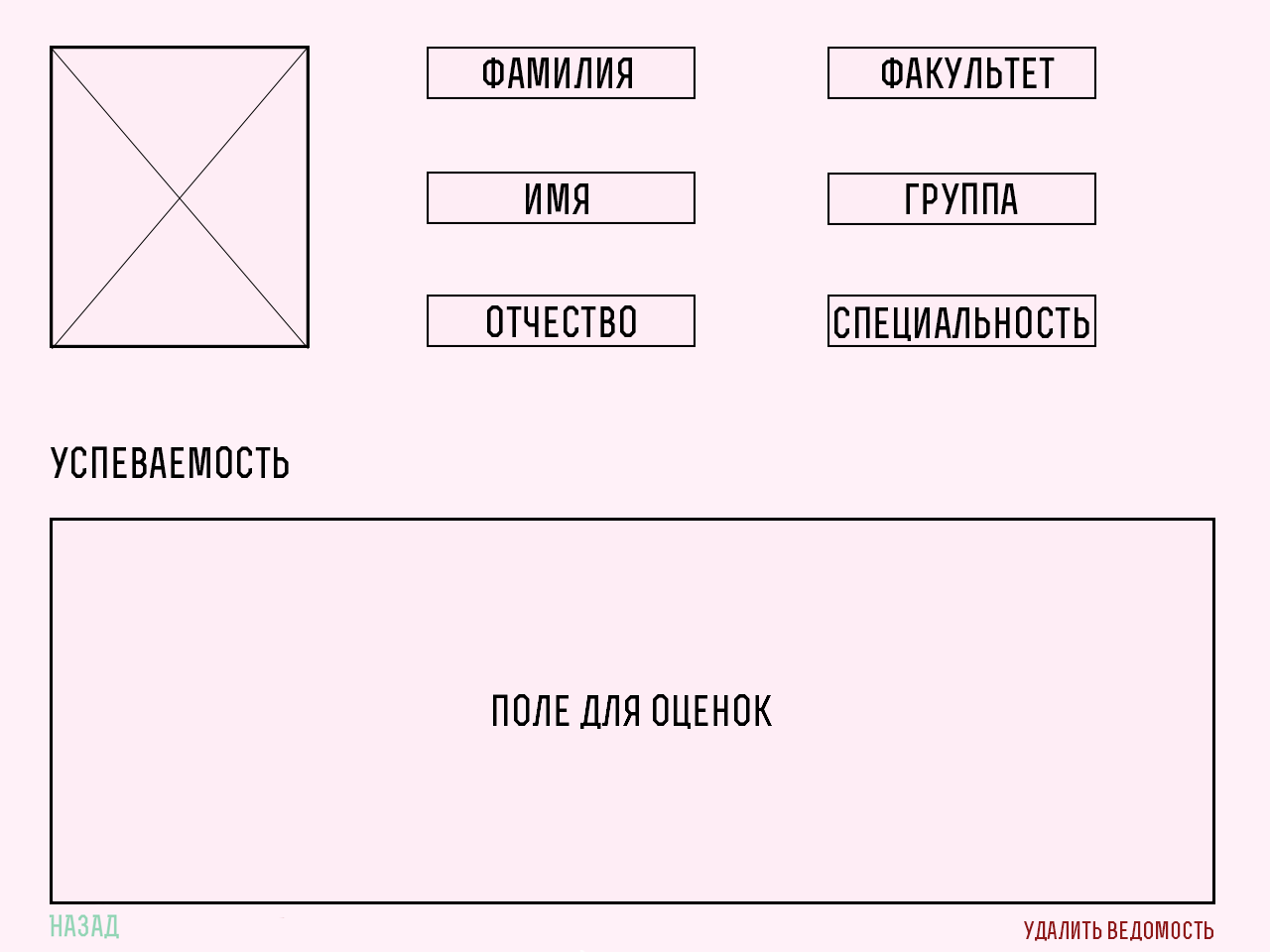


Рисунок 3 – Страница «Сведения о студенте»



Рисунок 4 – Страница «Поиск студента»

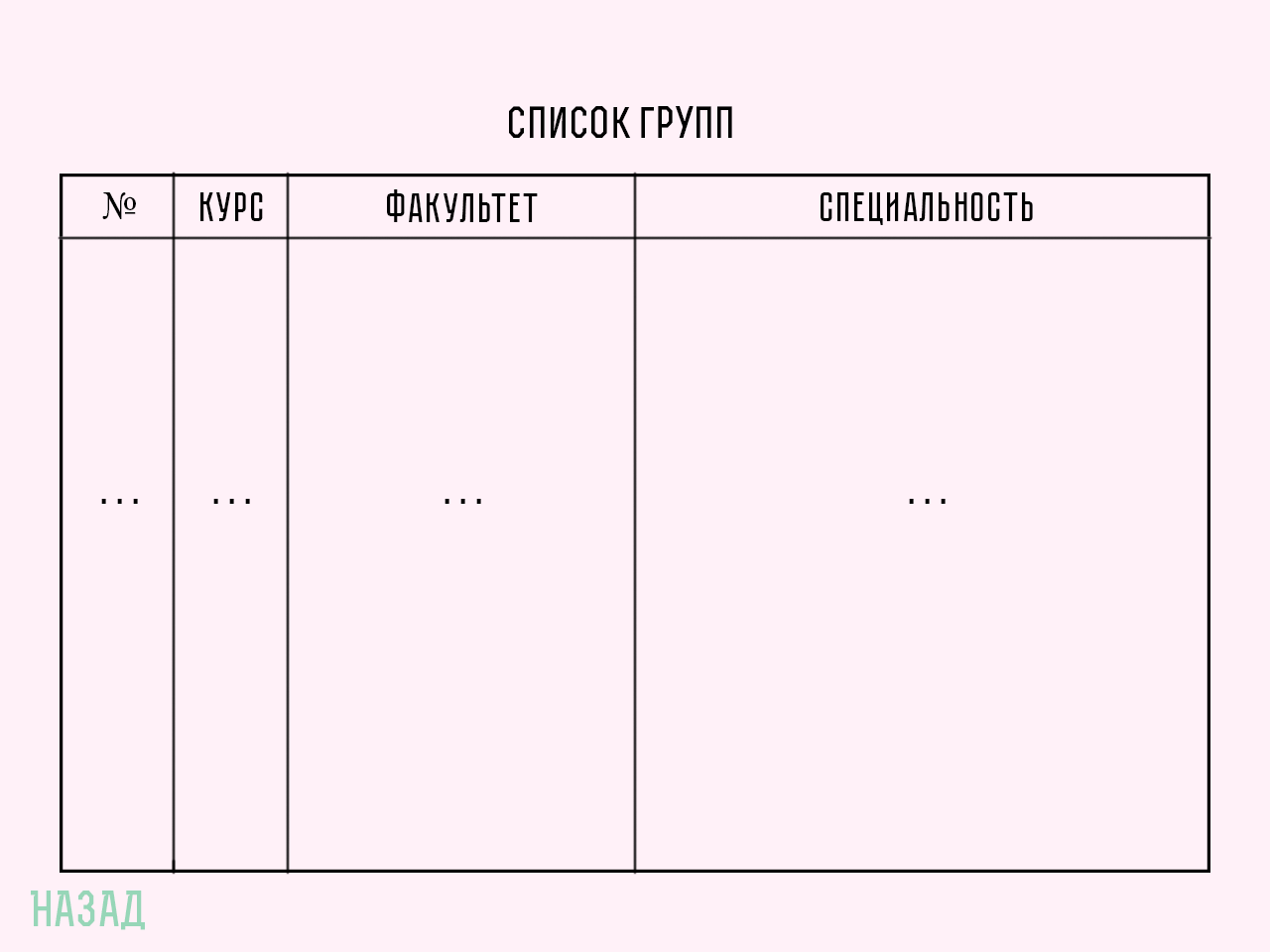


Рисунок 5 — Страница «Список групп»

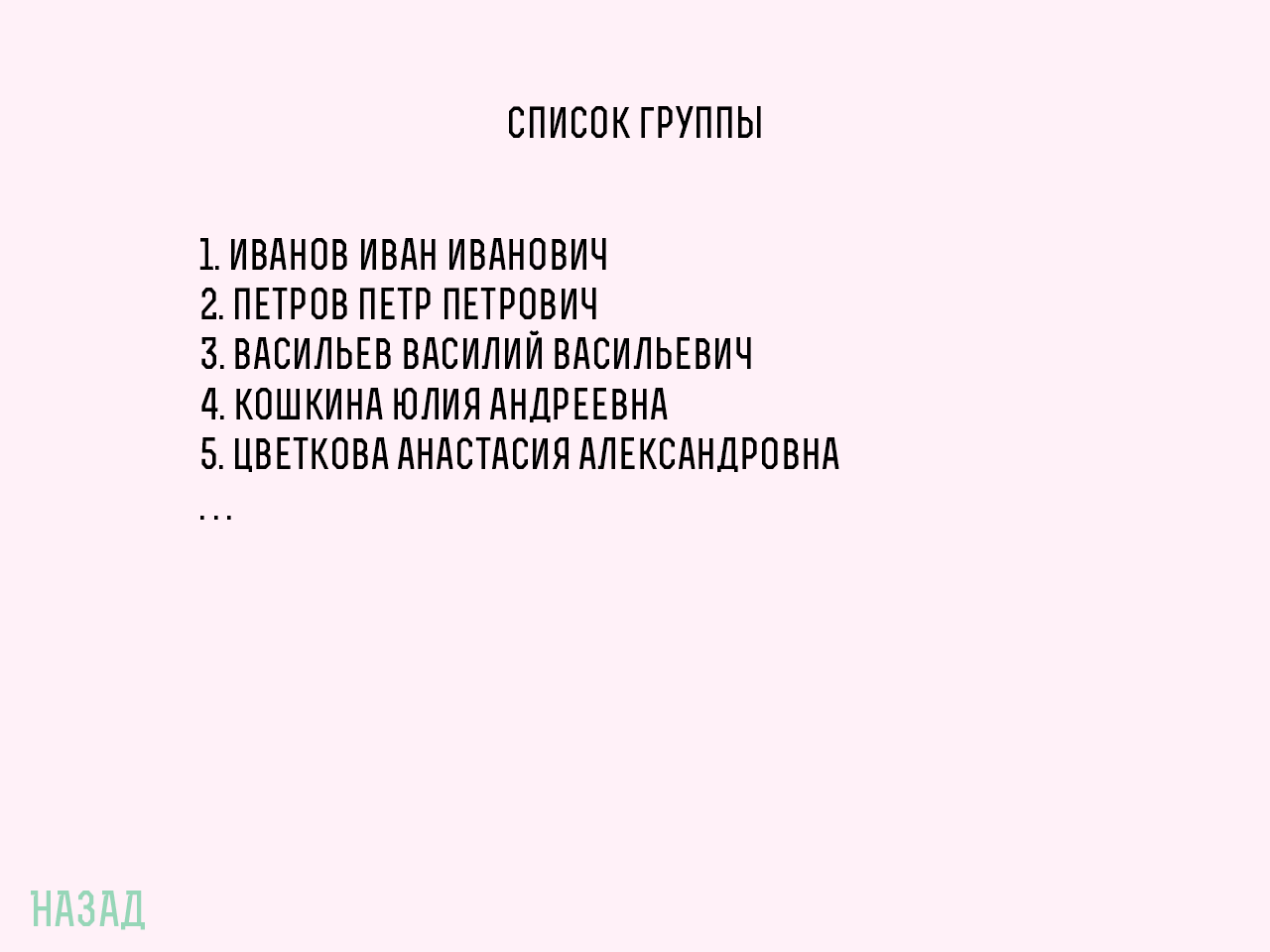


Рисунок 6 — Страница «Список группы»



Рисунок 7 — Страница «Список статистик»

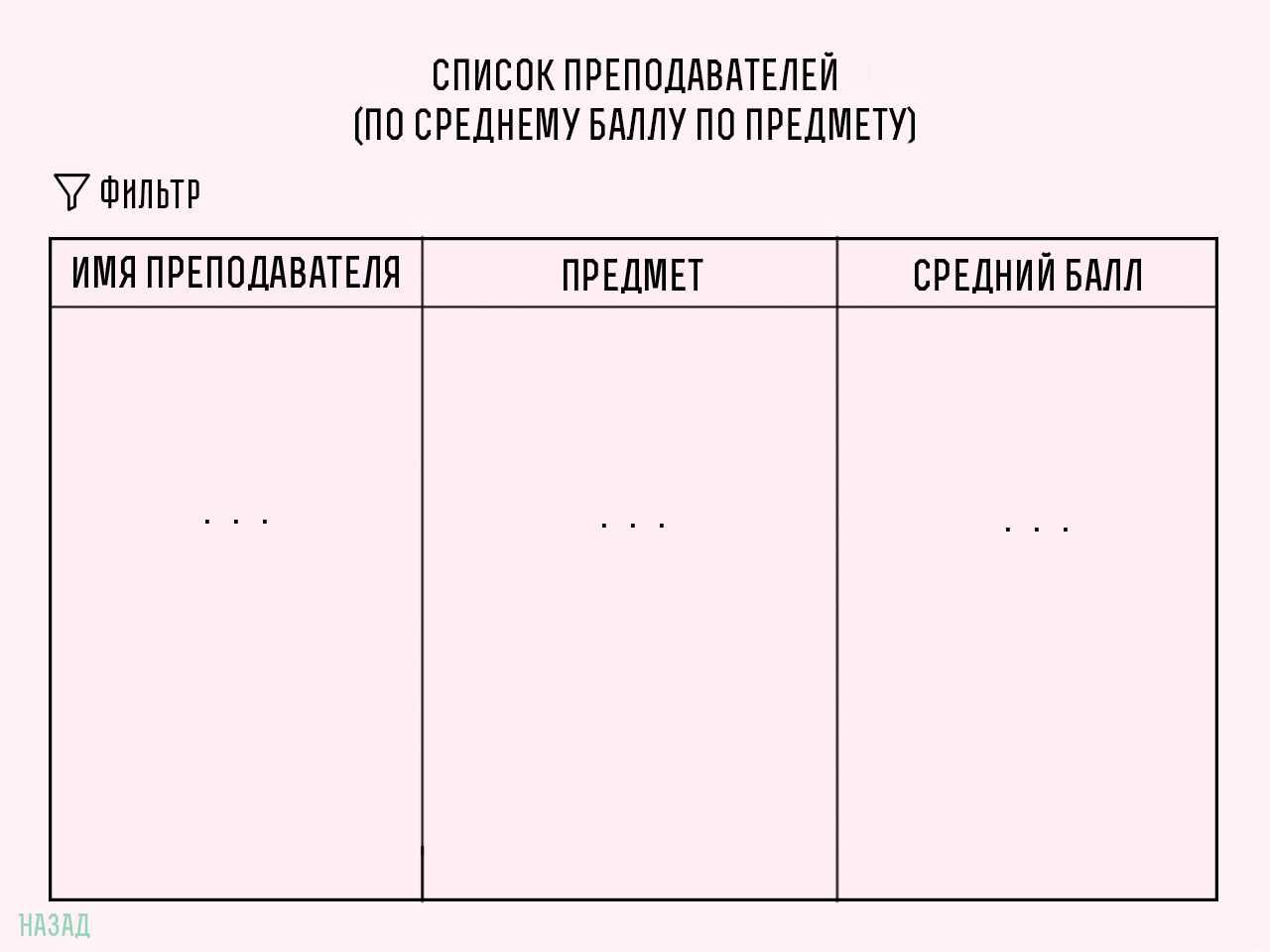


Рисунок 8 — Страница «Статистика»

# Описание сценариев использования

При запуске приложения пользователь оказывается на Главной странице и выбирает нужное ему действие: «Найти студента», «Статистики», «Добавить студента» или «Экспорт базы».

# Сценарий использования страницы «Поиск студента»

1) Пользователю необходимо найти студента, для этого, на главной странице он нажимает на кнопку "Найти студента";

2) после нажатия кнопки, пользователь переходит на страницу "Поиск студента", вводит необходимые данные (факультет, курс, специальность, номер группы (можно несколько и необязательно все)), после этого нажимает на кнопку ниже "Поиск";

3) Пользователь переходит на страницу "Список групп", выбирает нужную группу;

4) далее он попадает на страницу со списком выбранной группы и выбирает студента;

5) после выбора студента Пользователь видит страницу студента, где отображены его данные и успеваемость. На этой странице можно удалить информацию о студенте (страницу студента), если он был отчислен.

# Сценарий использования страницы «Статистики»

1) Пользователю необходимо узнать некую статистику, для этого, на главной странице он нажимает на кнопку "Статистики";

2) после нажатия кнопки, пользователь переходит на страницу "Статистики", где выбирает нужную;

3) далее он переходит на финальную страницу, где изображена информация по статистике в расчете на университет. На этой странице Пользователь может отфильтровать информацию по какому-либо параметру (параметры зависят от конкретной статистики, например, по половой принадлежности). Также некоторые статистики могут содержать диаграммы.

# Сценарий использования страницы «Добавление студента»

1) Пользователь хочет добавить студента. Для этого, на главной странице, он нажимает на кнопку «Добавить студента»;

2) приложение, после нажатия кнопки, отображает страницу «Добавить студента», на которой Пользователь выбирает файл с информацией о студенте, файл с фотографией (не обязательно), нажимает кнопку добавить;

3) приложение проверяет, есть ли уже этот студент в базе данных. Если есть, то всплывает сообщение «Такой студент уже существует!», если нет, то происходит переход на страницу созданного студента.

# Сценарий использования страницы «Экспорт базы данных»

1) Пользователь хочет экспортировать базу данных. Для этого, на главной странице, он нажимает на кнопку «Экспорт базы»;

2) приложение, после нажатия кнопки, запрашивает путь для сохранения и желаемое название и сохраняет базу на ПК.

# МОДЕЛЬ ДАННЫХ

# NoSQL модель данных

# Графическое представление

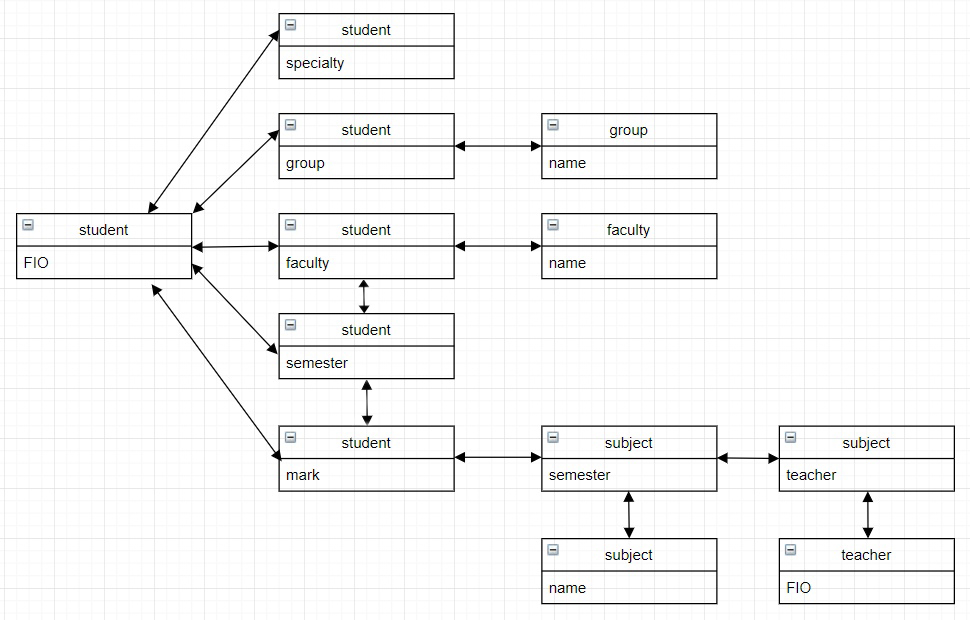
****

Рисунок 9 — Графическое представление NoSQL модели

# Подробное описание и расчёт объема

В нереляционной базе данных содержится 12 коллекций (сущностей):

1. **Student-FIO** – коллекция для хранения студентов и их имён.

* *student* – уникальный идентификатор студента;
* *fio* – имя студента.

1. **Student-Specialty** – коллекция для хранения студентов и их специальностей.

* *student* – уникальный идентификатор студента;
* *specialty\** – специальность студента.

1. **Student-Group** – коллекция для хранения студентов и групп, в которых они учатся.

* *student* – уникальный идентификатор студента;
* *group* – группа студента.

1. **Student-Faculty** – коллекция для хранения студентов и факультетов, на которых они числятся.

* *student* – уникальный идентификатор студента;
* *faculty* – факультет студента.

1. **Student-Semester** – коллекция для хранения студентов и семестров, на котором они обучаются.

* *student* – уникальный идентификатор студента;
* *semester* – семестр, на котором обучается студент.

1. **Student-Mark** – коллекция для хранения студентов и оценок.

* *student* – уникальный идентификатор студента;
* *mark* – оценки студента.

1. **Subject-Semester** – коллекция для хранения предмета и семестра, в котором его преподают.

* *subject* – уникальный идентификатор предмета;
* *semester* – номер семестра.

1. **Subject-Name** – коллекция для хранения предмета и его названия.

* *subject* – уникальный идентификатор предмета;
* *name* – название предмета.

1. **Subject-Teacher** – коллекция для хранения предмета и преподавателя, который его преподаёт.

* *subject* – уникальный идентификатор предмета;
* *teacher* – имя преподавателя.

1. **Teacher-FIO** – коллекция для хранения преподавателя и его имени.

* *teacher* – уникальный идентификатор преподавателя;
* *fio* – имя преподавателя.

1. **Group-Name** – коллекция для хранения группы и её номера.

* *group* – уникальный идентификатор группы;
* *name* – номер группы.

1. **Faculty-Name** – коллекция для хранения факультета и его названия.

* *faculty* – уникальный идентификатор факультета;
* *name* – название факультета.

*Тип всех данных –****string****.*

Для упрощения счёта все коллекции представлены в единичном экземпляре.

String = 50 Б, размер каждой коллекции будет составлять 100 Б.

Возьмём количество студентов, равное S:

«Чистый» объём: 7 × 50 × S = 350S Б.

Фактический объём: 12 × 50 × S + 600 = 600S + 600 Б.

Избыточность модели: 600(S + 1)/350S.

# Примеры запросов

* Для добавления студента:

client.set('student', student);

* Для поиска студента:

client.get('std1','ФИО);

* Для статистик:

client.get('stat1',function(err,val){ return(val); })

# SQL модель данных

# Графическое представление

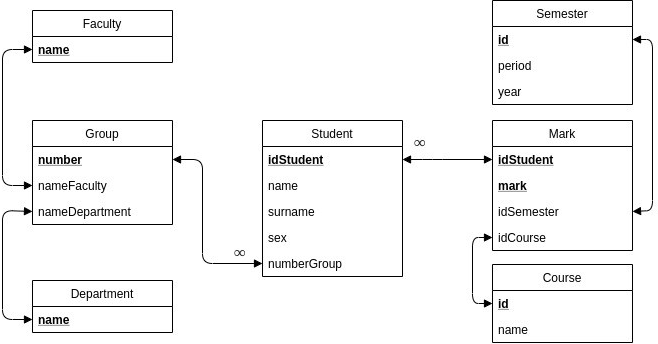


Рисунок 10 — Графическое представление SQL модели

# Подробное описание и расчёт объема

В реляционной базе данных содержится 5 сущностей:

1. **Student** – сущность, представляющая студента.

* ExamBookNum – уникальный идентификатор студента (номер зачётной книжки), тип: int;
* Name – имя студента, тип: string;
* Surname – фамилия студента, тип: string;
* FathersName – отчество студента, тип: string;
* Group – номер группы студента, тип: int.

1. **Subject** – сущность, представляющая предмет.

* idSubject – уникальный идентификатор предмета, тип: int;
* SubjectName – название предмета, тип: string.

1. **Faculty** – сущность, представляющая факультет.

* idFaculty – уникальный идентификатор факультета, тип: int;
* FacultyName – название факультета, тип: string.

1. **Group** – сущность, представляющая группу.

* GroupNum – уникальный идентификатор группы (номер группы), тип: int;
* Faculty – факультет, которому принадлежит группа, тип: string;
* Specialty – специальность, которой принадлежит группа, тип: string.

1. **Specialty**

* idSpecialty – уникальный идентификатор специальности, тип: int;
* SpecialtyName – название специальности, тип: string;
* FacultyName – факультет, которому принадлежит специальность, тип: string.

Создана таблица **Marks** для хранения оценок учащихся. В ней вводятся дополнительные поля idMark (int) и Semester (int).

*Для упрощения счёта все коллекции представлены в единичном экземпляре.*

Int = 4 Б, string = 50 Б, DBRef = 10 Б.

* Student {ExamBookNum: int, Name: string, Surname: string, FathersName: string, Group: int + DBRef}: 2 × 4 + 3 × 50 +10 = 168 Б.
* Marks {ExamBookNum: int, idMark: int, Semester: string, Subject: string}: 2 × 4 + 2 × 50 = 108 Б.
* Subject {idSubject: int, SubjectName: string}: 4 + 50 = 54 Б.
* Group {GroupNum: int, faculty: string, Specialty: string}: 4 + 2 × 50 + 2 × 10 = 104 Б.
* Faculty {idFaculty: int, FacultyName: string}: 4 + 50 = 54 Б.
* Specialty {idSpecialty: int, SpecialtyName: string}: 4 + 50 = 54 Б.

Возьмём количество студентов, равное S:

*«Чистый» объём*: (158 + 108) × S + 54 + 104 + 54 + 54 = 266S + 266 = 266(S + 1) Б.

*Фактический объём*: (168 + 108) × S + 54 + 104 + 54 + 54 = 266S + 266 = 276(S + 1) Б.

*Избыточность модели*: 276(S + 1)/266(S + 1), что гораздо меньше, чем у NoSQL-модели.

# Примеры запросов

* Для добавления студента:

INSERT INTO student (id, name, surname, group,facult) VALUES (1, "Иван", "Иванович","Иванов","1111","FKTI")

* Для поиска студента:

SELECT \* FROM student WHERE field\_name = 'Петр' AND field\_surname = 'Петрович' AND field\_ patronymic AND field\_group AND etc....

* Для статистик:

SELECT OBJECT\_NAME(s.object\_id) AS object\_name, COL\_NAME(sc.object\_id, sc.column\_id) AS column\_name, s.name AS statistics\_name FROM sys.stats AS s INNER JOIN sys.stats\_columns AS sc ON s.stats\_id = sc.stats\_id AND s.object\_id = sc.object\_id WHERE s.name like '\_WA%' ORDER BY s.name;

# Сравнение NoSQL и SQL моделей данных

### *Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров*

Количество запросов в реляционной модели меньше, чем в нереляционной, что связано с большим количеством коллекций и необходимостью добавлять или получать данные из каждой коллекции по отдельному запросу. Однако, запросы в нереляционной модели намного проще, чем в реляционной, т.к. представляют цепочку атомарных операций.

### *Количество задействованных коллекций (если есть)*

В данном случае в нереляционной модели потребуется намного больше коллекций (в 2 раза), чем в реляционной, что связано со спецификой работы и реализации предлагаемого программного обеспечения (memcached): может использоваться только пара ключ-значение, что и является сущностью/коллекцией/таблицей. В связи с этим, требуется создание множества коллекций.

* + Количество задействованных коллекций. В реляционной модели потребуется на одну таблицу (коллекцию) больше, так как в нереляционной модели каждая оценка студента хранится вместе со студентом, а в реляционной модели это невозможно, поэтому создается отдельная таблица для оценок.

Объём нереляционной базы данных в два раза превышает объём реляционной базы данных. Для данного задания (Электронный журнал успеваемости) рациональнее было бы использовать реляционную модель в связи с удобством реализации и возможностью написания сложных запросов, однако преимуществом данной нереляционной базы данных является скорость доступа к данным, равная О(1).

# РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

# Краткое описание

Приложение написано на python c использованием библиотеки для математических вычислений и с использованием memcached в качестве БД.

Приложение представляет собой

# Схема экранов приложения

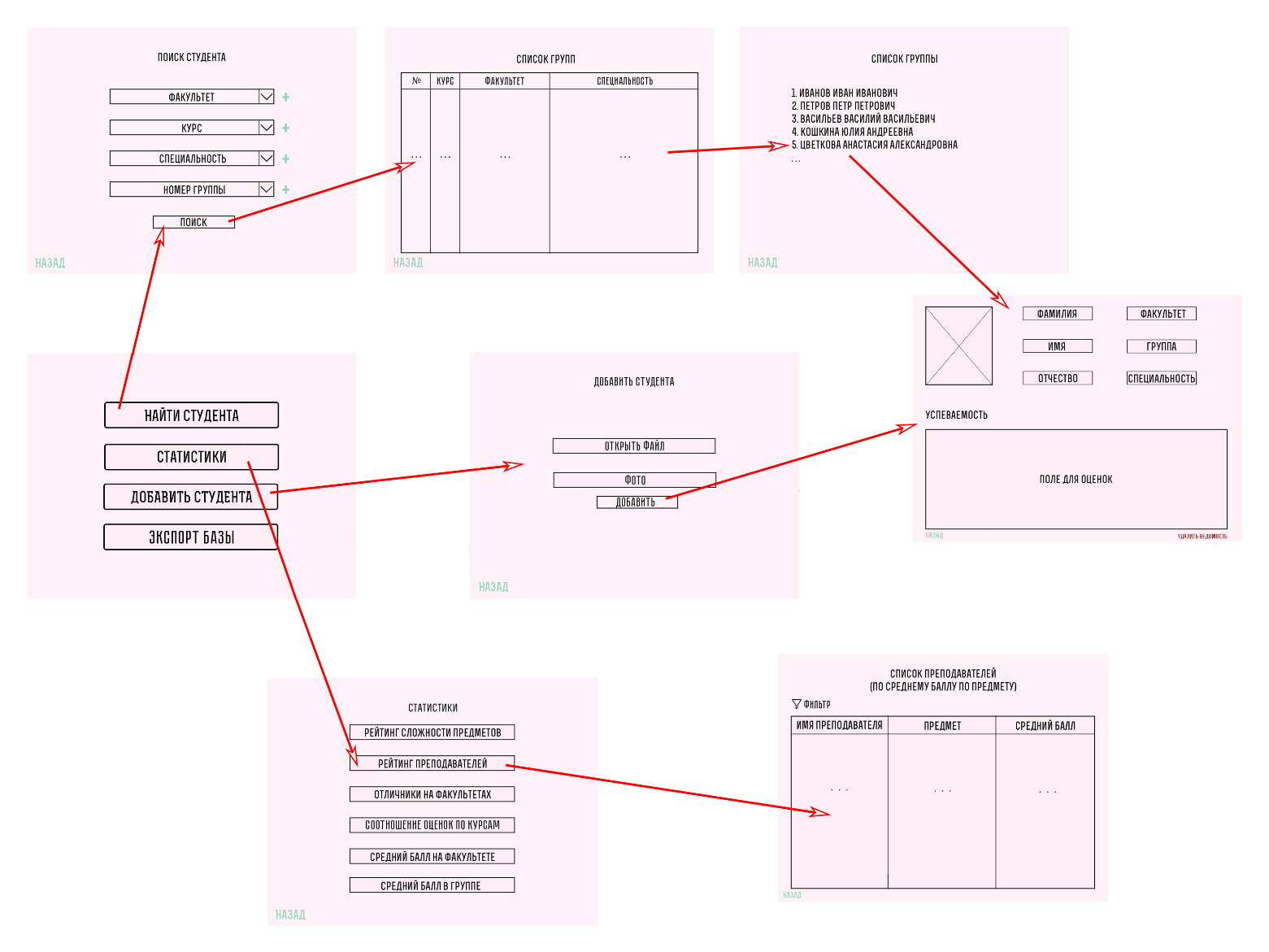
****

Рисунок 11 — Схема экранов приложения

# Использованные технологии

БД: Memcached [1, 2].

Frontend/Backend: Python.

# Ссылка на приложение

Ссылка на приложение находится в Приложении А.

# ЗАКЛЮЧЕНИЕ

В ходе работы была достигнута цель: разработано приложение для хранения информации о студентах и их успеваемости. Реализованы функции добавления и удаления информации. Предусмотрена возможность экспорта данных. В приложении можно просмотреть интересующую статистику по студентам, предметам или преподавателям.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Memcached – a distributed memory object caching system [Электронный ресурс]. URL: http://memcached.org/.
2. Python + Memcached: Эффективное кэширование в распределенных приложениях [Электронный ресурс]. URL: https://python-scripts.com/memcached

# ПРИЛОЖЕНИЕ А

# ССЫЛКА НА ПРИЛОЖЕНИЕ

<https://github.com/moevm/nosql2h19-decision/tree/master/DataBaseJournal>

# ПРИЛОЖЕНИЕ Б

# ИНСТРУКЦИЯ ПО СБОРКЕ И ЗАПУСКУ

1. Установка memcached.
2. Установка python.
3. Установка PYcharm.
4. Запуск серверной части командой memcached.exe -vv (в директории с файлом memcached.exe).
5. Заходите в Панель управления - Программы и компоненты
6. Заходите слева в Включение и отключение компонентов в Windows - и ставите галочку рядом с Telnet Client или Клиент Telnet
7. Не закрывая командную строку после шага 4, открываете новую командную строку и пишите: "telnet localhost 11211" Нажмите Enter