Национальный исследовательский университет

«Высшая школа экономики»

Факультет компьютерных наук

Прикладная математика и информатика

Проект по дисциплине

“Теория баз данных”

Выполнили:

Александров Никита

Гавриков Алексей

Лотфуллин Камиль

Проверяющий:

Незнанов Алексей



**Содержание**

Техническое задание

Концептуальное проектирование:

* Краткое описание предметной области
* Описание процесса построения инфологической модели с обоснованием выделения сущностей и связей
* ER ­диаграмма

Проектирование реляционной модели:

* Описание процесса перехода к реляционной модели
* Диаграмма реляционной модели/Диаграмма схемы БД
* Дополнительные механизмы обеспечения целостности данных

Развёртывание БД в выбранной СУБД:

* DDL ­скрипт создания схемы БД.
* Примеры DML­ операторов вставки тестовых данных.
* Контроль работы ограничений целостности с примерами.

Разработка клиентского приложения

Заключение

**Техническое задание**

GradeHim - система корпоративного оценивания, развернутая на примере университета.

**Проблема, которую решает проект:** отсутствие постоянного объективного feedback-а между преподавателями и студентами, студентом и студентом.

**Что мы собираемся делать?** Веб приложение, которое позволяет студентам оценивать преподавателя или друг друг после лекций, семинаров и прочих собраний. С помощью приложения пользователь на некоторой встрече сможет выделить и оценить отдельные качества другого участника, которому он хочет дать отзыв. В любой момент времени каждый участник собрания может видеть какие оценки ставят другие участники, это будет обеспечивать объективность feedback-а, потому что, видя отличные от своего мнения оценки, пользователь будет сомневаться в правильности своей оценки. Постоянство feedback-а будет обеспечиваться за счет использование данной системы на каждой встрече.

**Краткое описание предметной области**

Предметная область - это преподаватели и студенты, занятия, на которых они преподают или посещают. После каждой их встречи они оценивают другу друга, выделяют проявленные качества которые им понравились или не понравились. Наша система оценивания обеспечивает объективность оценок. Пользователи системы могут смотреть, как их оценили, какие их сильные стороны, а какие слабые.

**Описание процесса построения инфологической модели с обоснованием выделения сущностей и связей**

В модели было выделено 6 сущностей. Далее будет указан список сущностей с атрибутами.

1. Юзер
   * ФИО
   * Фотография
   * id
2. Студент
   * User.id
3. Преподаватель
   * User.id
4. Дисциплина
   * Id
   * Дата начала
   * Дата окончания
5. Собрание
   * Название
   * Место
   * Время
   * id
6. Характеристика
   * Название
   * Значение
7. Тип
   * Название
   * Студенты могут оценивать преподавателя
   * Студенты могут оценивать студентов
   * Преподаватель может оценивать студентов

Теперь опишем процесс построения связей между сущностями.

“Является” - связывает юзеров со своими классами. Юзер может быть преподавателем или студентом.

“Изучает” - связывает от 0 до N студентов с изучаемыми ими от 0 до N дисциплин. То есть 1 студент может изучать от 0 до N дисциплин и 1 дисциплину могут изучать от 0 до N студентов.

“Преподает” - связывает 1 до N преподавателей с 0 до N дисциплин и дисциплину с 1 до N преподавателями. То есть у дисциплины должен быть хотя бы один преподаватель и преподаватель может не преподавать ни одной дисциплины или много.

“Включает” - связывает 0 до 1 дисциплины с 0 до N встречами. То есть встреча принадлежит не более чем одной дисциплине, а дисциплина может иметь от 0 до N встреч.

“Ведет” - связывает от 0 до N преподавателей с 0 до N собраниями. То есть собрание может иметь от 0 до N преподавателей. Например встреча студентов по научному вопросу - не имеет преподавателей, а какой-нибудь мастер класс может иметь 2 преподавателей.

“Является” - связывает 0 до N собраний с 1 типом. То есть каждое собрание имеет обязательно 1 тип, но тии может не иметь собраний или иметь их много.

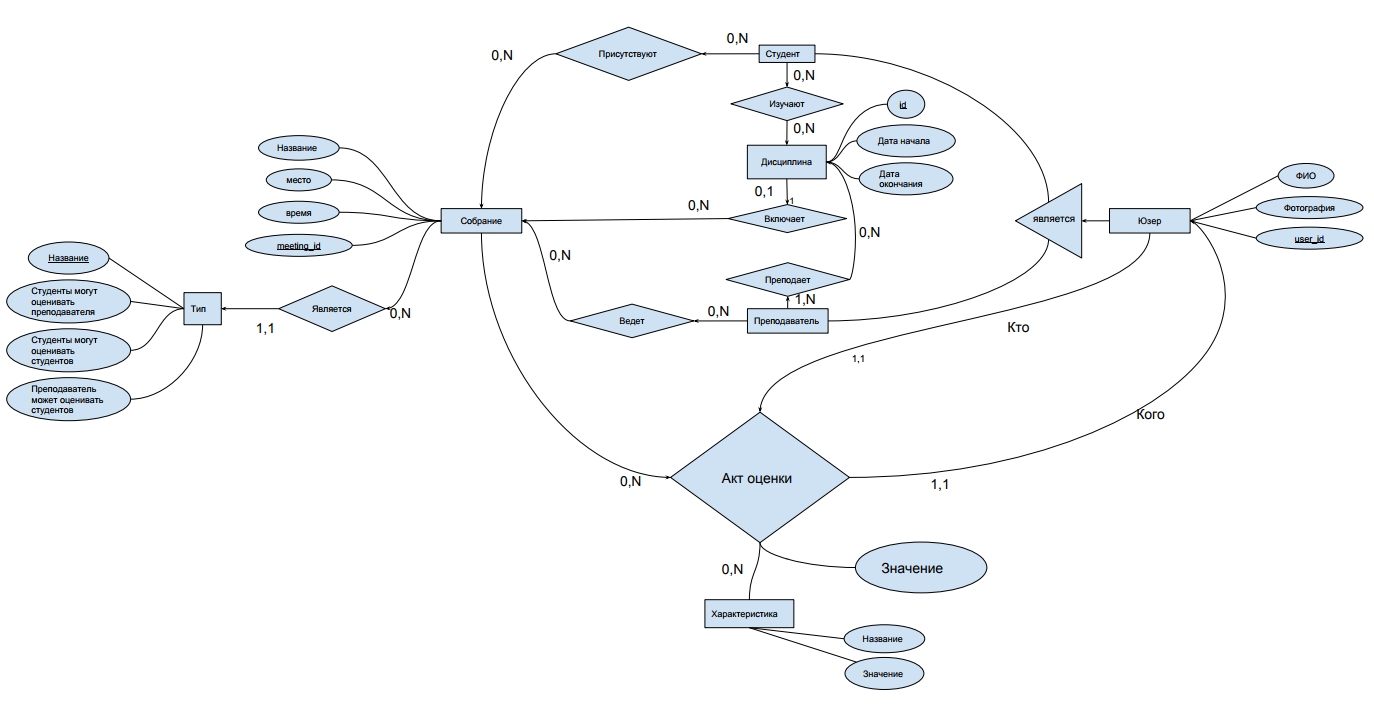
“Акт оценки”

* Значение - общее значение оценкию
* Характеристики

Данная сущность связывает 0 до N выделенных характеристик с 0 до N собраниями, оцениваемым и оценивающим юзерами. То есть у каждой встречи может быть много оценок, на каждой встречи каждого могли оценить много юзер и поставить много оценок.

**ER диаграмма**

[Ссылка на диаграмму в хорошем качестве](https://www.dropbox.com/s/vaw33hqjivbl24z/ER%20diagram.pdf?dl=0)



**Проектирование реляционной модели**

На основе ER-диаграммы строим реляционную модель.

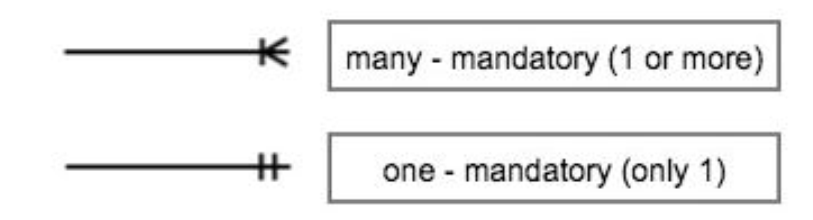
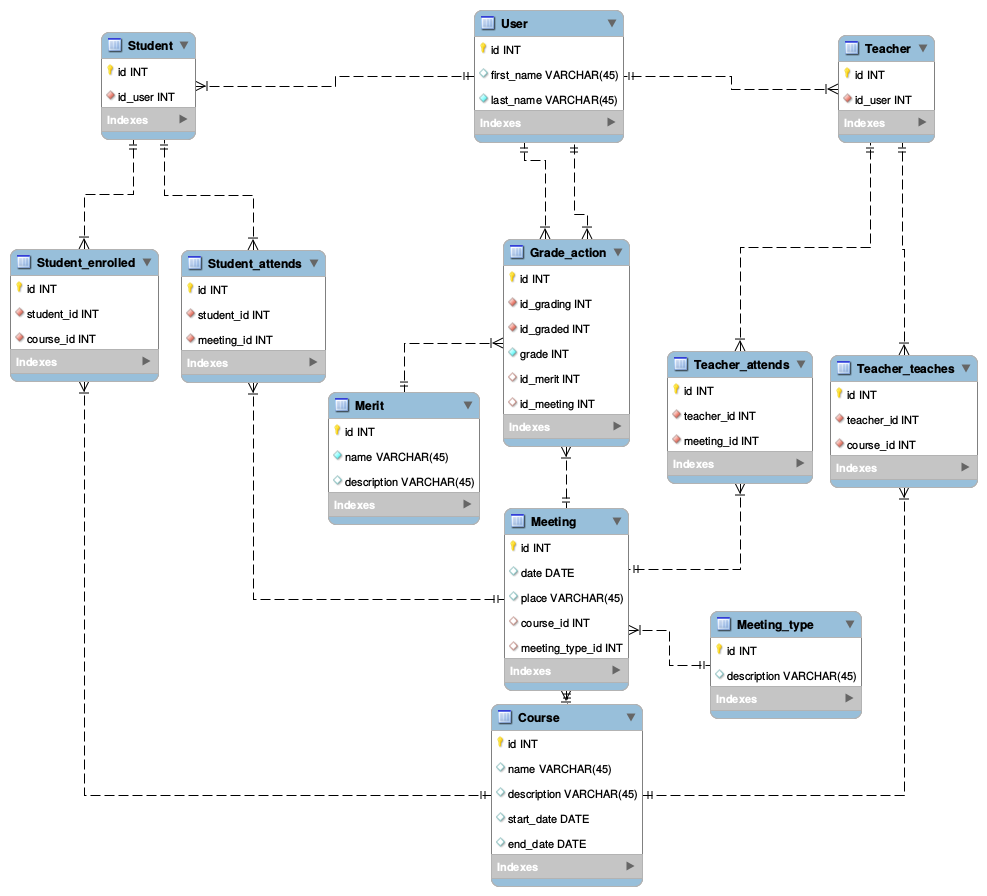
Сущности:

* User
* Student
* Teacher
* Merit
* Course
* Meeting
* Meeting\_type

Связи:

* «Один к одному»
  + User — Teacher
  + User — Student
* «Один ко многим»
  + Meeting - Meeting\_type
* «Многие ко многим»
  + Student\_enrolled: (Student, Course)
  + Student\_attends: (Student, Meeting)
  + Teacher\_teaches: (Teacher, Course)
  + Teacher\_attends: (Teacher, Meeting)
  + Grade\_action: (User, User, meeting, merit)

**Диаграмма реляционной модели / Диаграмма схемы БД**



**Обеспечение целостности данных**

Важные поля объектов класса сделаны со свойством not\_null.

Специфицированы действия при удалении объекта из базы, на который ссылаются другие сущности. Ниже декларация проверок этих ограничений при создании ORM моделей.

**class** Student(models.Model):  
 user = models.OneToOneField(User, on\_delete=models.CASCADE, not\_null=True)

**class** Teacher(models.Model):  
 user = models.OneToOneField(User, on\_delete=models.CASCADE, not\_null=True)

**class** Merit(models.Model):  
 name = models.CharField(not\_null=True)

**class** MeetingType(models.Model):  
 description = models.CharField(not\_null=True)

**class** Meeting(models.Model):  
 course = models.ForeignKey(Course, on\_delete=models.SET\_NULL, not\_null=False)  
 meeting\_type = models.ForeignKey(MeetingType, on\_delete=models.SET\_NULL, not\_null=False)

**class** TeacherTeaches(models.Model):  
 teacher = models.ForeignKey(Teacher, on\_delete=models.CASCADE, not\_null=True)  
 course = models.ForeignKey(Course, on\_delete=models.CASCADE, not\_null=True)

**class** StudentEnrolled(models.Model):  
 student = models.ForeignKey(Student, on\_delete=models.CASCADE, not\_null=True)  
 course = models.ForeignKey(Course, on\_delete=models.CASCADE, not\_null=True)+

**class** TeacherAttends(models.Model):  
 teacher = models.ForeignKey(Teacher, on\_delete=models.CASCADE, not\_null=True)  
 meeting = models.ForeignKey(Meeting, on\_delete=models.CASCADE, not\_null=True)

**class** StudentAttends(models.Model):  
 student = models.ForeignKey(Student, on\_delete=models.CASCADE, not\_null=True)  
 meeting = models.ForeignKey(Meeting, on\_delete=models.CASCADE, not\_null=True)

**class** GradeAction(models.Model):  
 grading = models.ForeignKey(User, on\_delete=models.CASCADE, not\_null=True)  
 graded = models.ForeignKey(User, on\_delete=models.CASCADE, not\_null=True)  
 merit = models.ForeignKey(Merit, on\_delete=models.SET\_NULL, not\_null=False)  
 meeting = models.ForeignKey(Meeting, on\_delete=models.SET\_NULL, not\_null=False)

**Развёртывание БД в выбранной СУБД**

В качестве реляционной базы данных была выбрана реляционная СУБД SQLite.

Данные сгенерированы и загружены с помощью скриптов, написанных на Python 3.

В качестве примера приведены python-классы, по которым Джанго генерирует таблицы сущностей и связей и класс DataManager, который генерирует и загружает в БД тестовые данные.

С помощью приведенных ниже строк, достаточно в папке с проектом написать несколько коротких строк и база данных будет подготовлена с нуля.

python manage.py migrate

python manage.py shell

import app.models as m  
m.DataManager.generate\_test\_data()

**from** django.db **import** models  
**from** django.contrib.auth.models **import** User  
**from** datetime **import** datetime, timedelta  
**from** dateutil.relativedelta **import** \*  
**import** random  
  
**def** save\_if(model, created):  
 **if** created:  
 print(**"created"**, model)  
 model.save()  
 **else**:  
 print(**"existing"**, model)  
  
models.Model.save\_if = save\_if  
  
**class** Student(models.Model):  
 user = models.OneToOneField(User, on\_delete=models.CASCADE)  
  
 **def** \_\_str\_\_(self):  
 **return** self.user.\_\_str\_\_()  
  
 @staticmethod  
 **def** create(user):  
 student, created = Student.objects.get\_or\_create(  
 user=user  
 )  
 save\_if(student, created)  
  
  
**class** Teacher(models.Model):  
 user = models.OneToOneField(User, on\_delete=models.CASCADE)  
  
 **def** \_\_str\_\_(self):  
 **return** self.user.\_\_str\_\_()  
  
 @staticmethod  
 **def** create(user):  
 teacher, created = Teacher.objects.get\_or\_create(  
 user=user  
 )  
 save\_if(teacher, created)  
  
  
**class** Course(models.Model):  
 name = models.CharField(max\_length=50)  
 description = models.CharField(max\_length=50, blank=**True**, null=**True**)  
 start\_date = models.DateField(blank=**True**, null=**True**)  
 end\_date = models.DateField(blank=**True**, null=**True**)  
  
 **def** \_\_str\_\_(self):  
 **return** self.name  
  
 @staticmethod  
 **def** create(name, description, start\_date, end\_date):  
 course, created = Course.objects.get\_or\_create(  
 name=name,  
 description=description,  
 start\_date=start\_date,  
 end\_date=end\_date,  
 )  
 save\_if(course, created)  
  
  
**class** Merit(models.Model):  
 name = models.CharField(max\_length=45)  
 description = models.CharField(max\_length=50, blank=**True**, null=**True**)  
  
 **def** \_\_str\_\_(self):  
 **return** self.name  
  
 @staticmethod  
 **def** create(name, description):  
 merit, created = Merit.objects.get\_or\_create(  
 name=name,  
 description=description  
 )  
 save\_if(merit, created)  
  
  
**class** MeetingType(models.Model):  
 description = models.CharField(max\_length=50)  
  
 **def** \_\_str\_\_(self):  
 **return** self.description  
  
 @staticmethod  
 **def** create(description):  
 meeting\_type, created = MeetingType.objects.get\_or\_create(  
 description=description  
 )  
 save\_if(meeting\_type, created)  
  
  
**class** Meeting(models.Model):  
 date = models.DateField()  
 place = models.CharField(max\_length=50, blank=**True**, null=**True**)  
 course = models.ForeignKey(Course, on\_delete=models.SET\_NULL, blank=**True**, null=**True**)  
 meeting\_type = models.ForeignKey(MeetingType, on\_delete=models.SET\_NULL, blank=**True**, null=**True**)  
  
 **def** \_\_str\_\_(self):  
 **return 'course {}; meeting\_type {}; date {}/{}/{}; place {}'**.format(  
 self.course, self.meeting\_type, self.date.day, self.date.month, self.date.year, self.place)  
  
 @staticmethod  
 **def** create(date, place, course, meeting\_type):  
 meeting, created = Meeting.objects.get\_or\_create(  
 date=date,  
 place=place,  
 course=course,  
 meeting\_type=meeting\_type  
 )  
 save\_if(meeting, created)  
  
  
**class** TeacherTeaches(models.Model):  
 teacher = models.ForeignKey(Teacher, on\_delete=models.CASCADE)  
 course = models.ForeignKey(Course, on\_delete=models.CASCADE)  
  
 **def** \_\_str\_\_(self):  
 **return '{} - {}'**.format(self.teacher, self.course)  
  
 @staticmethod  
 **def** create(course, teacher):  
 teacherTeaches, created = TeacherTeaches.objects.get\_or\_create(  
 course = course,  
 teacher = teacher  
 )  
 save\_if(teacherTeaches, created)  
  
  
**class** StudentEnrolled(models.Model):  
 student = models.ForeignKey(Student, on\_delete=models.CASCADE)  
 course = models.ForeignKey(Course, on\_delete=models.CASCADE)  
  
 **def** \_\_str\_\_(self):  
 **return '{} - {}'**.format(self.student, self.course)  
  
 @staticmethod  
 **def** create(course, student):  
 studentEnrolled, created = StudentEnrolled.objects.get\_or\_create(  
 course = course,  
 student = student  
 )  
 save\_if(studentEnrolled, created)  
  
  
**class** TeacherAttends(models.Model):  
 teacher = models.ForeignKey(Teacher, on\_delete=models.CASCADE)  
 meeting = models.ForeignKey(Meeting, on\_delete=models.CASCADE)  
  
 **def** \_\_str\_\_(self):  
 **return '{} - {}'**.format(self.teacher, self.meeting)  
  
 @staticmethod  
 **def** create(meeting, teacher):  
 teacherAttends, created = TeacherAttends.objects.get\_or\_create(  
 meeting = meeting,  
 teacher = teacher  
 )  
 save\_if(teacherAttends, created)  
  
  
**class** StudentAttends(models.Model):  
 student = models.ForeignKey(Student, on\_delete=models.CASCADE)  
 meeting = models.ForeignKey(Meeting, on\_delete=models.CASCADE)  
  
 **def** \_\_str\_\_(self):  
 **return '{} - {}'**.format(self.student, self.meeting)  
  
 @staticmethod  
 **def** create(meeting, student):  
 studentAttends, created = StudentAttends.objects.get\_or\_create(  
 meeting = meeting,  
 student = student  
 )  
 save\_if(studentAttends, created)  
  
  
**class** GradeAction(models.Model):  
 grading = models.ForeignKey(User, on\_delete=models.CASCADE, related\_name=**'who\_grading'**)  
 graded = models.ForeignKey(User, on\_delete=models.CASCADE, related\_name=**'who\_graded'**)  
 grade = models.IntegerField()  
 merit = models.ForeignKey(Merit, on\_delete=models.SET\_NULL, blank=**True**, null=**True**)  
 meeting = models.ForeignKey(Meeting, on\_delete=models.SET\_NULL, blank=**True**, null=**True**)  
  
 **def** \_\_str\_\_(self):  
 **return 'grading {}; graded {}; grade {}; merit {}; meeting {}'**.format(  
 self.grading, self.graded, self.grade, self.merit, self.meeting)

**class** DataManager(models.Manager):  
  
 @staticmethod  
 **def** generate\_test\_data():  
 DataManager.print\_all\_users\_roles()  
 DataManager.generate\_students\_teachers(num\_of\_users=35)  
 DataManager.generate\_courses()  
 DataManager.generate\_studentEnrolled\_teacherTeaches()  
 DataManager.generate\_meeting\_types()  
 DataManager.generate\_merits()  
 DataManager.generate\_meetings(weeks\_number=5)  
 DataManager.generate\_students\_teachers\_attends()  
 DataManager.generate\_grade\_actions()

Далее мы можем видеть, как генерируется таблица внутри Джанго

--

-- Create model Course

--

CREATE TABLE "app\_course" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "name" varchar(50) NOT NULL, "description" varchar(50) NULL, "start\_date" date NULL, "end\_date" date NULL);

--

-- Create model GradeAction

--

CREATE TABLE "app\_gradeaction" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "grade" integer NOT NULL, "graded\_id" integer NOT NULL REFERENCES "auth\_user" ("id"), "grading\_id" integer NOT NULL REFERENCES "auth\_user" ("id"));

--

-- Create model Meeting

--

CREATE TABLE "app\_meeting" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "date" date NOT NULL, "place" varchar(50) NULL, "course\_id" integer NULL REFERENCES "app\_course" ("id"));

--

-- Create model MeetingType

--

CREATE TABLE "app\_meetingtype" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "description" varchar(50) NOT NULL);

--

-- Create model Merit

--

CREATE TABLE "app\_merit" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "name" varchar(45) NOT NULL, "description" varchar(50) NULL);

--

-- Create model Student

--

CREATE TABLE "app\_student" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "user\_id" integer NOT NULL UNIQUE REFERENCES "auth\_user" ("id"));

--

-- Create model StudentAttends

--

CREATE TABLE "app\_studentattends" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "meeting\_id" integer NOT NULL REFERENCES "app\_meeting" ("id"), "student\_id" integer NOT NULL REFERENCES "app\_student" ("id"));

--

-- Create model StudentEnrolled

--

CREATE TABLE "app\_studentenrolled" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "course\_id" integer NOT NULL REFERENCES "app\_course" ("id"), "student\_id" integer NOT NULL REFERENCES "app\_student" ("id"));

--

-- Create model Teacher

--

CREATE TABLE "app\_teacher" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "user\_id" integer NOT NULL UNIQUE REFERENCES "auth\_user" ("id"));

--

-- Create model TeacherAttends

--

CREATE TABLE "app\_teacherattends" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "meeting\_id" integer NOT NULL REFERENCES "app\_meeting" ("id"), "teacher\_id" integer NOT NULL REFERENCES "app\_teacher" ("id"));

--

-- Create model TeacherTeaches

--

CREATE TABLE "app\_teacherteaches" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "course\_id" integer NOT NULL REFERENCES "app\_course" ("id"), "teacher\_id" integer NOT NULL REFERENCES "app\_teacher" ("id"));

--

-- Add field meeting\_type to meeting

--

ALTER TABLE "app\_meeting" RENAME TO "app\_meeting\_\_old";

CREATE TABLE "app\_meeting" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "date" date NOT NULL, "place" varchar(50) NULL, "course\_id" integer NULL REFERENCES "app\_course" ("id"), "meeting\_type\_id" integer NULL REFERENCES "app\_meetingtype" ("id"));

INSERT INTO "app\_meeting" ("id", "date", "place", "course\_id", "meeting\_type\_id") SELECT "id", "date", "place", "course\_id", NULL FROM "app\_meeting\_\_old";

DROP TABLE "app\_meeting\_\_old";

CREATE INDEX "app\_gradeaction\_graded\_id\_35798cf7" ON "app\_gradeaction" ("graded\_id");

CREATE INDEX "app\_gradeaction\_grading\_id\_8b64c3d2" ON "app\_gradeaction" ("grading\_id");

CREATE INDEX "app\_studentattends\_meeting\_id\_f17c8fe8" ON "app\_studentattends" ("meeting\_id");

CREATE INDEX "app\_studentattends\_student\_id\_646cf24b" ON "app\_studentattends" ("student\_id");

CREATE INDEX "app\_studentenrolled\_course\_id\_758c9269" ON "app\_studentenrolled" ("course\_id");

CREATE INDEX "app\_studentenrolled\_student\_id\_43f4d5bc" ON "app\_studentenrolled" ("student\_id");

CREATE INDEX "app\_teacherattends\_meeting\_id\_1e1796d9" ON "app\_teacherattends" ("meeting\_id");

CREATE INDEX "app\_teacherattends\_teacher\_id\_1a8eebd3" ON "app\_teacherattends" ("teacher\_id");

CREATE INDEX "app\_teacherteaches\_course\_id\_e05785fa" ON "app\_teacherteaches" ("course\_id");

CREATE INDEX "app\_teacherteaches\_teacher\_id\_dcc73bb6" ON "app\_teacherteaches" ("teacher\_id");

CREATE INDEX "app\_meeting\_course\_id\_01887828" ON "app\_meeting" ("course\_id");

CREATE INDEX "app\_meeting\_meeting\_type\_id\_e42ab290" ON "app\_meeting" ("meeting\_type\_id");

--

-- Add field meeting to gradeaction

--

ALTER TABLE "app\_gradeaction" RENAME TO "app\_gradeaction\_\_old";

CREATE TABLE "app\_gradeaction" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "grade" integer NOT NULL, "graded\_id" integer NOT NULL REFERENCES "auth\_user" ("id"), "grading\_id" integer NOT NULL REFERENCES "auth\_user" ("id"), "meeting\_id" integer NULL REFERENCES "app\_meeting" ("id"));

INSERT INTO "app\_gradeaction" ("id", "grade", "graded\_id", "grading\_id", "meeting\_id") SELECT "id", "grade", "graded\_id", "grading\_id", NULL FROM "app\_gradeaction\_\_old";

DROP TABLE "app\_gradeaction\_\_old";

CREATE INDEX "app\_gradeaction\_graded\_id\_35798cf7" ON "app\_gradeaction" ("graded\_id");

CREATE INDEX "app\_gradeaction\_grading\_id\_8b64c3d2" ON "app\_gradeaction" ("grading\_id");

CREATE INDEX "app\_gradeaction\_meeting\_id\_a1871769" ON "app\_gradeaction" ("meeting\_id");

--

-- Add field merit to gradeaction

--

ALTER TABLE "app\_gradeaction" RENAME TO "app\_gradeaction\_\_old";

CREATE TABLE "app\_gradeaction" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "grade" integer NOT NULL, "graded\_id" integer NOT NULL REFERENCES "auth\_user" ("id"), "grading\_id" integer NOT NULL REFERENCES "auth\_user" ("id"), "meeting\_id" integer NULL REFERENCES "app\_meeting" ("id"), "merit\_id" integer NULL REFERENCES "app\_merit" ("id"));

INSERT INTO "app\_gradeaction" ("id", "grade", "graded\_id", "grading\_id", "meeting\_id", "merit\_id") SELECT "id", "grade", "graded\_id", "grading\_id", "meeting\_id", NULL FROM "app\_gradeaction\_\_old";

DROP TABLE "app\_gradeaction\_\_old";

CREATE INDEX "app\_gradeaction\_graded\_id\_35798cf7" ON "app\_gradeaction" ("graded\_id");

CREATE INDEX "app\_gradeaction\_grading\_id\_8b64c3d2" ON "app\_gradeaction" ("grading\_id");

CREATE INDEX "app\_gradeaction\_meeting\_id\_a1871769" ON "app\_gradeaction" ("meeting\_id");

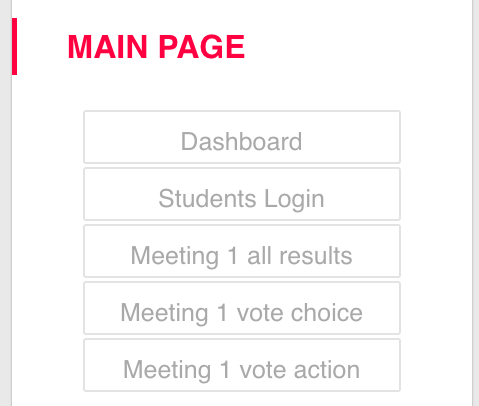
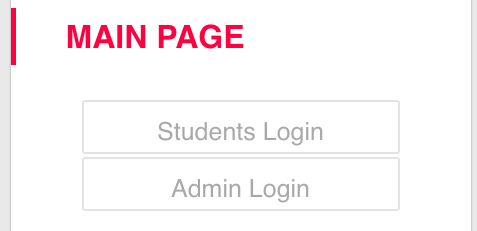
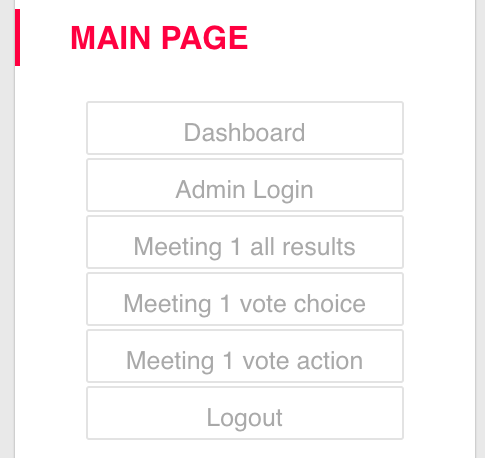
CREATE INDEX "app\_gradeaction\_merit\_id\_3113dc4a" ON "app\_gradeaction" ("merit\_id");

COMMIT;

**Примеры использования и вывода данных в веб приложении**

1. Для главной страницы (Main Page) выводятся определенные ссылки, в зависимости от аутентификации пользователя:

**def** main\_page(request):  
 **if** request.user.is\_authenticated():  
 is\_student = Student.objects.filter(user=request.user)  
 **if** is\_student:  
 title = **"Hello, User {}!"**.format(is\_student.first().user)  
 links\_list = Link.objects.all().exclude(name=**"Students Login"**).exclude(name=**"Main"**)  
 **else**:  
 is\_teacher = Teacher.objects.filter(user=request.user)  
 **if** is\_teacher:  
 title = **"Hello, Teacher {}!"**.format(is\_teacher.first().user)  
 links\_list = Link.objects.all().exclude(name=**"Students Login"**).exclude(name=**"Main"**)  
 **else**:  
 title = **"Hello, Admin {}!"**.format(request.user)  
 links\_list = Link.objects.all().exclude(name=**"Admin Login"**).exclude(name=**"Logout"**).exclude(name=**"Main"**)  
 **else**:  
 print(**"NOT AUTHENTICATED"**)  
 title = **"Hello, Guest!"**.format(request.user)  
 links\_list = Link.objects.all().exclude(name=**"Dashboard"**).exclude(name=**"Dashboard"**).exclude(name=**"Dashboard"**).exclude(name=**"Meeting 1 all results"**).exclude(name=**"Meeting 1 vote choice"**).exclude(name=**"Meeting 1 vote action"**).exclude(name=**"Logout"**).exclude(name=**"Main"**)  
 context = {  
 **"title"**: title,  
 **"links\_list"**: links\_list  
 }  
 **return** render(request, **'app/main.html'**, context)

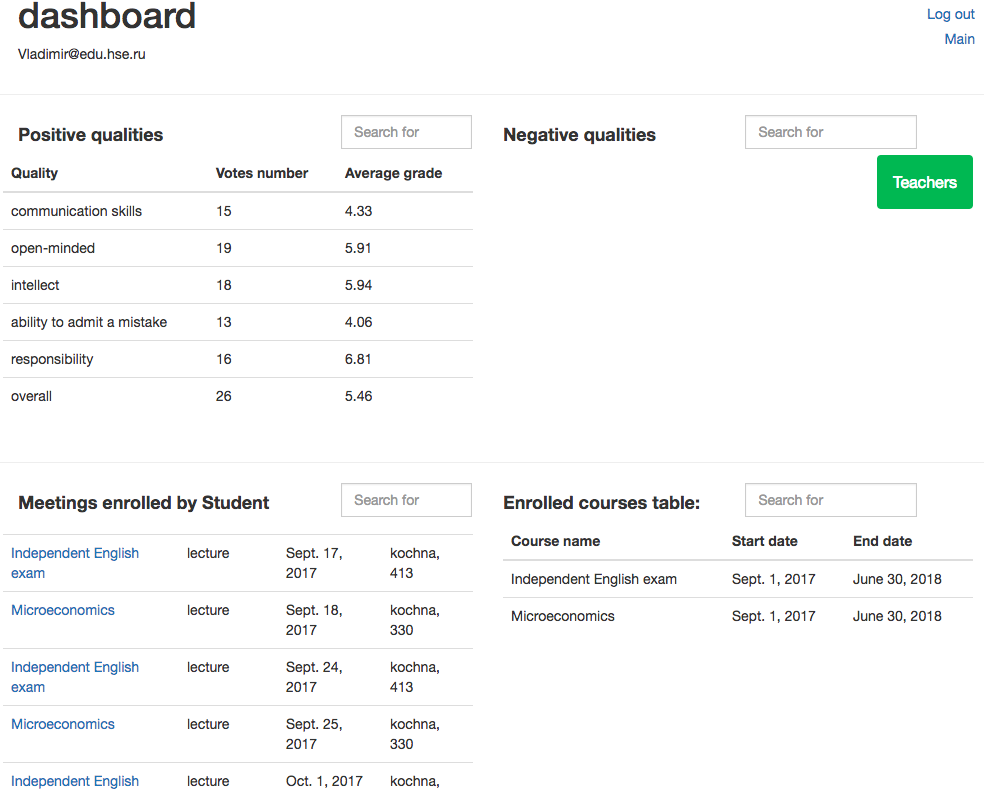


Для студентов Для нового юзера Для админа

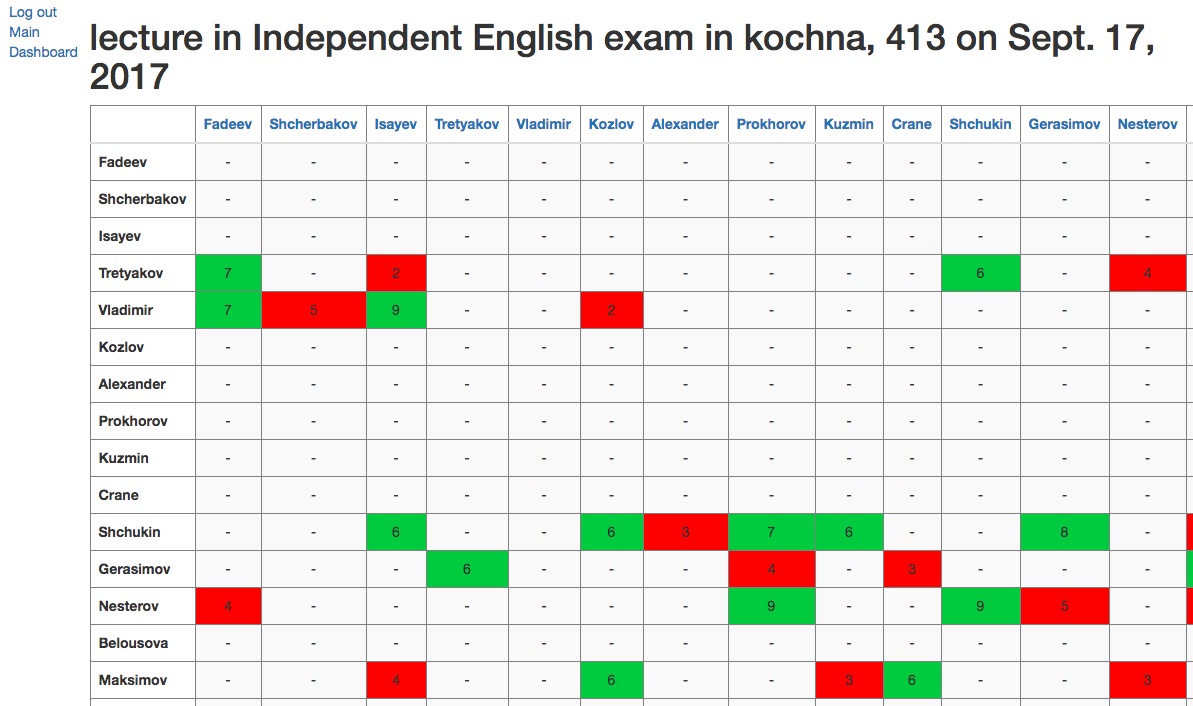
2. Для Dashboard выводятся основные сущности для конкретного юзера: То, как его оценивают другие пользователи по всем характеристикам(название, количество оценок, средняя оценка);

Курсы, на которые студент записан(название, дата начала, дата окончания), и занятия, которые студент посещает(название, тип, дата, место).

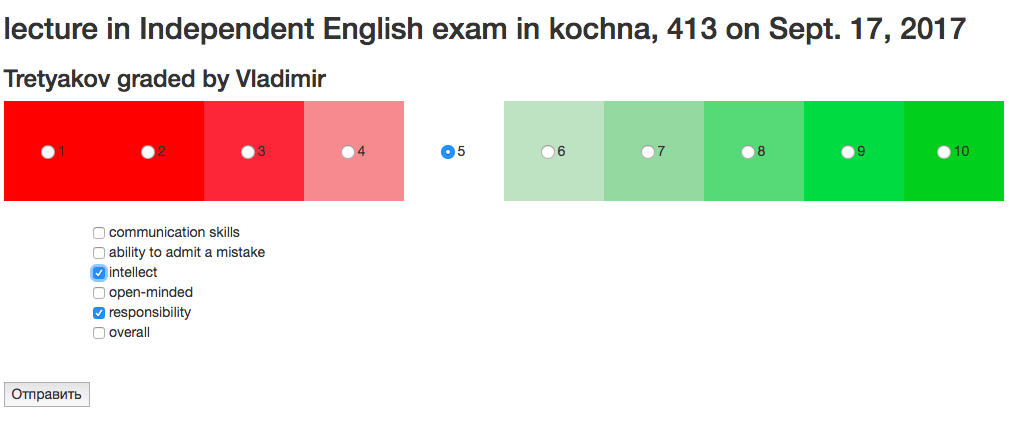
**def** dashboard(request):  
 current\_user = request.user  
 context = {**'courses\_enrolled'**:[],  
 **'first\_name'**: current\_user.first\_name,  
 **'last\_name'**: current\_user.last\_name,  
 **'email'**: current\_user.email,  
 **'meeting\_enrolled'**:[]}  
  
 students\_enrolled = StudentEnrolled.objects.filter(student\_\_user=current\_user)  
 **for** student\_enrolled **in** students\_enrolled:  
 course\_info = {**'course\_name'**: student\_enrolled.course.name,  
 **'start\_date'**: student\_enrolled.course.start\_date,  
 **'end\_date'**: student\_enrolled.course.end\_date }  
 context.get(**'courses\_enrolled'**).append(course\_info)  
  
 students\_attended = StudentAttends.objects.filter(student\_\_user=current\_user)  
 **for** student\_attended **in** students\_attended:  
 meeting = student\_attended.meeting  
 context.get(**'meeting\_enrolled'**).append(meeting)  
  
 meetings\_list = sorted(context[**"meeting\_enrolled"**], key=**lambda** x: x.date)  
 context[**"meeting\_enrolled"**] = meetings\_list  
 print(context[**"meeting\_enrolled"**])  
  
 grade\_actions = GradeAction.objects.all().filter(graded = current\_user)  
 context[**'positive\_qualities'**] = {}  
 context[**'negative\_qualities'**] = {}  
  
 **def** incr\_average(merit, plus):  
 last\_count = merit[0]  
 average = merit[1]  
 merit[0] = last\_count + 1  
 merit[1] = round((average \* last\_count + plus) / merit[0], 2)  
  
 *# merit [count=0 ,average=0]* **for** grade **in** grade\_actions:  
 merit = grade.merit  
 **if** context[**'positive\_qualities'**].get(merit.name) == **None**:  
 context[**'positive\_qualities'**][merit.name] = [0, 0]  
 incr\_average(context[**'positive\_qualities'**][merit.name], grade.grade)  
  
 teachers\_names = [teacher.user.username **for** teacher **in** Teacher.objects.all()]  
 context[**"teachers\_names"**] = teachers\_names  
 **return** render(request, **'app/dashboard.html'**, context)  
  
  
**from** .models **import** Meeting, GradeAction, TeacherAttends, StudentAttends  
**from** django.shortcuts **import** get\_object\_or\_404



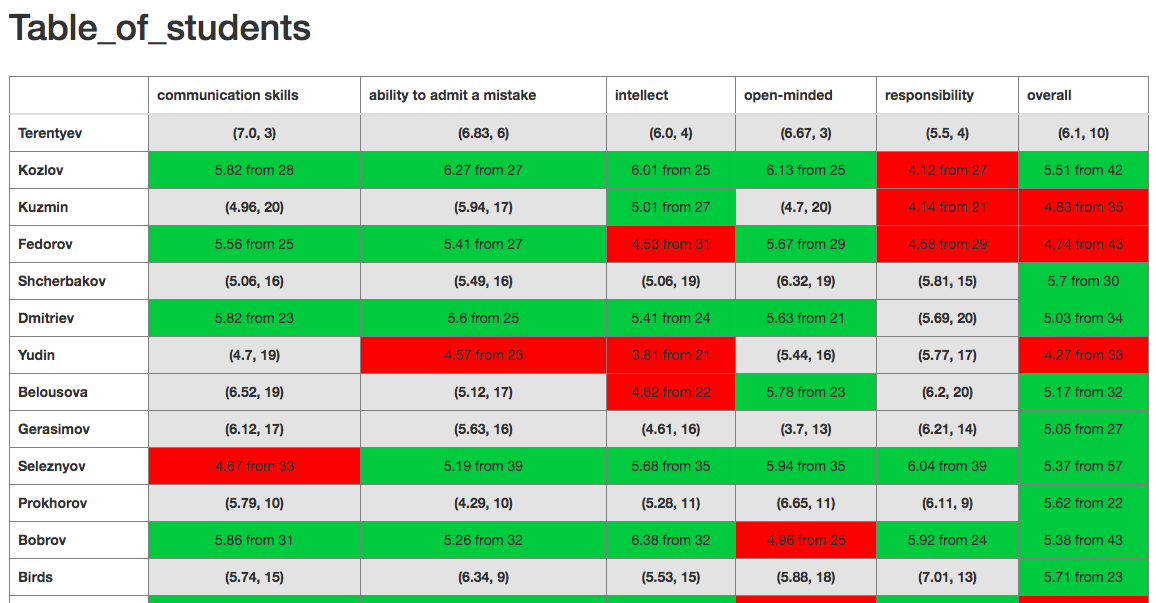
3. По нажатию на название встречи на экране Dashboard можно перейти на экран meeting/x/results, на котором можно просмотреть оценки пользователей и поставить оценку любому из пользователей.



4. По нажатию на имя пользователя можно перейти на экран оценивания meeting/id/vote\_choice/id/, где спроектирована и реализована функция выставления оценки с помощью POST методов.



5. С главного экрана можно посмотреть статистику по всем учителям или студентам, содержащую в себе оценки по основным характеристикам, визуализацию в цвете и прочее.



**Заключение**

В ходе данной работы спроектирована база данных для оценивания студентов и учителей по системе все-оценивают-всех. Она была реализована и наполнена данными, сгенерированными с помощью скриптов, написанных на Python 3. В процессе данной части работы была построена схема Базы Данных и построена диаграмма «Таблица-Связь».

Создано веб приложение, позволяющее пользователям под своими данными заходить в систему чтобы просматривать многостороннюю статистику и выставлять оценки другим юзерам. Пользователи могут просматривать основную статистику, ставить оценки за курс любому юзеру.

Что удалось: Реализовать полноценную систему с заданным функционалом. С качественной визуализацией данных и функцией оценивания. Кроме того динамический контент в зависимости от уровня доступа пользователя(админ или студент).

Что не удалось: Удобный интерфейс для доступа к функционалу в “3 клика”. На это требовалось объективно больше времени и ресурсов.