Министерство Образования и Науки Кыргызской Республики

Кыргызский Государственный Технический Университет

Им. И. Раззакова

Факультет Информационных Технологий

**Пояснительная записка**

Тема: Электронный Ассистент Академического Советника

Группа: ИВТ-1-18

Студент: Баякеев Рустам

Бишкек 2022

Оглавление

[Введение 3](#_Toc106206229)

[Глава 1: Обзор и анализ 6](#_Toc106206230)

[1.1. Анализ области темы 6](#_Toc106206231)

[1.2. Обзор существующих систем. 8](#_Toc106206232)

[1.2.1. Google Classroom 8](#_Toc106206233)

[1.2.2. Moodle 10](#_Toc106206234)

[1.2.3. WebTutor 11](#_Toc106206235)

[1.2.4. Выводы 12](#_Toc106206236)

[1.3. Функции системы 13](#_Toc106206237)

[1.3.1. Основные функции. 13](#_Toc106206238)

[1.3.2. Подсистемы и их функции 14](#_Toc106206239)

[1.4. Постановка задач 17](#_Toc106206240)

[1.4.1. Аккаунт пользователя. 17](#_Toc106206241)

[1.4.2. Основной алгоритм системы 19](#_Toc106206242)

[Глава 2: Разработка Электронного Ассистента Академического Советника 20](#_Toc106206243)

[2.1 Модификация стандартной установки 20](#_Toc106206244)

[2.2 Реализация разбития пользователей на группы. 21](#_Toc106206245)

[2.3 Реализация входа в систему. 27](#_Toc106206246)

[2.4 Реализация проверки типа пользователя при авторизации. 30](#_Toc106206247)

[2.5 Реализация дополнительных моделей пользователей и групп со студентами. 31](#_Toc106206248)

# Введение

Наблюдая за цифровизацией современного общества, связей внутри общества, рабочего процесса или учебного процесса, начинает казаться что из классического формата взаимосвязи между людьми остались лишь рудименты, конечно, я не говорю о работе, где требуется взаимодействие между людьми лицом к лицу, тем не менее, цифровизация взаимосвязей данного типа будет не лишней, и станет даже очень полезной, сохранит много времени.

Основной целью моей ВКР является реализация «Электронный Ассистент Академического Советника». Несмотря на название, Ассистент будет включать в себя не только инструменты для академического советника, а также полезный функционал для студентов. Одной из целей моей работы является повысить уровень коммуникации между Студентом и Академическим советником.

Исходя из вышеперечисленного можно сделать какие-то выводы. Мою работу несмотря на название, можно будет позиционировать как систему, а система под собой представляет взаимосвязь между чем-либо, ну и конечно же взаимосвязь подсистем образующих единое целое.  
Из этого можно сделать вывод, что свою работу я буду представлять и разрабатывать как общую систему состоящую из отдельны подсистем, это может дать большое преимущество в будущем, особенно в том случае если эту систему кто-то захочет дополнить новым элементом (подсистемой).  
Системы бывают разных видов, встроенные, программно-насыщенные, вычислительно-ориентированные, в моем случае система будет опираться на основу от «Программно-насыщенной системы», у каждой системы есть свои приоритеты опираясь на общую базовую характеристики всех систем в целом, а именно Цель, Функции, Входы, Обработка, Выходы, Временные характеристики, Оборудование и Типы пользователей которые должны быть в этой системе.

Теперь вернемся к главными принципами программно-насыщенной системы:

* Целью такой системы считается работа с данными, в том числе это могут быть как большие массивы данных, итогом должно стать либо постоянное поддержание какого-либо решения, либо приобретение какого-либо знания.
* Система в основном выполняет транзакционные функции.
* В качестве входной информации должны быть информация или объекты.
* Непосредственно в обработке должны участвовать как манипуляция с данными, так и графический интерфейс пользователя, и данные должны обмениваться по сети.
* Временные характеристики у этого вида системы очень вариативные, в моем случае я выбрал реальное время.
* В качестве оборудования обычно выступает клиент-серверная архитектура.
* В качестве пользователей выступают руководители и персонал либо же подчиненные, или под ответственные группы.

Поговорим об «Ассистенте». Если рассматривать данное понятие, не опираясь на слово «Электронный», то мы увидим следующие определения в интернете:

1. Помощник какого-либо научного деятеля высокого ранга.
2. Или самая младшая должность в Высших Учебных Заведениях.

Отчасти это определение подходит к сути моей ВКР, но все же это Электронный Ассистент, поэтому давайте попробуем понять какие функции, и вообще в целом, что из себя должен представлять Электронный Ассистент.

Если мы обратимся к поисковой системе Google, она нам выдаст в первую очередь информацию о «Виртуальном ассистенте», и почитав статьи о виртуальных ассистентах мы сразу же поймем, о чем идет речь, а именно о так называемых агентах, с которыми знаком почти каждый человек, ведь агент есть даже в вашем телефоне, в качестве примеров будут показаны агенты в лице «Google Ассистент», «Siri», «Алиса» и т.п... Эти виртуальные ассистенты в действительности можно назвать агентами, т.к. они выполняют просто колоссальный объем требований, думаю это определение если и подходит к моей системе, то только в словах «Виртуальный» или в том плане, что он выполняет различного рода запросы внутри подсистем и системы в целом.

Читая больше информации, в итоге мы придем к такому определению как «Виртуальный цифровой помощник» его так же упоминают как VDA (от англ. Virtual Digital Assistant). Что же представляет собой VDA? Как говорит нам источник, VDA – в большинстве случаев, выполнен в виде веб-сервиса или приложения, как для персональных компьютеров и ноутбуков, так и для более компактных смартфонов. Почти всегда VDA является по сути, некой формой секретаря, который решает следующие проблемы:

* Планирование графика.
* Организация повседневных дел или их выполнение.

Снова определение подходит к моей работе частично, но отмечу что все же, эти определения в какой-то степени подходят для описания моей работы, и соответственно можно сделать логическое заключение о том что в целом, часть функций ассистента/помощника не сделают его менее помощником или ассистентом.  
В заключении введения хочу отметить, что цифровизация даже небольшой части большой системы, а это ускорение внутренних процессов и без сомнений, это так же и толчок к необратимому развитию всей инфраструктуры нашего с вами Высшего Учебного Заведения.

# Глава 1: Обзор и анализ

## Анализ области темы

Как было описано в ведении, тема моей ВКР - Электронный Ассистент Академического Советника. Который представляет собой систему, а точнее подсистему АОС, которая отвечает за автоматизации процесса взаимодействий, между Академическим советником и Студентом, и эта подсистема предназначена для следующих целей:

* По возможности интеграция с имеющейся системой автоматизации AVN.
* Повышение осведомленности, как студентов о своей успеваемости, так и академических советников, о ситуации со студентами и группами в целом.
* Экономия времени академических советников, посредством уменьшения количества задаваемых вопросов и взаимодействием с системой вместо отдельного, персонального взаимодействия со студентом.
* Усовершенствование имеющейся базы данных, и развитие проекта в независимом направлении, в последствии доработка и полный переход на эту систему.

Большинство таких подсистем, включают в себя следующий функционал:

* Личный кабинет.
* Система наглядного контроля успеваемости студентов. Эта система предназначена для анализа успеваемости каждого студента, а так же анализ успеваемости группы.
* Система обязательно должна иметь встроенный поиск, какой-либо информации.
* Система должна нести полезную информационную нагрузку
* Должна предупреждать академических советников о возможных проблемных ситуациях.
* Должна помогать студентам путем информирования их о возможных проблемных ситуациях, а также помогать упростить усваивание определенного рода информации.
* Так же часто встречаются системы, в которых имеется календарь событий.
* Новостная лента тоже встречается достаточно часто.
* И конечно же, уведомлять пользователей в случае надобности.

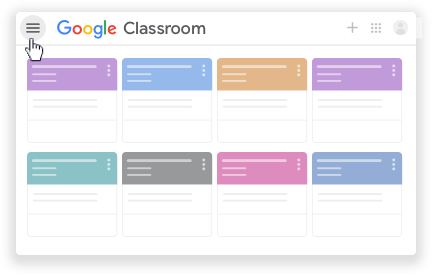
В последнее время можно заметить системы автоматизации процессов почти во всех отраслях начиная от тяжелой промышленности до малого бизнеса. Так же это не обошло и образовательный сектор, на рынке множество примеров систем разного рода направления, и все они выполняют почти схожие функции, но к ним мы вернемся позже.

В заключении хочу добавить, что учебный процесс как показывает практика, бывает достаточно непредсказуем, и для того, чтобы избежать некоторых неожиданностей, вводятся системы информирования и автоматизации процессов. Так же хочется отметить, что человеческий фактор остается и будет оставаться еще очень долгое время, в связи с неприспособленностью людей к такому быстрому развитию информационных технологий, а также восприятия в серьёз данного рода систем. Может показаться что студенту и так может быть известны, различного рода ситуации с его успеваемостью, но мы пойдем иным путем и не будем обращать на такие мнения внимания, т.к. это не то, что нам нужно, мы за развитие IT-индустрии, и далее в данном изложении будем описаться на принцип «Осведомлен – вооружен».

## Обзор существующих систем.

Т.к. существующие системы в основном узко специализированные и представляют собой продукты, спроектированные под определенный формат учебного заведения, схожие системы будет достаточно проблематично найти.   
Т.к. моя работа достаточно узконаправленная, сравнение я буду проводить с подсистемами систем по организации учебного процесса.

### Google Classroom



(рис. 1)  
Главное окно Google Classroom

Начнем с самого популярного и известного сервиса, и это Google Classroom, он является бесплатным, и для его использования требуется лишь аккаунт Google. Он выполнена как веб-сервис, цель которого упрощение распространения информации и ее определение оценки ее усваивания. Основным преимуществом данной системы является ее независимость, и бесплатное распространение файлов, т.е. за место на сервере не нужно платить. Но нас больше интересуют нужные нам функции, и первой из них будут аккаунты.

- Аккаунты в данной системе реализованы очень просто, любой пользователь может как студентом, так и преподавателем, для входа в систему требуется всего лишь аккаунт почтовой системы Google Mail. Да это очень удобно, но эта система общедоступная и не может называться узкоспециализированной. Поэтому мы продолжим.

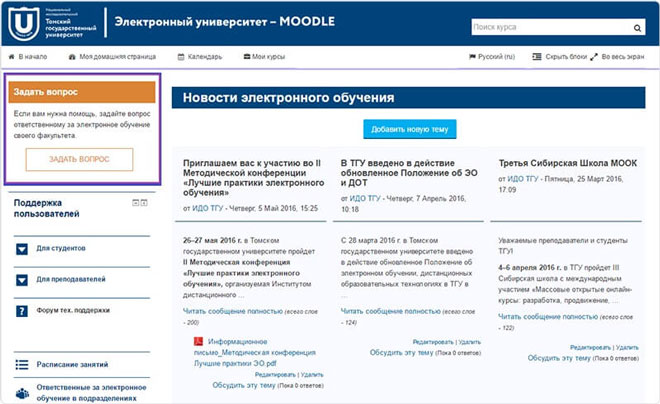
- Уведомления приходят во время публикации задания, и снова это не совсем то, что нам нужно, но тем не менее мы можем заметить, что как я и написал ранее уведомления, напрямую связаны с данными о курсах студента, и временем, соответственно может возникнуть вопрос, а есть ли в этой системе календарь?

- Действительно в этой системе есть календарь, но это календарь всей системы Google, к которой привязан ваш аккаунт. Т.е. вперемешку с отмеченными в календаре днями рождений, добавится информация, опубликованная преподавателем. Это можно отнести как к плюсу, так и к минусу. К плюсу за то, что все в одном календаре, а к минусу, информационная нагрузка на человека, в свободное от занятий время.

- Новостной ленты как таковой тут нет, но есть лента курса, куда выкладываются посты с заданиями или информацией по курсу. С натяжкой, но все же можно назвать это новостью, особенно когда она еще и приходит в качестве уведомления.

- Контроль успеваемости реализован достаточно хорошо, но за успеваемостью могут следить только преподаватели курса, т.е. несмотря на то, что информации по успеваемости больше, но т.к. моя работа, узконаправленная мне, будет нужна информация о всех подконтрольных группах.

### Moodle

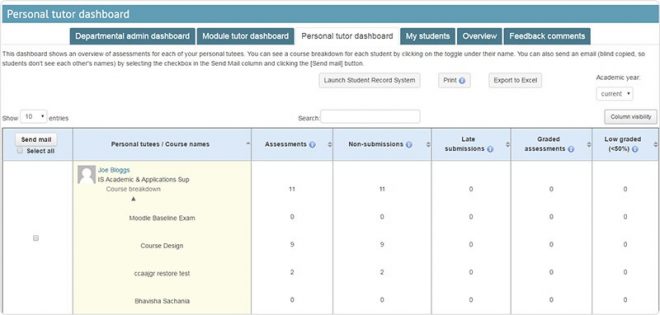


(рис. 2)  
Главная страница интерфейса Moodle

Перейдем к более продвинутой системе по автоматизации учебного процесса, точнее можно сказать, система для полного перехода на онлайн обучение, и это Moodle, и как отмечает разработчик она бесплатна, с открытым исходным кодом.

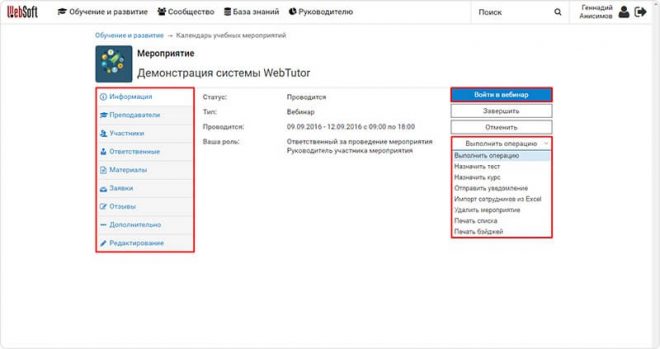
В данной системе есть все необходимые инструменты для организации учебного процесса. Так же эта система может работать в интеграции с некоторыми другими приложениями. Она так же может дополняться необходимыми плагинами что я считаю очень полезным свойством данной системы.

Так же очень полезной подсистемой является система отчетов, она полностью настраиваемая. И может включать в себя до 38 пунктов, в зависимости от плагинов. Можно, например просмотреть сколько времени студенты тратят на курсы или, например посмотреть успеваемость каждого пользователя, количество входов в систему и т.п., всего не перечислить, отчетность у данной системы действительно на высоком уровне, до сих пор непонятно как Google Classroom может пользоваться такой популярностью, когда имеется такая система как Moodle.



(рис. 3)  
Подсистема отчетов успеваемости Moodle

### WebTutor



(рис. 4)  
Главное страница интерфейса WebTutor

Еще одной интересной системой является WebTutor. Рассматриваемая система не является бесплатной модули покупаются. Так же, как и предыдущий программный продукт, этот так же построен на модулях, модули устанавливаются в зависимости от необходимых задач.

И снова мы видим возможность интеграции с другими сервисами. Присутствует загрузка файлов любого формата.

В отличие от того же Google Classroom пользователей можно распределять по группам, создавать и назначать им уникальные роли, которые в свою очередь будут определять доступ к нужному контенту.

Система отчетности, тут тоже, как и Moodle на высоком уровне, можно собирать любую информацию внутри системы, и ее большим плюсом в системе отчетности можно отметить возможность вывода отчетов в виде графиков и диаграмм.

### Выводы

Проведя относительно небольшой анализ, стало понятно, что уведомления, новости, система отчетности, календарь событий и поиск отдельного студента, должны быть в такой системе в обязательном порядке.

Уникальным мой проект будет делать то, что он узконаправленный, и анализ идет по большей степени не отдельного курса, а всех курсов. А также расчетный калькулятор, который подходит для системы AVN.

И так как моя система в большей степени будет являться информационной, и ориентирована на предотвращение неприятных ситуаций, и обеспечение интерактивной коммуникации между студентом и академ советником, а не преподавателем, то отсутствие обучающего материала, курсов и т.п. я не считаю минусом.

## Функции системы

### Основные функции.

* Сбор имеющихся данных.

В качестве примера представим, что у нас уже есть база данных со всеми данными студентов, с их успеваемостью, с данными о их зарегистрированных предметах и т.п. и эта система должна собирать такие данные, в непрерывном режиме. Это необходимо для постоянной точности выходных данных.

* Анализ собранных данных.

Далее по уже имеющимся данным система должна проводить нужный, определенный заранее расчет, для формирования выходных данных.

* Внесение новых данных в БД

После совершения анализа и расчета по данным, система внесет результаты вычислений в БД, это делается для сохранности данных и более удобному их выводу, несмотря на данное решение, имеют место быть так же данные без внесения в БД, но обычно, такие данные не статичны и постоянно изменяются, примером может стать вывод времени.

* Вывод информации

Из БД берется информация и выводится в нужном представлении непосредственно в подготовленный для этого интерфейс. Данные могут быть любого типа, все зависит от инструментов, с которыми ведется работа.

### Подсистемы и их функции

* Подсистема профилей.

Т.к. система работает с большим массивом данных, необходимо эти данные привязывать к определенным пользователям. В случае моей ВКР, пользователи будут делиться по ролям, в нашем случае будет 3 типа пользователей:

1. Администратор система – этой ролью будет обладать доверенное лицо, способное в случае чего конфигурироваться всю систему или ее подсистемы, добавлять пользователей.
2. Академический советник – основная роль в системе, создатель событий, новостей кафедры, уведомлений и наблюдатель за ситуацией в группах и успеваемостью студентов в целом.
3. Студент – вторая основная роль в системе, может только дополнять свой профиль, читать новости, просматривать предстоящие события и получать уведомления от академического советника.

В соответствии с ролями будет определяться отображаемая информация.

* Подсистема контроля успеваемости.

В данной системе будет производиться анализ успеваемости студентов, и успеваемость всей группы в целом. Информация будет выводиться в дружелюбном виде, посредством адаптивных таблиц. Для пользователей с ролью академ. советника, в списке будет отображаться студент или группа, его основные, важные данные, и проблемы с успеваемостью, а именно, задолженности, недостаток баллов, и неоплаченные в срок кредиты. Критические моменты будут выделяться, и иметь кнопку отправки студенту уведомления. Для пользователей с ролью студент, будет отображаться так же таблица, только уже в более развернутом формате и о его успеваемости по каждому предмету отдельно. В кабинет каждого пользователя, будут контактные данные в виде E-Mail и номера мобильного телефона, так же можно будет добавить дополнительное поле для других контактных данных.

* Подсистема уведомлений.

Как было сказано ранее, в таблицах будут указываться критические моменты, снова представим ситуацию, допустим у студента за неделю до начала модулей, все еще не оплачена частичная или полная сумма контракта, в этом случае за неделю до модулей, система создает поле в таблице уведомлений, о том что у такого-то студента есть долг, и у студента это уведомление появляется в разделе уведомлений, академическому советнику уведомление не будет приходить, у него эта информация будет отображаться на главном экране, и лишь в случае если студент все еще не оплатил контракт, у академического советника появится кнопка о критическом уведомлении студента, или же академ. советник сможет связаться с ним перейдя в профиль студента и взяв оттуда нужные контактные данные. Доступ к критическим уведомлениям появится, условно, за 3 дня до начала модулей. Т.е. у нас есть данные о оплате студентом зарегистрированных кредитов, и соответственно, чтобы узнать сколько осталось оплатить студенту достаточно просто уже оплаченную сумму, вычесть из общей суммы зарегистрированных кредитов. Звучит очень просто, но на расчете дело не заканчивается, система должна занести эти данные в свои таблицы БД, допустим она будет называться «notifications», чтобы после можно было их вывести. Итак, у нас имеются данные о том, что какой-то студент не оплатил контракт и до начала модулей осталось условно неделя, система анализирую данные понимает, что что-то не так, и рассылает уведомления как студенту, так и академическому советнику о том, что срок оплаты уже почти подошел, и студенту стоит поторопиться, а академическому советнику эти данные могут быть полезны.

* Календарь событий

Чтобы предыдущий пункт работал, ему необходимо брать данные о времени, а их можно взять напрямую из базы данных, но для улучшения функциональности системы, в большинстве схожих систем вводятся такие инструменты как «Event Calendar», т.е. Календарь событий, как академ. советникам, так и студентам, этот календарь будет полезен. На нем они смогут увидеть все предстоящие события, время открытого или закрытого доступа к AVN, включая модули, праздники, сессию, расписание семестров и т.п.  
Календарь или таблица с данными о времени, необходимы в обязательном порядке.

* Новостная подсистема

Чтобы не уведомлять каждого студента отдельно каком-то внеплановом событии или держать в курсе подачи необходимых документов и т.п., во многих системах имеются новостные панели, в них обычно сообщается внеплановая дополнительная информация. В моем же случае данная подсистема будет выполнять схожие действия, надеюсь получится даже реализовать прикрепление к новости документа.

* Подсистема регистрационного расчета (для студентов)

Часто случается, что студент из-за неведения ситуации, в итоге оказывается в неприятном положении при регистрации «как он рассчитывал», подсистема регистрационного расчета должна будет устранить этот фактор риска.  
В ней так же будут браться данные уже по имеющимся кредитам, и по тем предметам которые идут по учебному плану и сравниваться, таким образом допустим студент имеющий задолженность на втором курсе, будет иметь недостающее количество кредитов, соответственно система этот факт обнаружит, и покажет студенту какой у него недобор кредитов, и предложит на выбор зарегистрировать в качестве дополнительных предметов, предметы которых у этого студента нет в базе данных, или есть, но уже с просроченным сроком сдачи.

С основными функциями системы и ее подсистем мы ознакомились, теперь переходим к следующей главе, где сравним со схожими системами.

## Постановка задач

**Аккаунт пользователя.**

Система будет закрытой, и эту проблему мы легко решим ограничением доступа к системе посредством входа через личный аккаунт пользователя. Но перед этим необходимо создать продуманный интерфейс чтобы примерно ориентироваться какие в данной таблице будут поля. Ориентировочно сновными же полями в базе данных станут:

1. E-mail пользователя.
2. Пароль.
3. ФИО пользователя.
4. Номер телефона.
5. Тип пользователя.

Как я и указывал раньше пользователи будут делиться на 3 роли.

1. Администратор
2. Академический советник
3. Преподаватель
4. Студент.

Дополнительная поля для академического советника:

1. ID Контроля

Дополнительная поля для студента:

1. ID Группы
2. ID Направления
3. Количество оплаченных кредитов
4. Количество закрытых кредитов за все время

Таблица группы (как связь между студентом и академ. советником)

1. ID Контроля
2. Название группы
3. Название факультета
4. Название отделения
5. Форма обучения
6. Форма оплаты
7. Тип группы (бакалавр/магистранты)

Так же, отдельными таблицами хотелось бы отметить предметы студентов.

1. ID направления
2. Название дисциплины
3. Количество кредитов дисциплины
4. Форма контроля
5. Номер модуля

Данные о предметах будут хранится в отдельной странице пройденных или не пройденных предметов.

1. ID студента
2. ID дисциплины
3. Статус дисциплины
4. Количество оплаченных кредитов

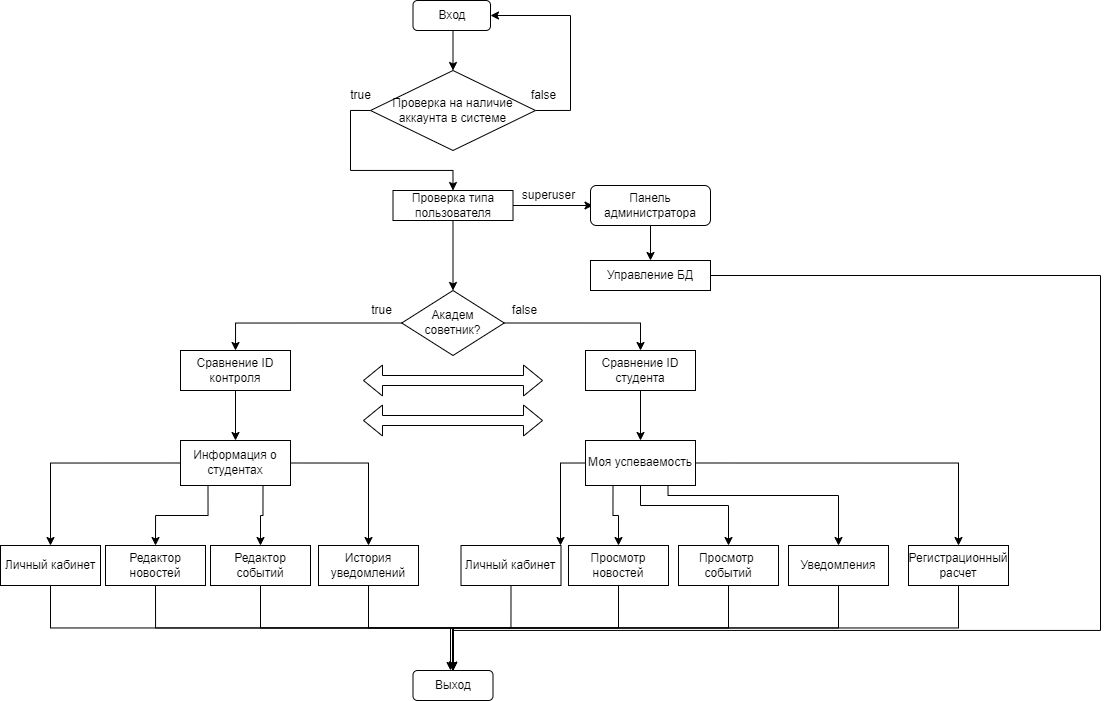
И предметы можно будет связать со студентом через таблицу группы.

Все связи и вывод данных из других таблиц будет проходить по выбранной роли.

Студентам будет закрыт доступ к какому-либо редактированию на уровне кода. Так же можно будет добавить отдельно поле для дополнительных контактных данных.

### Основной алгоритм системы

Основной алгоритм системы, должен будет объединить в себе все подсистемы, и обеспечить корректный вывод выходной информации в интерфейсе системы.



(рис. 5)  
Основной алгоритм системы

# Глава 2: Разработка Электронного Ассистента Академического Советника

## 2.1 Модификация стандартной установки

Т.к. установка Django является стандартной и описана на многих веб-ресурсах, особого внимания этому мы не будем уделять.

В этом разделе мы больше заострим внимание на некоторых отступлениях от стандартной установки.

Т.к. нам не совсем подходят стандартные инструменты Django, в частности стандартное СУБД SQL-Lite.

Придется совершить несколько манипуляций.

Для того чтобы подключить к нашему проекту PostgreSQL в первую очередь нам придется установить установим psycopg2. Этот модуль позволяет корректно работать с базой данных PostgreSQL. А по отзывам большинства разработчиков, Django работает с PostgreSQL даже лучше, чем со встроенной SQL-Lite. Для установки psycopg2 потребуется прописать в консоли, запущенной в виртуальном пространстве venv следующее.

    (venv): pip install psycopg2

Следующим шагом, нам потребуется создать базу данных в PostgreSQL.  
В запущенном окне SQLShell (psql).exe создаем пользователя, БД, и назначаем пользователя владельцем БД.

create user owner\_db with password 'um5V5ZFvJQ1FvgZEsqC4QaK5PF3HQeq3';

alter role owner\_db set client\_encoding to 'utf8';

alter role owner\_db set default\_transaction\_isolation to 'read committed';

alter role owner\_db set timezone to 'UTC';

create database project\_db owner owner\_db;

Следующее что нам потребуется, уже стандартным способом, используя функции psycopg2, подключиться к созданной БД и войти в нее через созданного владельца.

## 2.2 Реализация разбития пользователей на группы.

Реализовать разбитие пользователей на группы мы сможем благодаря грамотному построению структуры базы данных.

В первую очередь, нам нужно будет определить главную модель пользователя.

Что мы и сделаем для этого создадим новое приложение проекта, назовем его **app\_users**, внутри приложения в файле **models.py**, создадим новую модель и назовём ее user она будет наследоваться от стандартной модели пользователей Django.

В этой модели мы создадим следующие поля:

* Электронная почта
* Номер телефона
* ФИО
* Советник этот пользователь или нет
* Учитель этот пользователь или нет
* Администратор этот пользователь или нет
* Активен ли профиль пользователя или нет

Так же пропишем логику работы, новой модели пользователей.

class UserManager(*BaseUserManager*):

    use\_in\_migrations = True

    def \_create\_user(*self*, *email*, *phone\_number*, *full\_name*, *password*, \*\**extra\_fields*):

        if not email:

            raise *ValueError*('Должен быть введен Email')

        if not phone\_number:

            raise *ValueError*('Должен быть введен номер телефона')

        if not full\_name:

            raise *ValueError*('Должны быть введены ФИО')

        email = *self*.normalize\_email(email)

        user = *self*.model(*email*=email, *phone\_number*=phone\_number, *full\_name*=full\_name, \*\*extra\_fields)

        user.set\_password(password)

        user.save(*using*=*self*.\_db)

        return user

    def create\_user(*self*, *email*, *phone\_number*, *full\_name*, *password*=None, \*\**extra\_fields*):

        extra\_fields.setdefault('is\_staff', False)

        extra\_fields.setdefault('is\_superuser', False)

        return *self*.\_create\_user(email, phone\_number, full\_name, password, \*\*extra\_fields)

    def create\_superuser(*self*, *email*, *phone\_number*, *full\_name*, *password*, \*\**extra\_fields*):

        extra\_fields.setdefault('is\_staff', True)

        extra\_fields.setdefault('is\_superuser', True)

        if extra\_fields.get('is\_staff') is not True:

            raise *ValueError*('Суперпользователь должен иметь значения is\_staff=True.')

        if extra\_fields.get('is\_superuser') is not True:

            raise *ValueError*('Суперпользователь должен иметь значения is\_superuser=True.')

        return *self*.\_create\_user(email, phone\_number, full\_name, password, \*\*extra\_fields)

from django.core.mail import send\_mail

from django.contrib.auth.base\_user import AbstractBaseUser

from django.contrib.auth.models import PermissionsMixin

from django.contrib.auth.validators import UnicodeUsernameValidator

from django.db import models

from django.utils.translation import ugettext\_lazy as \_

class User(*AbstractBaseUser*, *PermissionsMixin*):

    email = models.EmailField(\_('Почта'), *max\_length*=125, *unique*=True)

    phone\_number = models.CharField(\_('Номер телефона'), *max\_length*=13, *unique*=True)

    full\_name = models.CharField(\_('ФИО'), *max\_length*=125)

#   profile\_photo = models.ImageField(\_('Фото профиля'), upload\_to='avatars/', null=True, blank=True)

    is\_advisor = models.BooleanField(\_('Cоветник'), *default*=False)

    is\_teacher = models.BooleanField(\_('Преподаватель'), *default*=False)

    is\_active = models.BooleanField(\_('Профиль активен'), *default*=True)

    is\_staff = models.BooleanField(\_('Доступ к админ-панели'), *default*=False)

    USERNAME\_FIELD = 'email'

    REQUIRED\_FIELDS = [ 'phone\_number', 'full\_name']

    class Meta:

        verbose\_name = 'Пользователя'

        verbose\_name\_plural = '1. Пользователи'

        ordering = ['-is\_superuser', '-is\_advisor', '-is\_teacher', 'full\_name']

Далее нам понадобиться произвести миграцию с базой данных PostgreSQL.

После этого модель стала рабочей, но доступа у нас к ней нет.

Чтобы получить доступ к модели нам нужно будет создать представление в панели администратора. Для этого откроем/создадим файл **admin.py** в папке этого же проекта. И в нем опишем отображение, так же сделаем поля пригодными для использования.

@admin.register(User)

class UserAdmin(*UserAdmin*):

    model = User

    list\_display = ('id', 'full\_name', 'phone\_number', 'email', 'is\_advisor', 'is\_teacher', 'is\_active', 'is\_staff')

    list\_display\_links = ('id', 'full\_name', 'phone\_number', 'email')

    search\_fields = ('full\_name', 'phone\_number', 'email')

    list\_editable = ('is\_active',)

    list\_filter = ('is\_advisor', 'is\_superuser', 'is\_teacher')

    fieldsets = (

        ('Редактор', {

            'classes': ('wide',),

            'fields': ('full\_name', 'phone\_number', 'email', 'is\_active',)

                }),

            )

    add\_fieldsets = (

        ('Создание', {

            'classes': ('wide',),

            'fields': ('full\_name', 'email', 'phone\_number', 'password1', 'password2', 'is\_active', 'is\_advisor', 'is\_teacher'),

        }),

    )

Не смотря на проделанные действия у нас все еще нет доступа к панели администратора. Т.к. у нас еще нет созданных пользователей.

Чтобы решить эту проблемы откроем консоль и запустим команду файла Django

(venv): py **manage.py** createuser

У нас появится приписанная нами же регистрация пользователя, которая потребует все необходимые данные.

    +----------------------+---------------------------------------------+

    |    Почта             |    e3a.assistant@gmail.com                  |

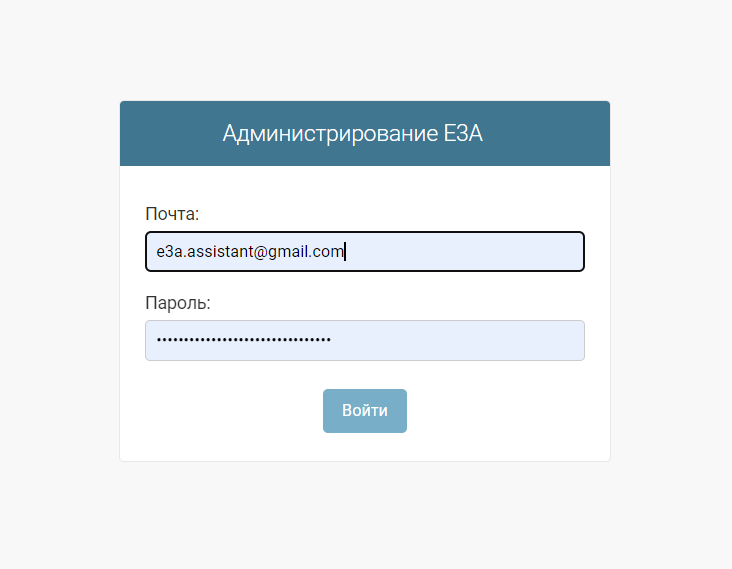
    |    ФИО               |    Администратор Электронного Ассистента    |

    |    Номер телефона    |    +996111222222                            |

    |    Пароль            |    lkd9j7WrAtSHBM4xIb8cBac4LkolOqvs         |

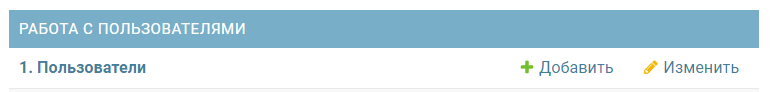
    +----------------------+---------------------------------------------+

Теперь, мы можем войти в панель администратора, используя ссылку /admin и данные пользователя.



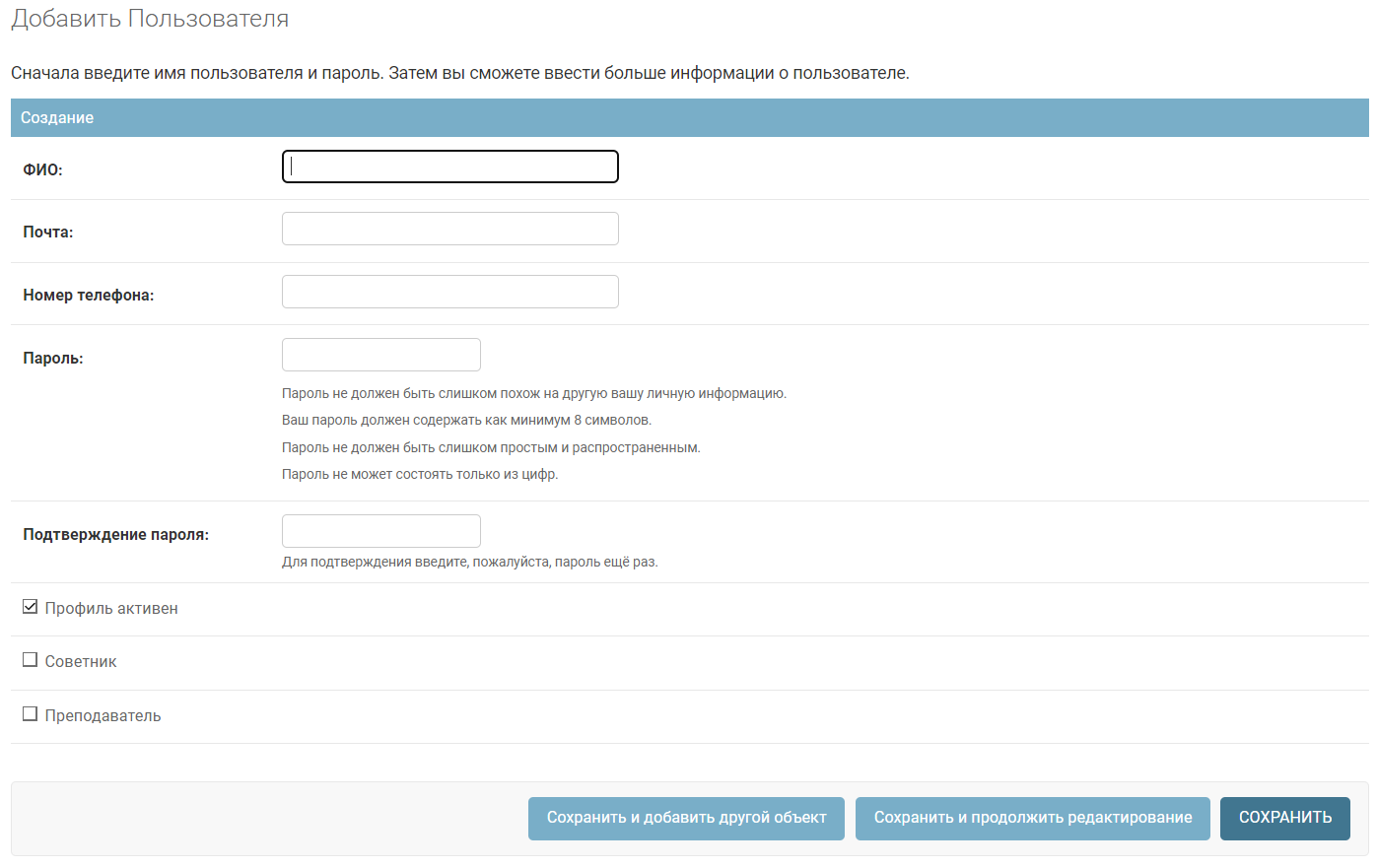
(рис. 6)  
Окно входа в панель администратора

И как мы можем заметить, у нас появилась панель управления пользователям.



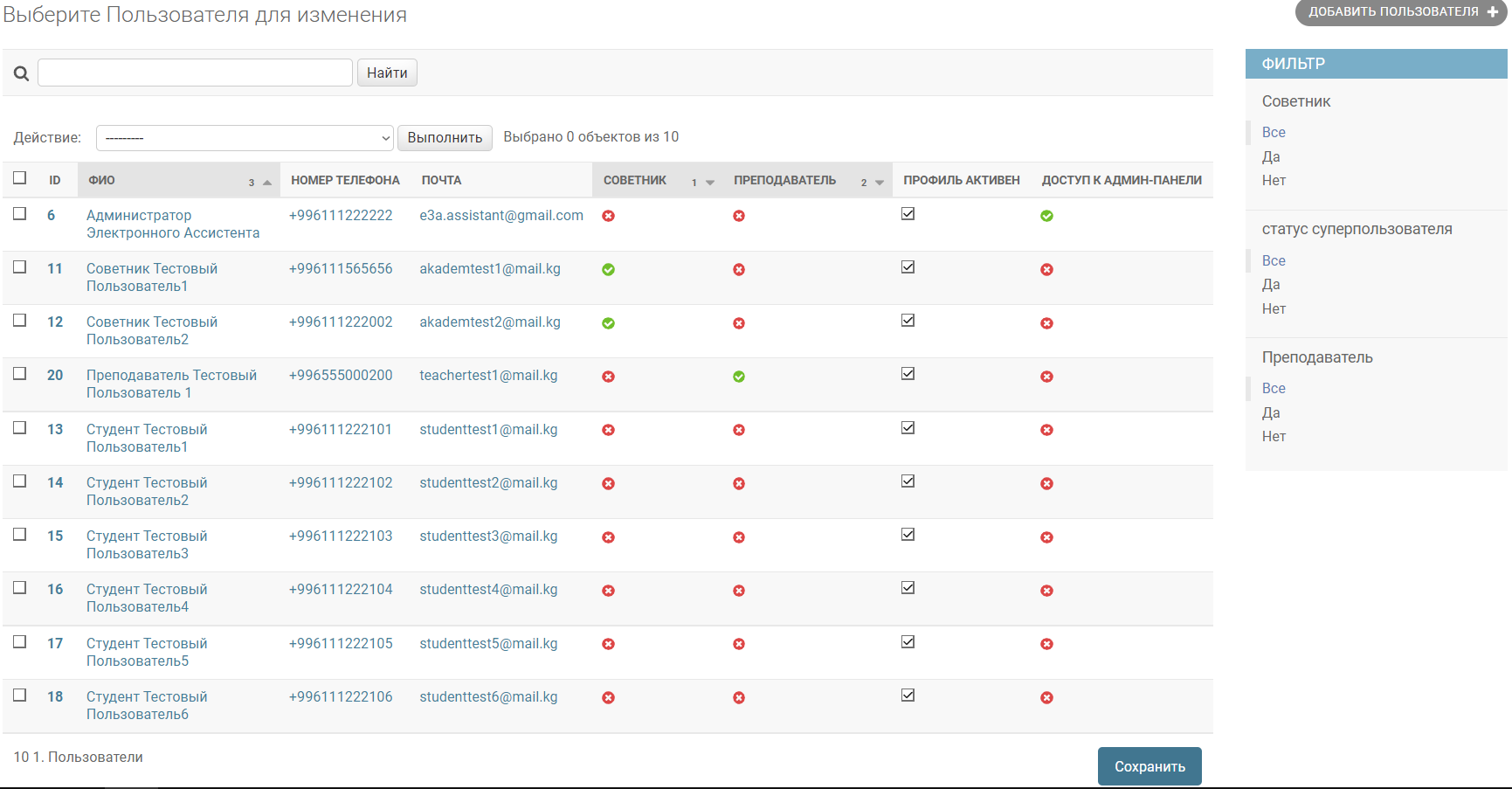
(рис. 7)  
Панель работы с пользователями

При нажатии на кнопку добавить, у нас появится окно добавления пользователя. Т.к. система узко-ориентированная именно такой способ добавления пользователей будет актуальным. Это поможет избежать беспорядка с регистрацией абсолютно любого человека. Пользователи будут добавляться администратором системы.



(рис. 7)  
Окно создания пользователя

Для наглядности создадим несколько тестовых пользователей.



(рис. 8)  
Окно отображения созданных пользователей

## 2.3 Реализация входа в систему.

Следующим этапом нам нужно будет создать окно для входа пользователей в пользовательский интерфейс. Для этого в папке приложения **app\_users** в **urls.py,** нам будет необходимо подключить формы для входа, которые уже автоматически подстроились под нашу модель пользователя, благодаря тому что мы наследовали модель от стандартной модели.

Создадим шаблон для входа, для этого в любом из приложений создадим папку **template,** в ней создадим папку **registrations**, в ней создадим файл с названием login.html. В файле **urls.py** укажем путь к данному файлу чтобы он мог корректно работать.

А также укажем названия заголовков панели администратора.

admin.site.site\_title = 'Админпанель E3A'

admin.site.site\_header = 'Администрирование E3A'

admin.site.site\_title = 'Сайт администрации E3A'

urlpatterns = [

        path('empty\_login/', views.login, *name*='login'),

        path('', include('django.contrib.auth.urls')),

        path('', views.empty, *name*='empty'),

]

В файле **views.py** создадим условие проверяющее авторизован ли пользователь. Если нет, то будет направлен на представление входа, если авторизован, то будет перенаправлен, на другую страницу, которую создадим дальше.

def login(*request*):

        if request.user.is\_authenticated == False:

                return render(request, 'registration/login.html', {'title':'Страница входа'})

Т.к. стандартные формы Django выглядят немного лучше стандартных форм языка HTML, мы можем создать свои представления форм, это займет немного времени, но проблемой будет дублирование кода, что нарушает одно из правил программирования: «Не повторяй самого себя».

И чтобы обойти эту проблему, мы воспользуемся модулем, меняющим стили форм на стили форм Bootstrap 5. Для этого в консоли, в эмулированном пространстве пропишем:

pip install crispy-bootstrap5

Теперь вызовем форму в шаблоне, используя фильтр **| crispy.** Дополним страницу и оформим ее стиль.

<form *action*="{% url 'login' %}" *method*="post" *class*="p-4 p-md-5 border rounded-3 bg-light" *novalidate*>

<small *class*="text-danger">

 {% if form.errors %}

{{ form.errors.error }}

{% endif %}

</small>

{% csrf\_token %}

{{ form | crispy }}

<label>

<input *type*="checkbox" *value*="remember-me"> Запомнить

</label>

<button *class*="w-100 btn btn-lg btn-outline-primary" *type*="submit">Войти</button>

<hr *class*="my-4">

<small *class*="text-muted">Нажимая кнопку "Войти", вы соглашаетесь с правилами использования системы, и подтверждаете что готовы делиться персональными данными.</small>

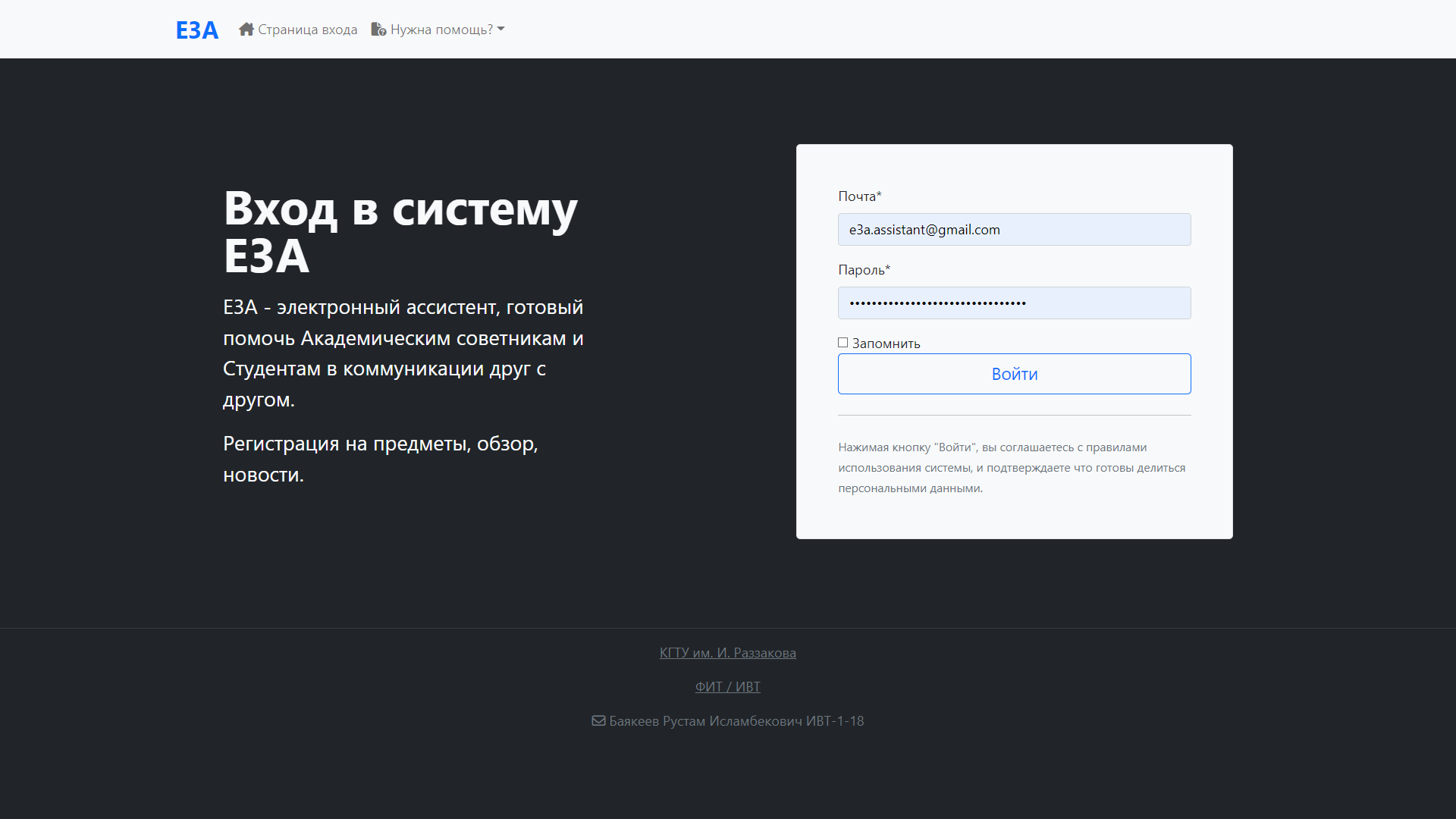
</form>

Небольшое отступление. Шаблон в Django в основном написан на HTML, CSS и Javascript в файле .html. Фреймворк Django эффективно обрабатывает и динамически генерирует веб-страницы HTML, которые видны конечному пользователю. Django в основном работает с серверной частью, поэтому для обеспечения внешнего интерфейса и макета нашего веб-сайта мы используем шаблоны. Существует два способа добавления шаблона на наш сайт в зависимости от наших потребностей.

Мы можем использовать один каталог шаблонов, который будет распределен по всему проекту.

Для каждого приложения нашего проекта мы можем создать отдельный каталог шаблонов.

Для нашего текущего проекта мы создадим единый каталог шаблонов, который для простоты будет распределен по всему проекту. Шаблоны уровня приложения обычно используются в больших проектах или в случае, если мы хотим предоставить разные макеты для каждого компонента нашей веб-страницы.



(рис. 9)  
Окно входа в систему.

## 2.4 Реализация проверки типа пользователя при авторизации.

Так как в нашей системе уже есть несколько групп пользователей (логические поля в модели User), а именно Советник, Учитель, и Администратор, и Студент (пользователь, не попадающий ни в одну из этих групп).

Нам понадобится как-то логически разбить интерфейс на разные интерфейсы, для каждого пользователя свой интерфейс.

Для этого мы создадим новое приложение и назовем его **app\_assistant**, оно будет отвечать за главную страницу и все ее функции и представления, шаблоны и т.п. так же, т.к., по сути, это основное приложение, все шаблоны будут храниться в папке **templates** этого приложения.

Приступим к реализации. Для этого создадим шаблоны для каждого пользователя в папку **templates/app\_assistant**, и в файле **view.py** укажем пути и поставим условие проверки пользователя.

def my\_redirect(*request*):

    if request.user.is\_authenticated == False:

        return redirect('login/')

    elif request.user.is\_advisor == True:

        return render(request, 'app\_assistant/assistant\_t.html', *context*=context)

    elif request.user.is\_staff == True:

        return render(request, 'admin\_redirect.html', {'title':'Администратор'})

    else:

        return render(request, 'app\_assistant/assistant\_s.html', *context*=context)

    endif

Добавим этот метод в **urls.py**, чтобы могло работать отображение страниц.э

urlpatterns = [

    path('assistant/', views.my\_redirect, *name*='my\_redirect'),

]

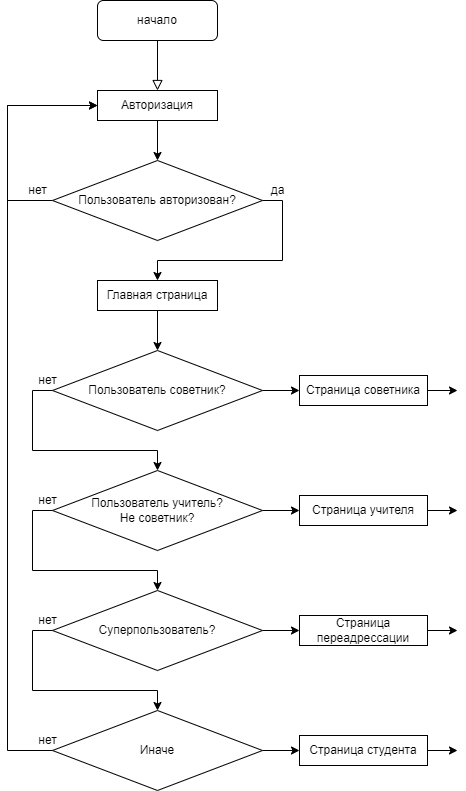
После проверок, мы понимаем, что переадресация в зависимости от типа пользователя работает как положено.

Так же добавим ссылку на главную страницу в представление **login**.

        else:

                return redirect('assistant/')

        endif



(рис. 10)  
Алгоритм проверки авторизации

## 2.5 Реализация дополнительных моделей пользователей и групп со студентами.

Для корректного отображения и работы с пользователями, общей модели будет недостаточно т.к. в дальнейшем нам нужно будет делать много запросов, и полей общей модели просто будет недостаточно.

Для этого в первую очередь дополним модель советника, по внешнему ключу, в файле моделей приложения пользователей:

class Advisor(*models*.*Model*):

    user = models.ForeignKey("User", *verbose\_name*=\_("ID Пользователя"), *on\_delete*=models.PROTECT, *null*=True)

    class Meta:

        verbose\_name = 'Советника'

        verbose\_name\_plural = '2. Советники'

        ordering = ['id',]

Следующей создадим расширяющую модель учителя по внешнему ключу:

class Teacher(*models*.*Model*):

    position = models.CharField(\_("Должность"), *max\_length*=50, *null*=True)

    user = models.ForeignKey("User", *verbose\_name*=\_("ID Пользователя"), *on\_delete*=models.PROTECT, *null*=True)

    class Meta:

        verbose\_name = 'Преподавателя'

        verbose\_name\_plural = '3. Преподаватели'

        ordering = ['id',]

Теперь после того, как мы расширили моделями явно выделяющихся пользователей, попробуем создать модель группы для студентов. Но т.к. у групп присутствует иерархия в виде факультетов и кафедр. Для начала создадим факультеты и кафедры.

Факультеты:

class Faculty(*models*.*Model*):

    name = models.CharField(("Название"), *max\_length*=150)

    short\_name = models.CharField(("Короткое название"), *max\_length*=50)

    cipher = models.IntegerField(("Шифр факультета"), *null*=True)

    def \_\_str\_\_(*self*):

        return *self*.short\_name

    class Meta:

        verbose\_name = 'Факультет'

        verbose\_name\_plural = '1. Факультеты'

        ordering = ['-name',]

Кафедры:

class Department(*models*.*Model*):

    name = models.CharField(("Название"), *max\_length*=150)

    short\_name = models.CharField(("Короткое название"), *max\_length*=50)

    сipher = models.IntegerField(("Шифр отделения"), *null*=True)

    faculty = models.ForeignKey("app\_disciplines.Faculty", *verbose\_name*=("Факультет"), *on\_delete*=models.PROTECT)

    def \_\_str\_\_(*self*):

        return *self*.short\_name

    class Meta:

        verbose\_name = ("Кафедру")

        verbose\_name\_plural = ("2. Кафедры")

Так же создадим форму обучения для групп.

class Form\_e(*models*.*Model*):

    name = models.CharField(\_("Форма обучения"), *max\_length*=25, *null*=True)

    def \_\_str\_\_(*self*):

        return *self*.name

    class Meta:

        verbose\_name = 'Форму обучения'

        verbose\_name\_plural = 'Формы обучения'

        ordering = ['-name',]

После, перейдем к созданию модели групп, и соединим ее по внешнему ключу с кафедрой, формой обучения и академическим советником.

class Groups(*models*.*Model*):

    name = models.CharField(\_("Наименование группы"), *max\_length*=25)

    advisor = models.ForeignKey("Advisor", *verbose\_name*=\_("Советник"), *on\_delete*=models.PROTECT, *null*=True)

    cours = models.IntegerField(\_("Курс"), *null*=True)

    credit\_price = models.IntegerField(\_("Стоимость кредита"), *null*=True)

    form\_of\_e = models.ForeignKey("Form\_e", *verbose\_name*=\_("Форма обучения"), *on\_delete*=models.PROTECT, *null*=True)

    department = models.ForeignKey("app\_disciplines.Department", *verbose\_name*=\_("Отделение"), *on\_delete*=models.PROTECT, *null*=True)

    def \_\_str\_\_(*self*):

        return *self*.name

    class Meta:

        verbose\_name = 'Группу'

        verbose\_name\_plural = '4. Группы'

        ordering = ['-cours', '-name']

Теперь создадим модель пользователя, и соединим ее с моделью пользователя и группой по внешним ключам:

class Student(*models*.*Model*):

    cipher = models.CharField(\_("Шифр студента"), *max\_length*=25, *unique*=True)

    user = models.ForeignKey("User", *verbose\_name*=\_("Студент"), *on\_delete*=models.PROTECT, *null*=True)

    groups = models.ForeignKey("Groups", *verbose\_name*=\_("Группа"), *on\_delete*=models.PROTECT, *null*=True)

    def \_\_str\_\_(*self*):

        return *self*.cipher

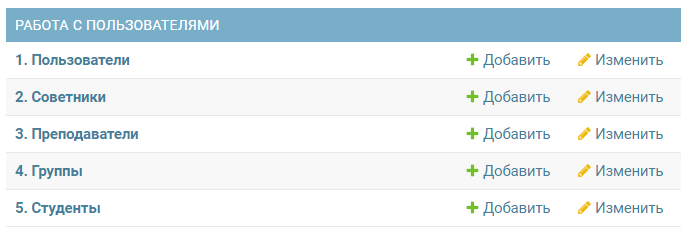
    class Meta:

        verbose\_name = 'Студента'

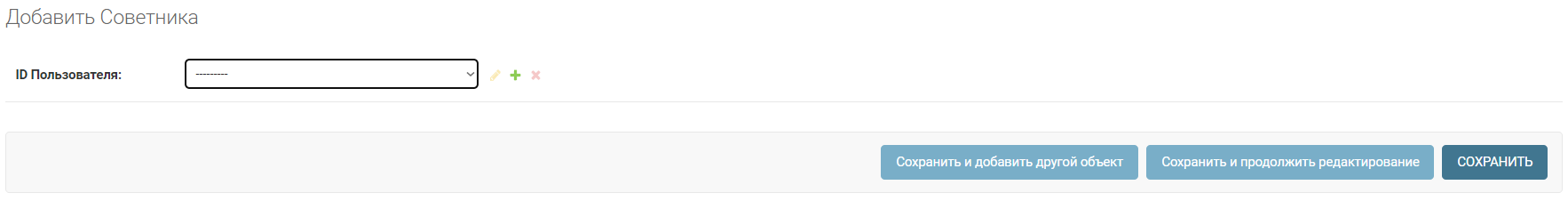
        verbose\_name\_plural = '5. Студенты'

        ordering = ['-cipher', '-id']

Как было указано раньше, нам будет необходимо добавить отображение модели в панели администратора. После регистрации моделей в панели администратора мы можем увидеть следующее.

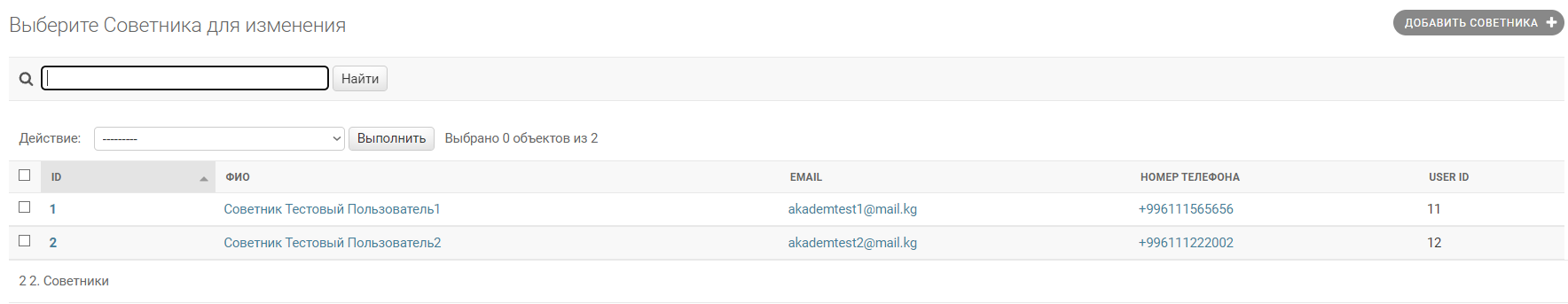


(рис. 11)  
Панель работы с пользователями и группами



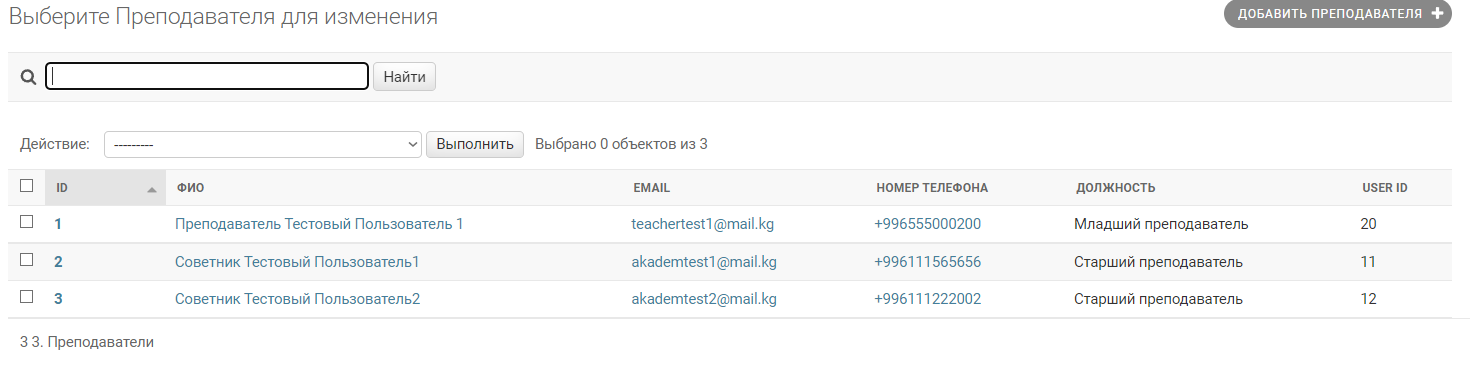
(рис. 12)  
Добавление советника

Для примера дадим существующим тестовым пользователям роли.



(рис. 13)  
Панель управления советниками

Панели советников и преподавателей схожи, поэтому не будем приводить скриншоты в пример, и просто проделаем то же самое с панелью преподавателей. И сделаем советников преподавателями тоже.



(рис. 14)  
Панель управления советниками

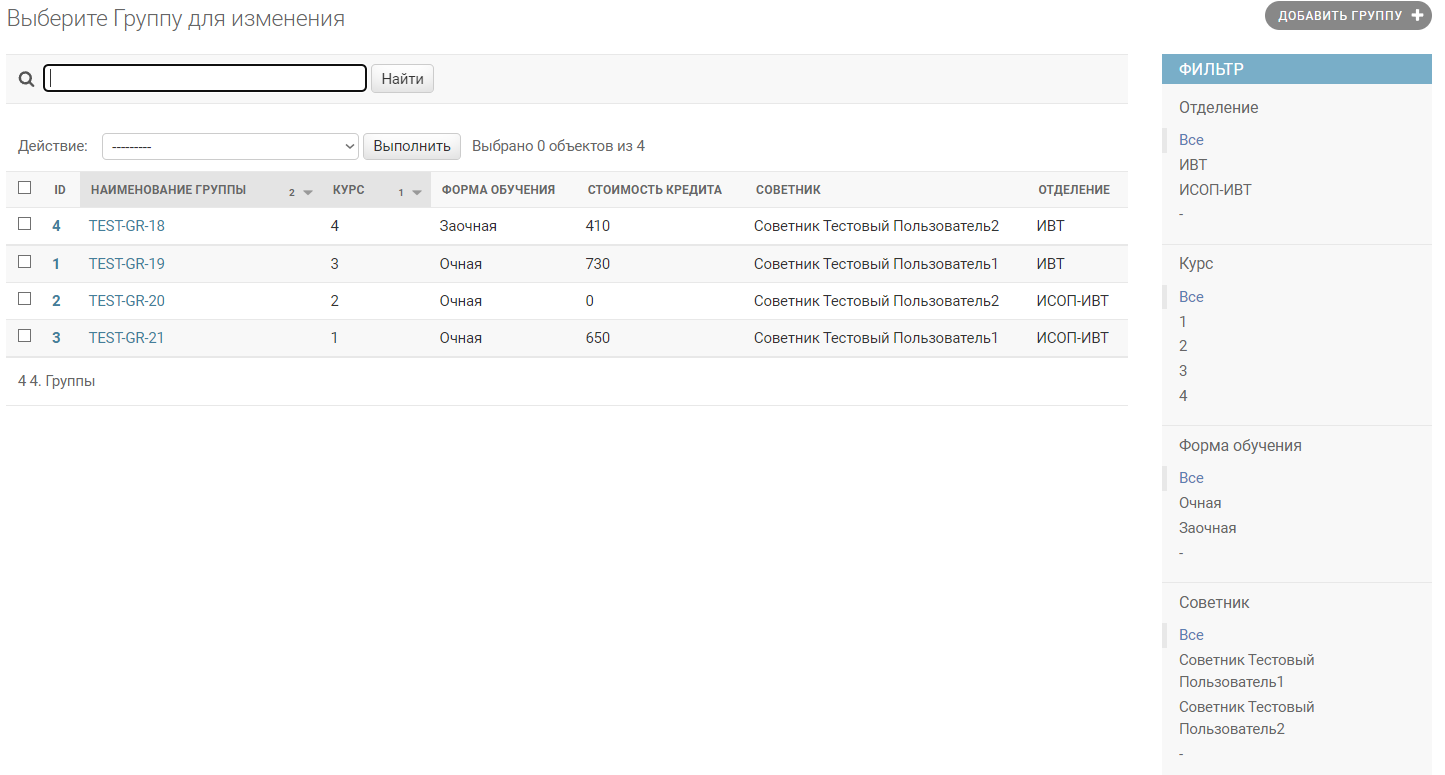
Как вы можете увидеть в вышеприведенных формах, связи в PostgreSQL под управлением Django отрабатывают отлично, как часы.

Создадим тестовые группы. Назначим для них отделения и привяжем группу к пользователю.

Так же укажем к какому отделению привязана группа.

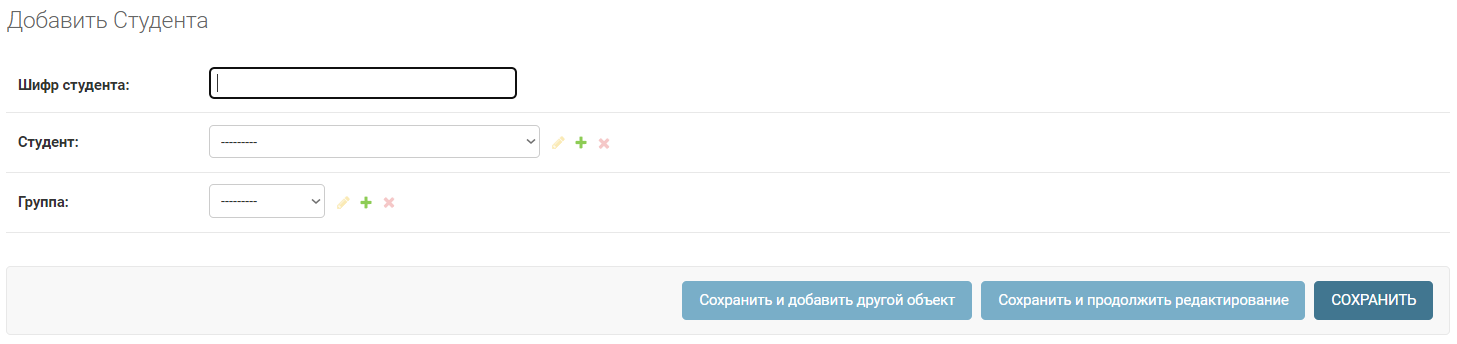


(рис. 15)  
Панель создания групп

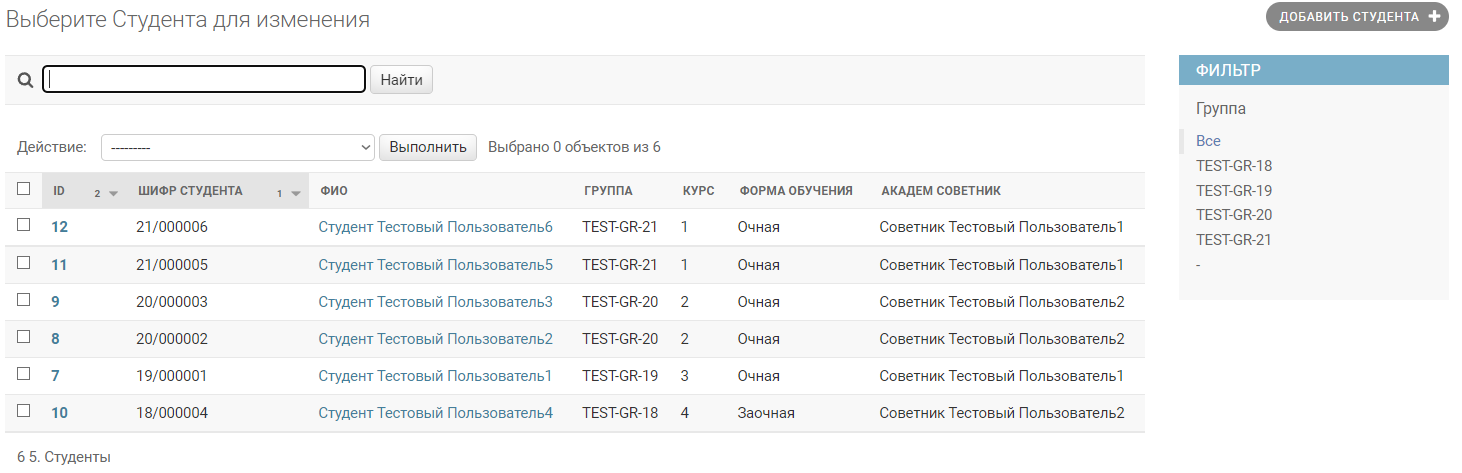


(рис. 16)  
Панель управления группами.

Осталось добавить несколько тестовых студентов.



(рис. 17)  
Панель добавления студентов

(рис. 18)  
Панель управления студентами

## 2.6 Реализация дополнительных моделей пользователей и групп со студентами.

Исходя из созданных моделей. И добавленных данных, мы можем начать построение главной страницы академического советника.

Для отображения данных из новых моделей, нам необходимо добавить в представление ассистента следующие денные.

Во первых подключить модели.

from app\_users.models import User, Advisor, Groups, Student, Teacher

И добавим код к имеющемуся условию проверки на академического советника.

user = User.objects.all()

        groups = Groups.objects.all()

        student = Student.objects.all()

        advisor = Advisor.objects.all()

        teacher = Teacher.objects.all()

        discipline = Disciplines.objects.all()

        dis\_reg = Discipline\_reg.objects.all()

        years = Ac\_years.objects.all()

        context = {

            'users': user,

            'advisor': advisor,

            'groups': groups,

            'student': student,

            'teacher': teacher,

            'discipline': discipline,

            'dis\_reg': dis\_reg,

            'years': years,

            'title': 'Ассистент Советника',

        }

Через контекст, мы можем передавать данные из базы данных напрямую в необходимый шаблон, в нашем случае это **assistant\_s.html**

Вывод информации будет происходить по следующему алгоритму:



(рис. 19)  
Алгоритм вывода студентов

{% for a in advisor %}

                            {% if user.id == a.user\_id %}

                              {% for g in groups %}

                                {% if a.pk == g.advisor\_id %}

                                  {% for s in student %}

                                    {% if s.groups\_id == g.id %}

                                    <tr>

                                      <th *scope*="row"> {{ s.cipher }} </th>

                                      <td>{{ s.groups }}</td>

                                      <td>{{ s.name }}</td>

                                      <td>

                                        <a *href*="tel:{{ s.user.phone\_number }}" *title*="Позвонить" *class*="text-decoration-none"><i *class*="fa-solid fa-phone"></i></a>

                                        {{ s.user.phone\_number }}

                                      </td>

                                      <td>

                                        <a *href*="mailto:{{ s.user.email }}" *title*="Написать" *class*="text-decoration-none"><i *class*="fa-solid fa-envelope"></i></a>

                                        {{ s.user.email }}

                                      </td>

                                    </tr>

                                    {% endif %}

                                  {% endfor %}

                                {% endif %}

                              {% endfor %}

                            {% endif %}

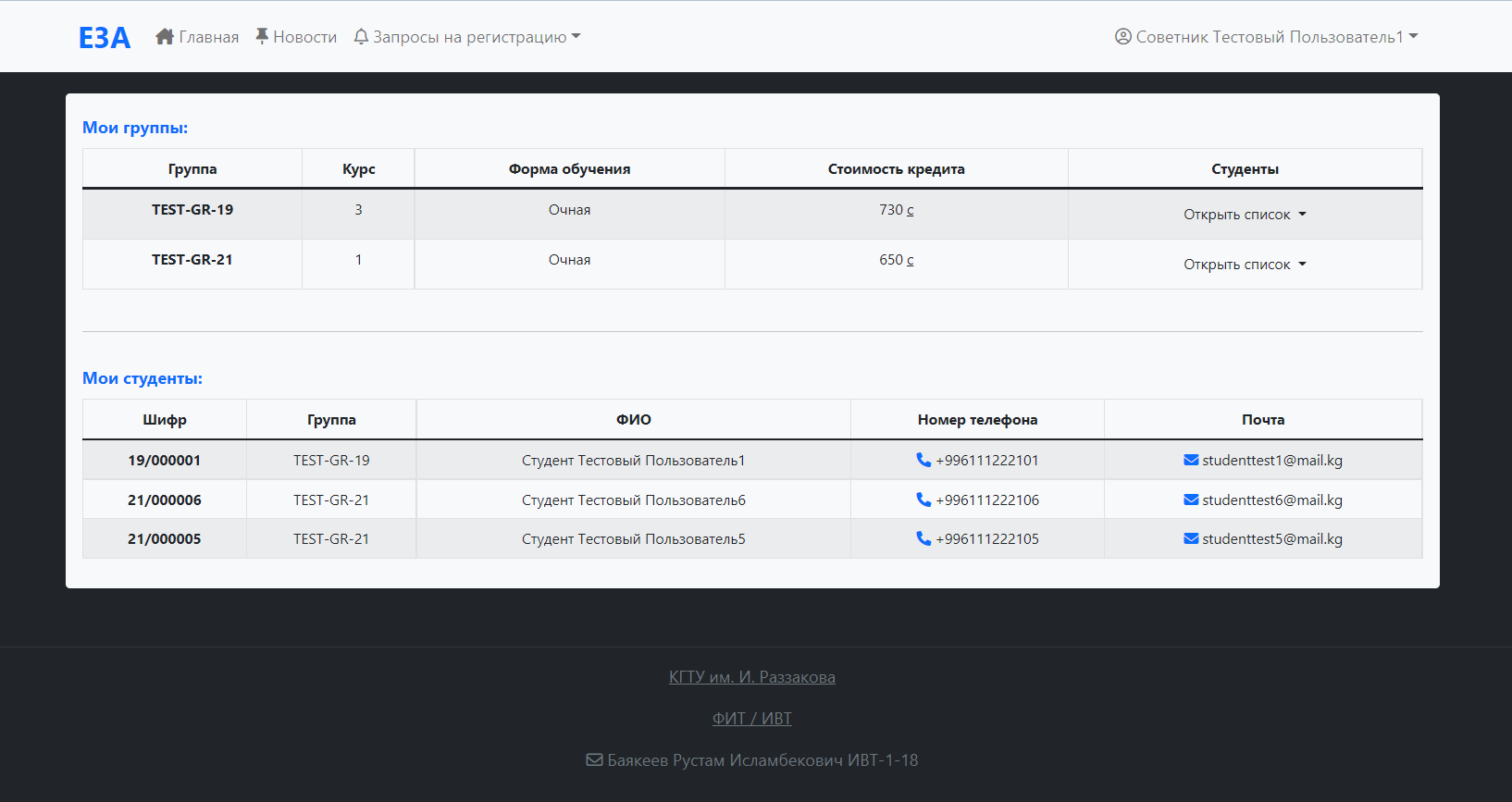
                          {% endfor %}

Так же можно позвонить студенту или написать ему письмо используя выделенные (синего цвета кнопки).

Принцип работы вывода отдельных групп схож с вышеприведенным.

Единственным отличием является то, что вместо студентов, выводятся группы, т.е. проверка не доходит до цикла, связанного со студентами.

Здесь можно увидеть, что возможности как Python, так и фреймворка Django, на высоком, можно даже сказать «магическом» уровне.



(рис. 20)  
Главная страница академического советника.

У студента выведем информацию схожим образом, относительно предыдущего алгоритма.

{% for s in student %}

                    {% if user.id == s.user\_id %}

                              <table *class*="table border-$gray-500"> <!-- Все стденты -->

                                <tbody>

                                  <tr>

                                    <th>ФИО :</th>

                                    <td> {{ s.name }} </td>

                                  </tr>

                                  <tr>

                                    <th>Группа :</th>

                                    <td>{{ s.groups }}</td>

                                  </tr>

                                  <tr>

                                    <th>Курс :</th>

                                    <td>{{ s.course }}</td>

                                  </tr>

                                  <tr>

                                    <th>Стоимость одного кредита :</th>

                                    <td>{{ s.groups.credit\_price }} <u>с</u></td>

                                  </tr>

                                  <tr>

                                    <th>Кредитов зарегистрированно :</th>

                                    <td>

                                      <span *class*="text-secondary">

                                        {{ cf }}

                                      </span>

                                    / <span *class*="text-success">120</span></td>

                                  </tr>

                                  <tr>

                                    <th>Советник :</th>

                                    <td>

                                      {{ s.advisor }}

                                      <br>

                                      <a *href*="tel:{{ s.advisor.user.phone\_number }}" *title*="Позвонить" *class*="text-decoration-none"><i *class*="fa-solid fa-phone"></i></a>

                                      {{ s.advisor.user.phone\_number }}

                                      <br>

                                      <a *href*="mailto:{{ s.advisor.user.email }}" *title*="Написать" *class*="text-decoration-none"><i *class*="fa-solid fa-envelope"></i></a>

                                      {{ s.advisor.user.email }}

                                    </td>

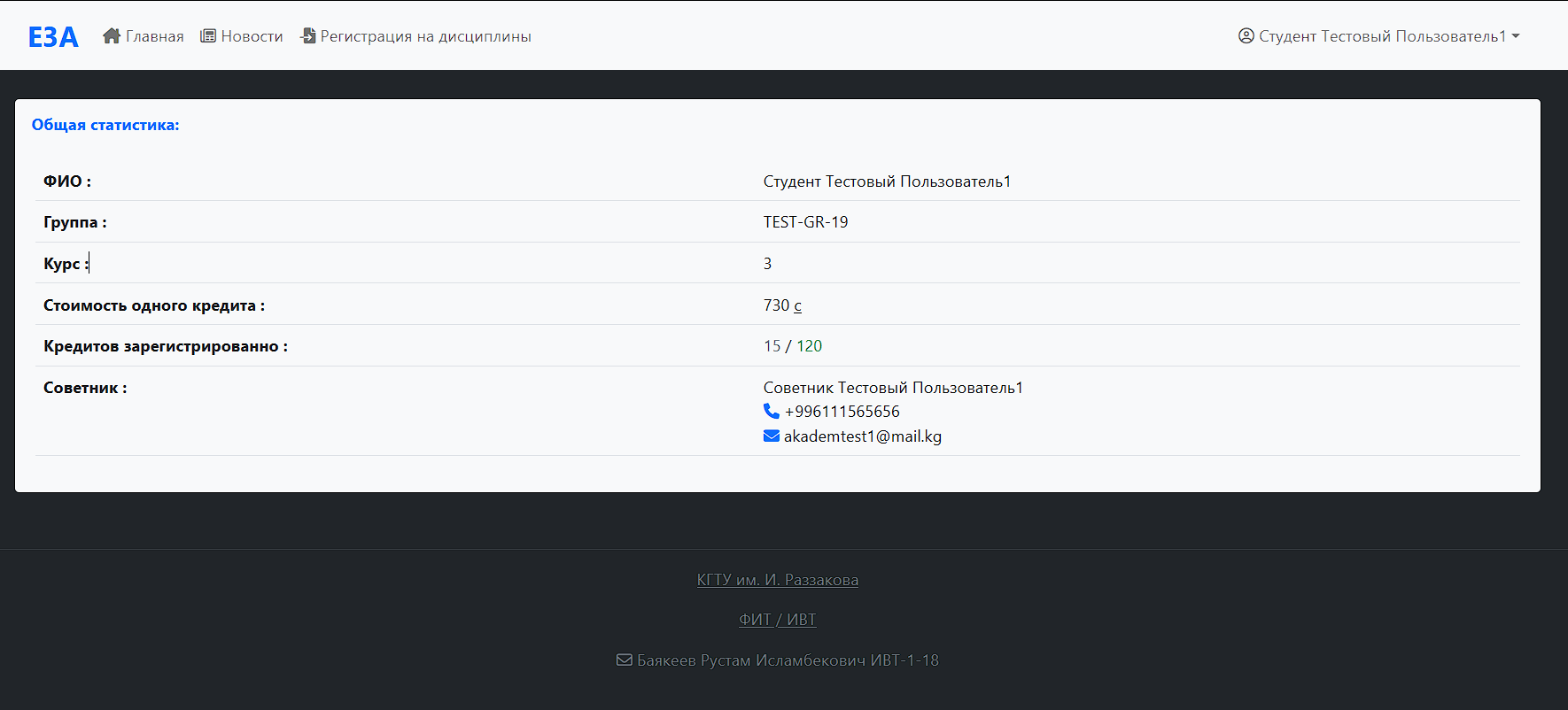
                                  </tr>

                                </tbody>

                              </table>

                    {% endif %}

                  {% endfor %}



(рис. 21)  
Главная страница студента.

## 2.7 Реализация моделей дисциплин.

Перед созданием модели дисциплины нужно создать форму контроля, Факультет и Кафедра уже существуют.

Факультет и кафедра будут использоваться в дальнейшем для определения «Рабочей программы».

class Form\_controls(*models*.Model):

    name = models.CharField(("Название"), *max\_length*=50, *null*=True)

    short\_name = models.CharField(("Короткое название"), *max\_length*=50, *null*=True)

    def \_\_str\_\_(*self*):

        return *self*.short\_name

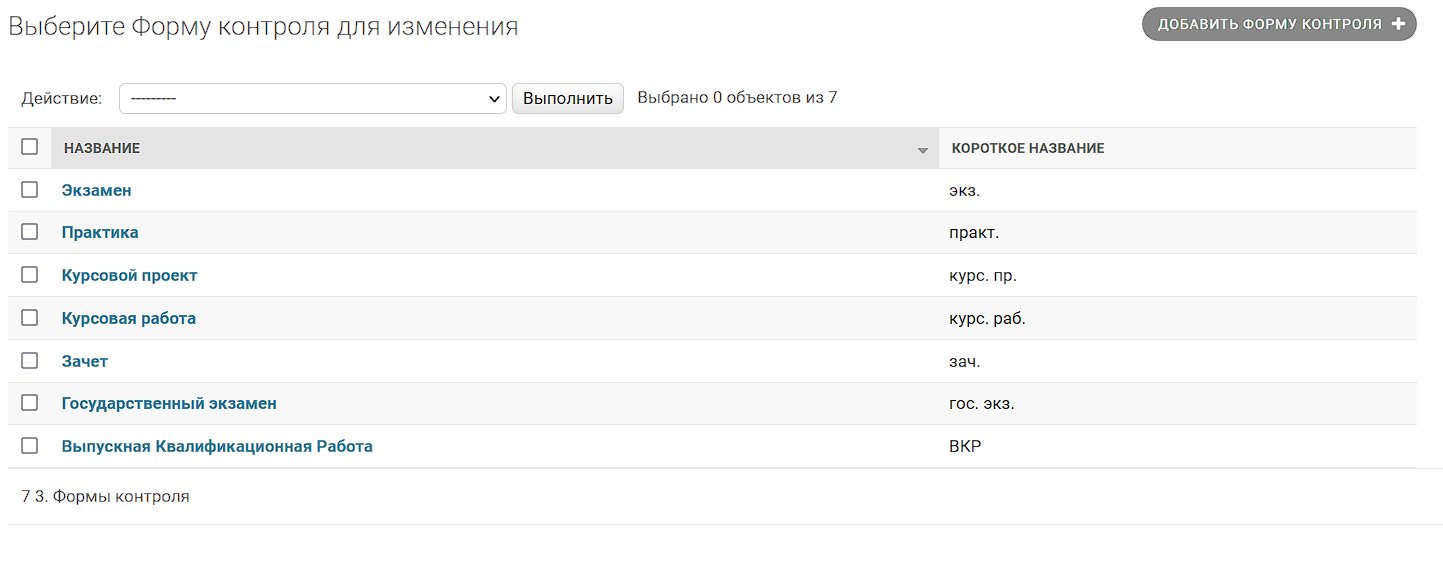
    class Meta:

        verbose\_name = 'Форму контроля'

        verbose\_name\_plural = '3. Формы контроля'

        ordering = ['-name',]

Так же регистрируется в панели администратора. И создадим несколько форм контроля.



(рис. 22)  
Панель управления формами контроля

Создаем модель дисциплины и по внешнему ключу связываем ее с кафедрой, формой онтроля и преподавателем.

class Disciplines(*models*.Model):

    name = models.CharField(("Название"), *max\_length*=150)

    short\_name = models.CharField(("Короткое название"), *max\_length*=75)

    credits = models.IntegerField(("Количество кредитов"), *null*=True)

    hours = models.IntegerField(("Количество часов"), *null*=True)

    department = models.ForeignKey("app\_disciplines.Department", *verbose\_name*=("Отделение"), *on\_delete*=models.PROTECT, *null*=True)

    form\_control = models.ForeignKey("Form\_controls", *verbose\_name*=("Форма контроля"), *on\_delete*=models.PROTECT, *null*=True)

    teacher = models.ForeignKey("app\_users.Teacher", *verbose\_name*=("Преподаватель"), *on\_delete*=models.PROTECT, *null*=True)

    fh = models.BooleanField(("Первое полугодие"), *null*=True)

    sh = models.BooleanField(("Второе полугодие"), *null*=True)

    xtr = models.BooleanField(("Дополнительно"), *null*=True)

    def \_\_str\_\_(*self*):

        return *self*.name

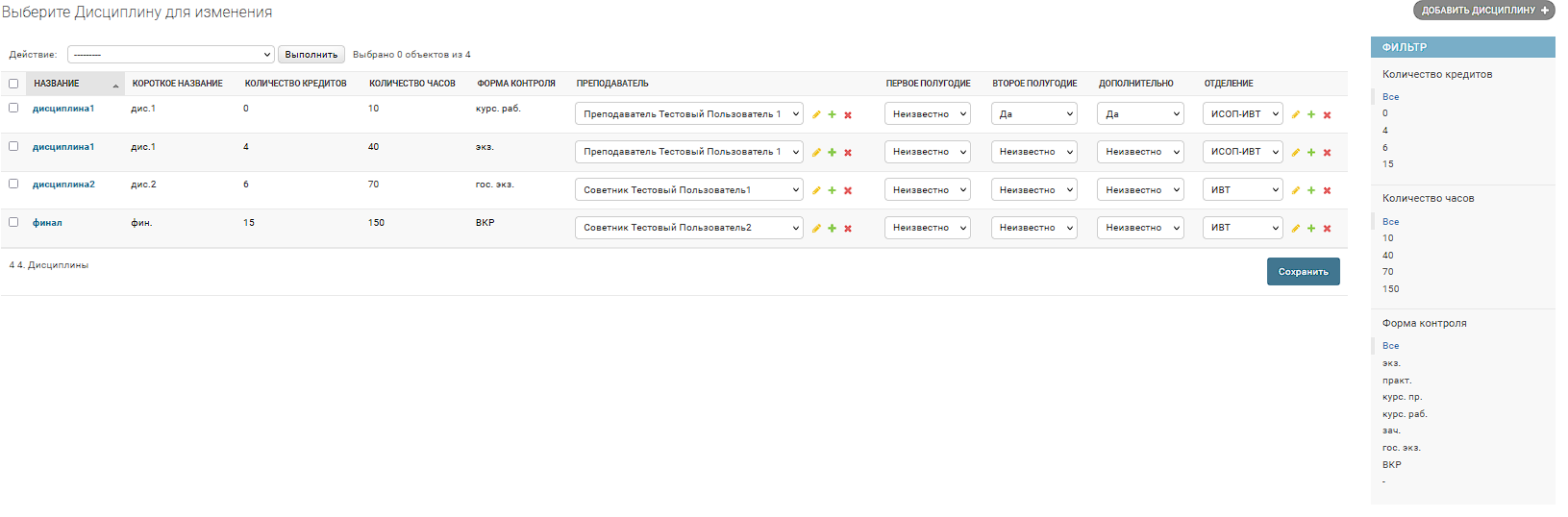
   class Meta:

        verbose\_name = 'Дисциплину'

        verbose\_name\_plural = '4. Дисциплины'

        ordering = ['name', ]

И снова создадим несколько тестовых дисциплин и назначим им преподавателей.



(рис. 23)  
Панель управления дисциплинами

Выведем данные о дисциплинах в интерфейсе преподавателей (академический советник тоже попадает в этот список).

{% for d in discipline %}

                            {% for t in teacher %}

                              {% if t.user\_id == user.id %}

                                {% if d.teacher\_id == t.id %}

                                  <tr>

                                    <th *scope*="row"> {{d.short\_name}} </th>

                                    <td> {{d.credits}} </td>

                                    <td> {{d.hours}} </td>

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

                                    <td> {{d.form\_control}} </td>

                                    <td>

                                      {% if d.fh == true %}

                                        Осень,

                                      {% endif %}

                                      {% if d.sh == true %}

                                        Весна,

                                      {% endif %}

                                      {% if d.xtr == true %}

                                        I,

                                      {% endif %}

                                  </td>

                                  <tr>

                                {% endif %}

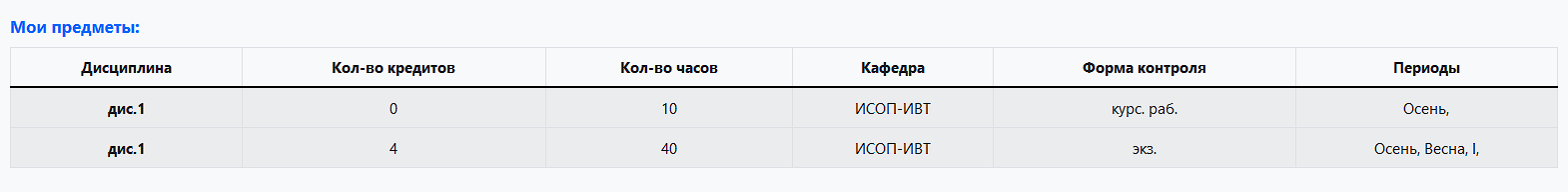
                              {% endif %}

                            {% endfor %}

                          {% endfor %}

Блок схема схожа с последней вышеприведенной схемой на рис. 19.

В результате получаем такую таблицу:



(рис. 24)  
Результаты вывода

## 2.7 Реализация модели и функций регистрации на дисциплины.

Перейдем к созданию регистраций на дисциплины. Для этого создадим модель регистрации на дисциплины.

class Discipline\_reg(*models*.Model):

    discipline = models.ForeignKey("Disciplines", *verbose\_name*=("Название дисциплины"), *related\_name*='groups', *on\_delete*=models.PROTECT)

    student = models.ForeignKey("app\_users.Student", *verbose\_name*=("Шифр студента"), *related\_name*='student', *on\_delete*=models.PROTECT)

    date\_of\_reg = models.DateField(("Дата регистрации"), *auto\_now*=False, *auto\_now\_add*=True)

    conf = models.BooleanField(("Подтверждаю"), *default*=False)

    abon = models.BooleanField(("Отказываю"), *default*=False)

    send = models.BooleanField(("Отправить советнику"), *default*=False)

    academ\_c = models.BooleanField(("Подтверждение советника"), *default*=False)

    academ\_a = models.BooleanField(("Отказ советника"), *default*=False)

    hide = models.BooleanField(("Скрыть результат"), *default*=False)

class Meta:

        verbose\_name = 'Регистрацию на предмет'

        verbose\_name\_plural = '5. Регистрации на предметы'

        ordering = ['discipline', 'date\_of\_reg']

Создаем представление в панели администратора, добавляем несколько дисциплин, и начинаем работу с формой.

Первым делом регистрация на предметы нужна для студентов.

Создадим новое представление **dis\_reg,** и начнем делать обработку запроса.

Создадим форму для создания запросов на регистрацию в **app\_disciplines/forms.py**.

class DisciplineRegFrom(ModelForm):

    class Meta:

        model = Discipline\_reg

        fields = ('discipline', 'student')

        success\_url = reverse\_lazy('dis')

В **app\_disciplines/views.py** сделаем обработчик действия.

Т.к. нам нужно чтобы студент выбирался автоматически, а не мог выбрать кого угодно, сделаем следующее.

Найдем ID студента

def find\_stud():

        for s in student:

            if s.user\_id == *request*.user.id:

                return s.pk

Добавим ID в поле Student.

        initaial\_data = {

            'student': find\_stud,

        }

Создадим обработчик события, по методу POST.

      submitted = False

if *request*.method == "POST":

form = DisciplineRegFrom(*request*.POST, *initial*=initaial\_data)

                if form.is\_valid():

                    form.save()

                    return HttpResponseRedirect('/disciplines/?submitted=true')

        else:

            form = DisciplineRegFrom(*initial*=initaial\_data)

            if 'submitted' in *request*.GET:

                submitted = True

Так же нам нужно будет отфильтровать предметы по факультету:

def disc\_filter():

*list* = []

            for s in student:

                if s.user\_id == *request*.user.id:

                    for g in groups:

                        if s.groups\_id == g.id:

                            for d in dis:

                                if d.department\_id == g.department\_id:

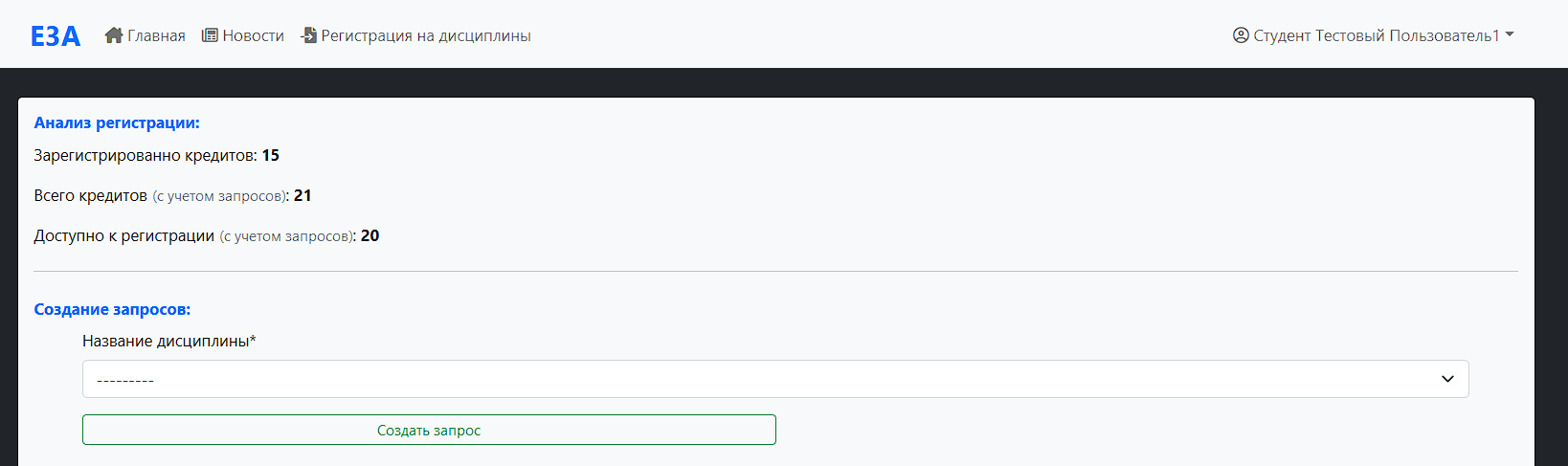
                                    return d.department\_id

Скроем форму студента, и укажем значение для фильтра предметов.

 form.fields['student'].widget = forms.HiddenInput()

        form.fields['discipline'].queryset = Disciplines.objects.filter(*department\_id*=disc\_filter())

Таким образом студенту доступны только предметы его кафедры, а сами данные студента уже будут записаны.



(рис. 25)  
Добавление запросов на предметы.

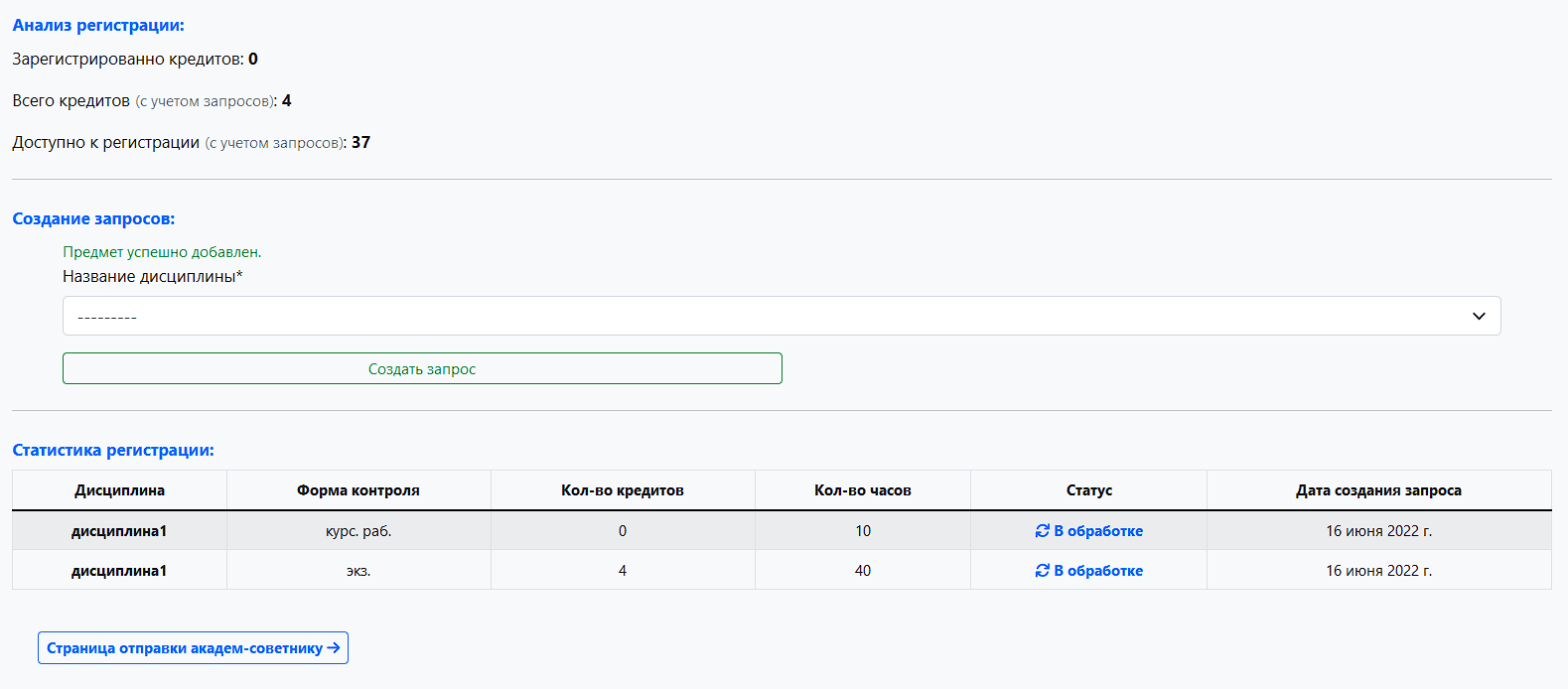
Выведем список, который можно отправлять преподавателю.

if 'academ' in *request*.POST:

                academ\_list = *request*.POST.getlist('send')

                for snd in academ\_list:

                    dis\_reg.filter(*pk*=int(snd)).update(*send*=True)



(рис. 26)  
Добавление запросов на предметы.

Создадим выбор в принятом запросе преподавателю.

if *request*.method == "POST":

            conf\_list = *request*.POST.getlist('cnf')

            for c in conf\_list:

                dis\_reg.filter(*pk*=int(c)).update(*conf*=True)

            abon\_list = *request*.POST.getlist('abn')

            for a in abon\_list:

                dis\_reg.filter(*pk*=int(a)).update(*abon*=True)

            return redirect('dis')

Теперь в html файле обозначим имена соберем токен и откроем все это внутри формы

<div *class*="form-check">

<input *class*="form-check-input" *type*="checkbox" *value*="{{dr.id}}" *name*="cnf"><span *class*="text-success fw-bold">Подтвердить</span>

</div>

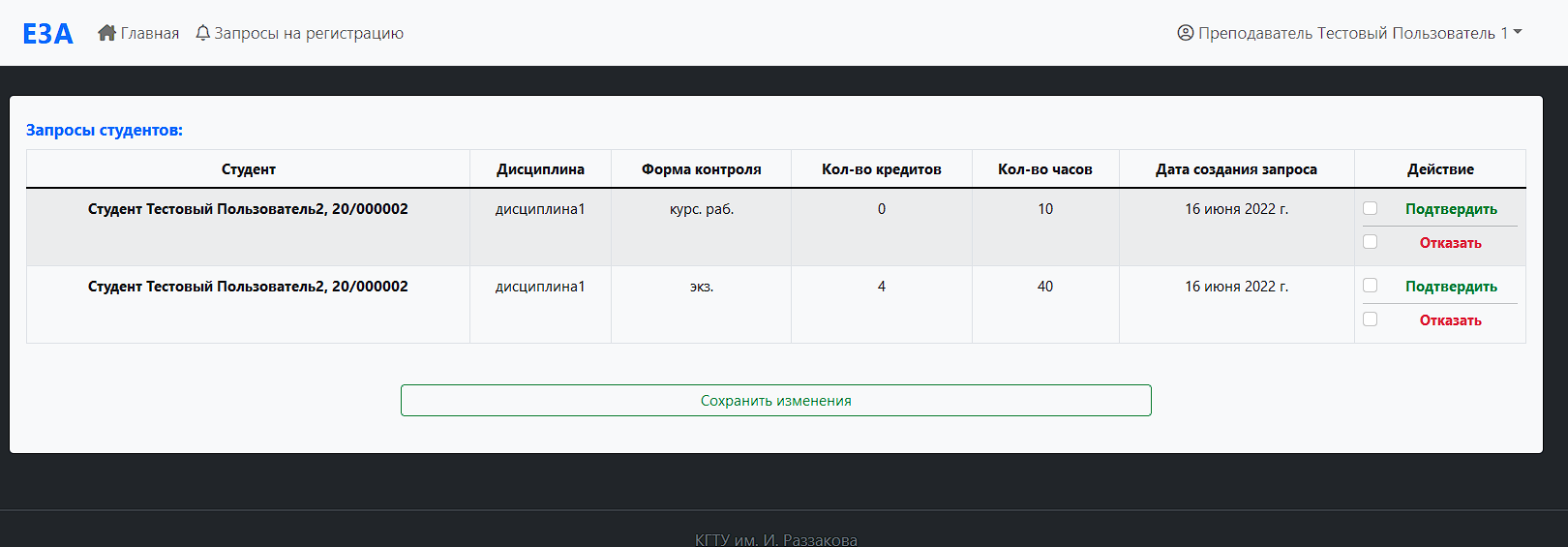
<hr *class*="mt-1 mb-1">

<div *class*="form-check">

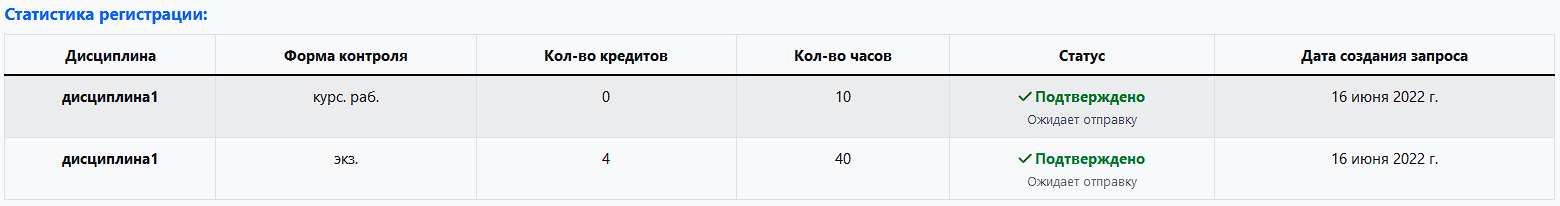
<input *class*="form-check-input" *type*="checkbox" *value*="{{dr.id}}" *name*="abn"><span *class*="text-danger fw-bold">Отказать</span>

</div>

 <button *class*="btn w-50 btn-sm btn-outline-success m-4" *type*="submit">Сохранить изменения</button>



(рис. 27)  
Преподавателю приходят запросы, и он может на них отвечать.



(рис. 28)  
Результаты меняются.

После получения результатов результаты должны подтвердить советники.

if *request*.method == "POST":

            if 'academ' in *request*.POST:

                academ\_list = *request*.POST.getlist('send')

                for snd in academ\_list:

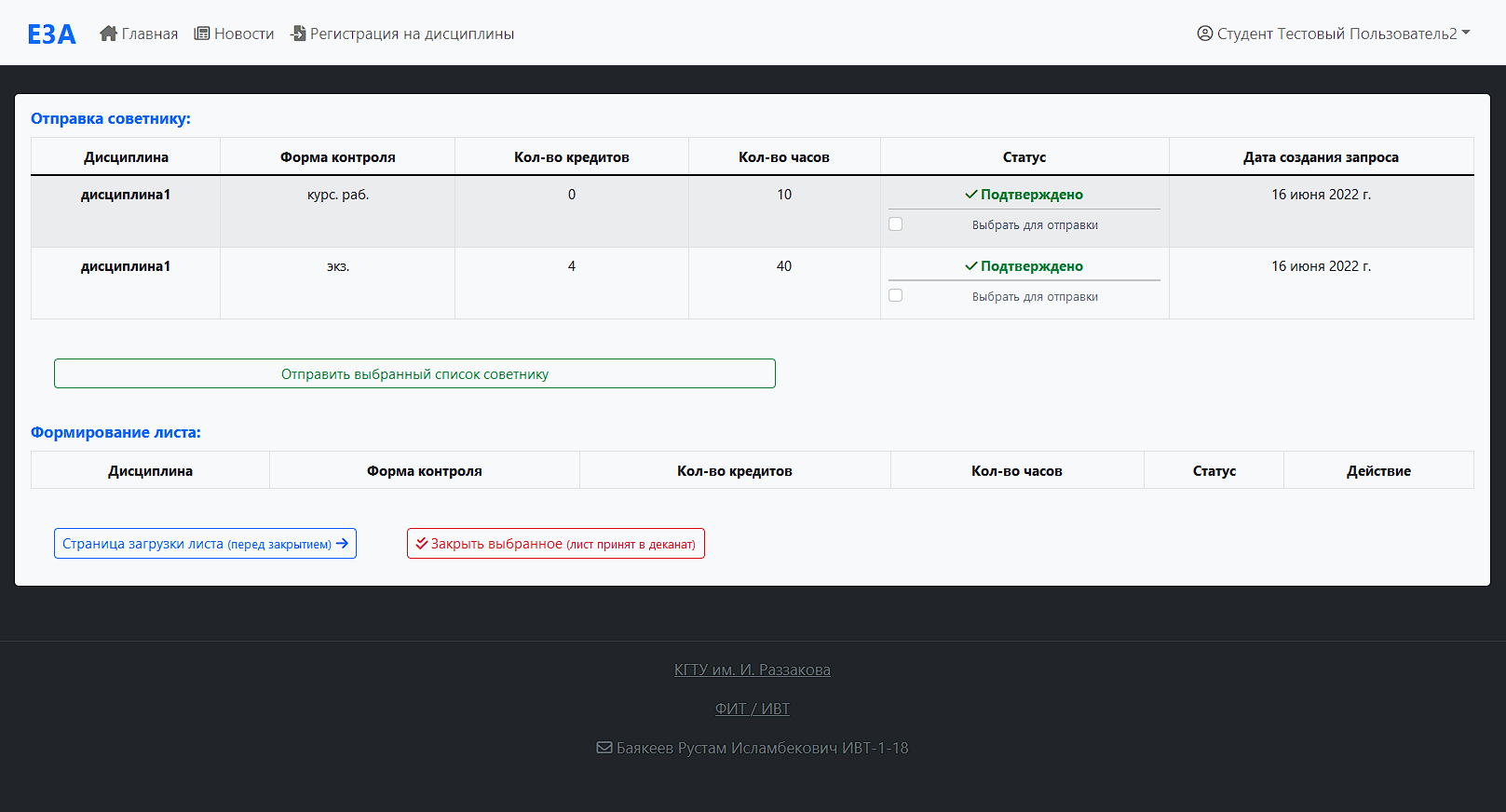
                    dis\_reg.filter(*pk*=int(snd)).update(*send*=True)

            elif 'clear' in *request*.POST:

                clear\_list = *request*.POST.getlist('hide')

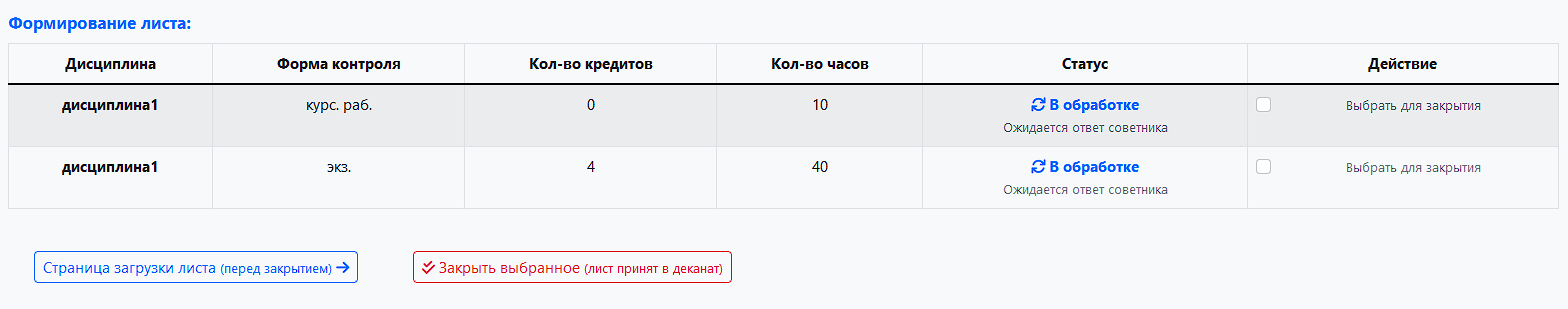
                for clr in clear\_list:

                    dis\_reg.filter(*pk*=int(clr)).update(*hide*=True)



(рис. 29)  
Результаты меняются.

После отправки советнику статус меняется на «В обработке»



(рис. 30)  
Результаты меняются.

if *request*.method == "POST":

            conf\_list = *request*.POST.getlist('cnf')

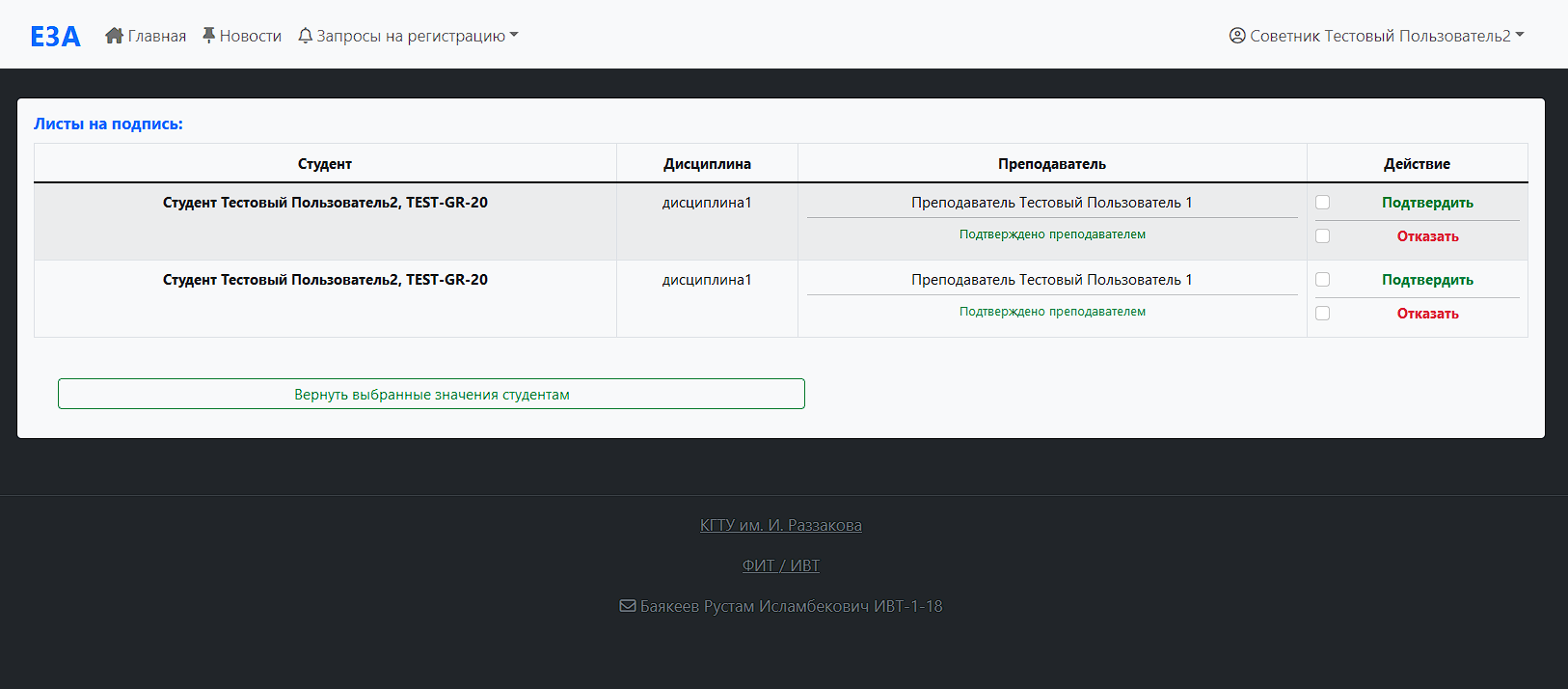
            for c in conf\_list:

                dis\_reg.filter(*pk*=int(c)).update(*academ\_c*=True)

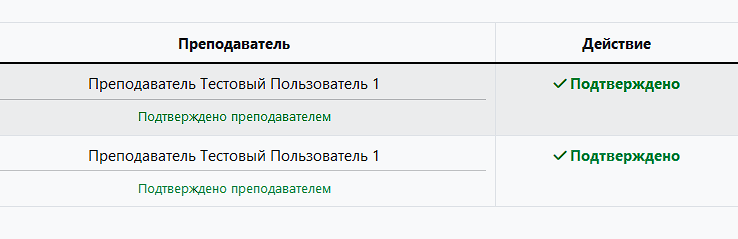
            abon\_list = *request*.POST.getlist('abn')

            for a in abon\_list:

                dis\_reg.filter(*pk*=int(a)).update(*academ\_b*=True)

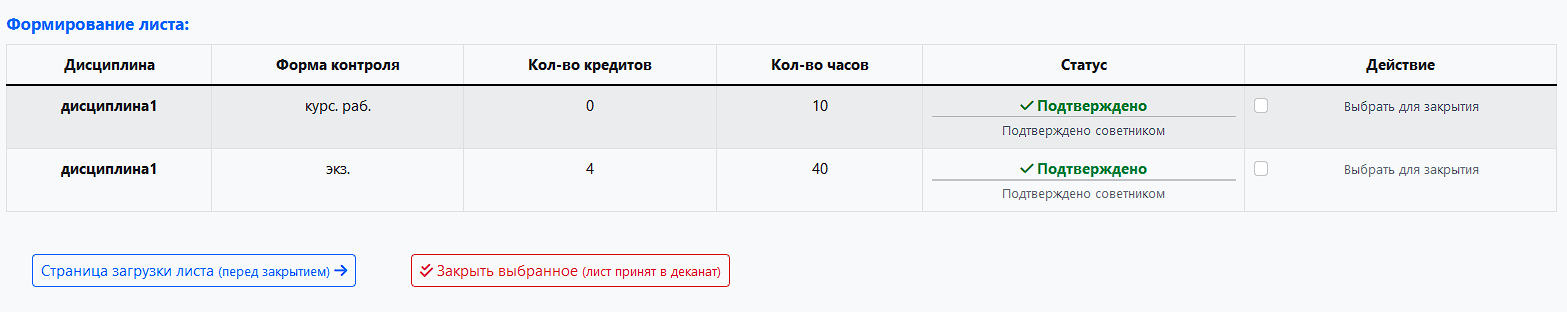


(рис. 31)   
Запросы поступили академическому советнику.



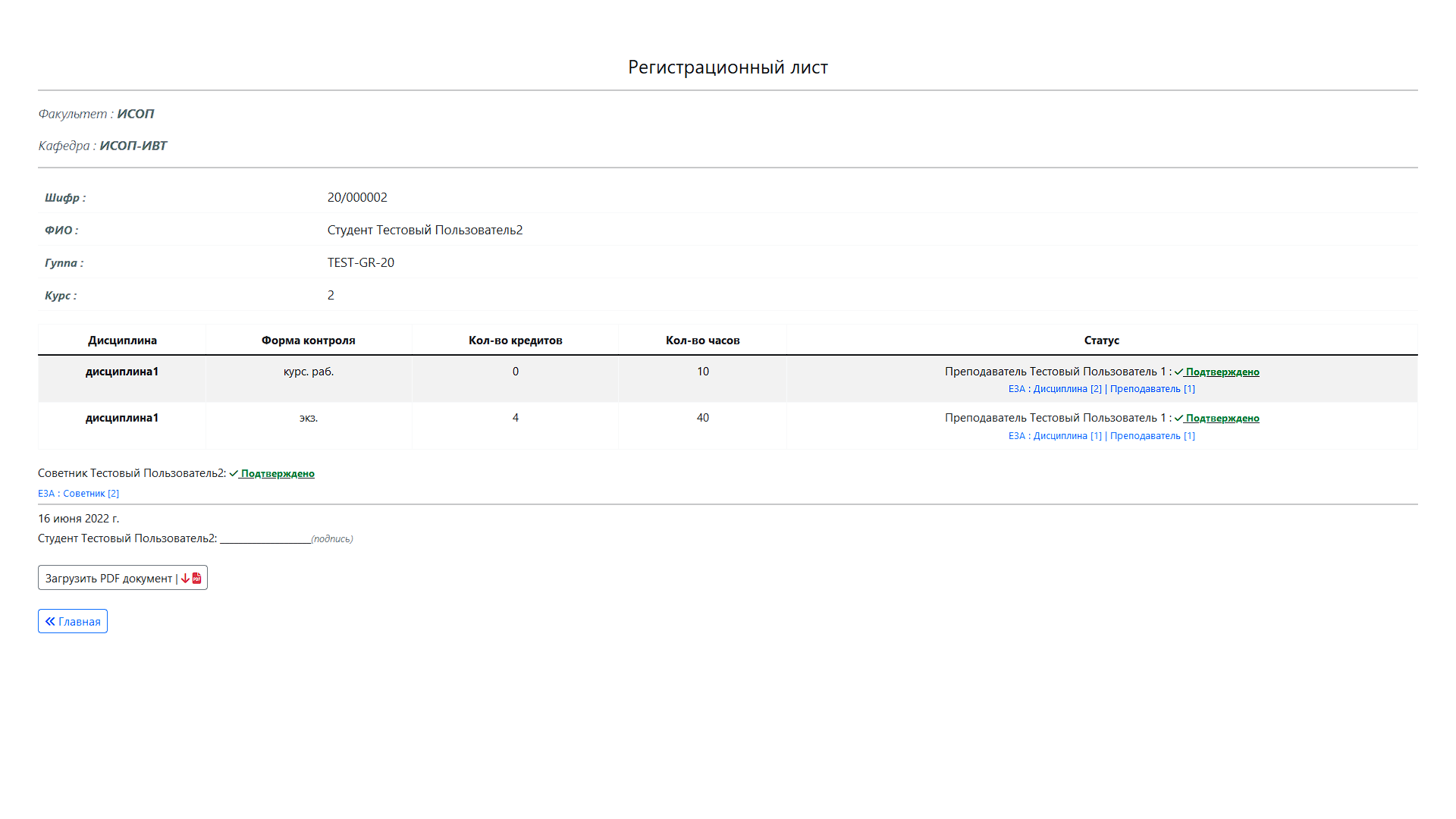
(рис. 32)   
После отправки.

Тем временем студенту уже пришли новые значения.



(рис. 33)   
Подтверждено.

И можно открыть регистрационный слит.



(рис. 33)   
Лист.

Подтвержденные предметы можно будет увидеть у студента на главной странице студента. Немного кода для подсчета статистики

def credit\_counting\_conf():

            all\_data = []

            for dr in dis\_reg:

                for s in student:

                    if s.user\_id == *request*.user.id:

                        if dr.student\_id == s.id and dr.academ\_c == True:

                            all\_data.append(dr.discipline.credits)

            count = 0

            for a in all\_data:

                count = count + a

            return count

        def credit\_counting\_req():

            all\_data\_r = []

            for dr in dis\_reg:

                for s in student:

                    if s.user\_id == *request*.user.id:

                        if dr.student\_id == s.id and dr.abon != True:

                            all\_data\_r.append(dr.discipline.credits)

            count\_r = 0

            for a in all\_data\_r:

                count\_r = count\_r + a

            return count\_r

        def credit\_ost():

            all\_data\_r = []

            for dr in dis\_reg:

                for s in student:

                    if s.user\_id == *request*.user.id:

                        if dr.student\_id == s.id:

                            all\_data\_r.append(dr.discipline.credits)

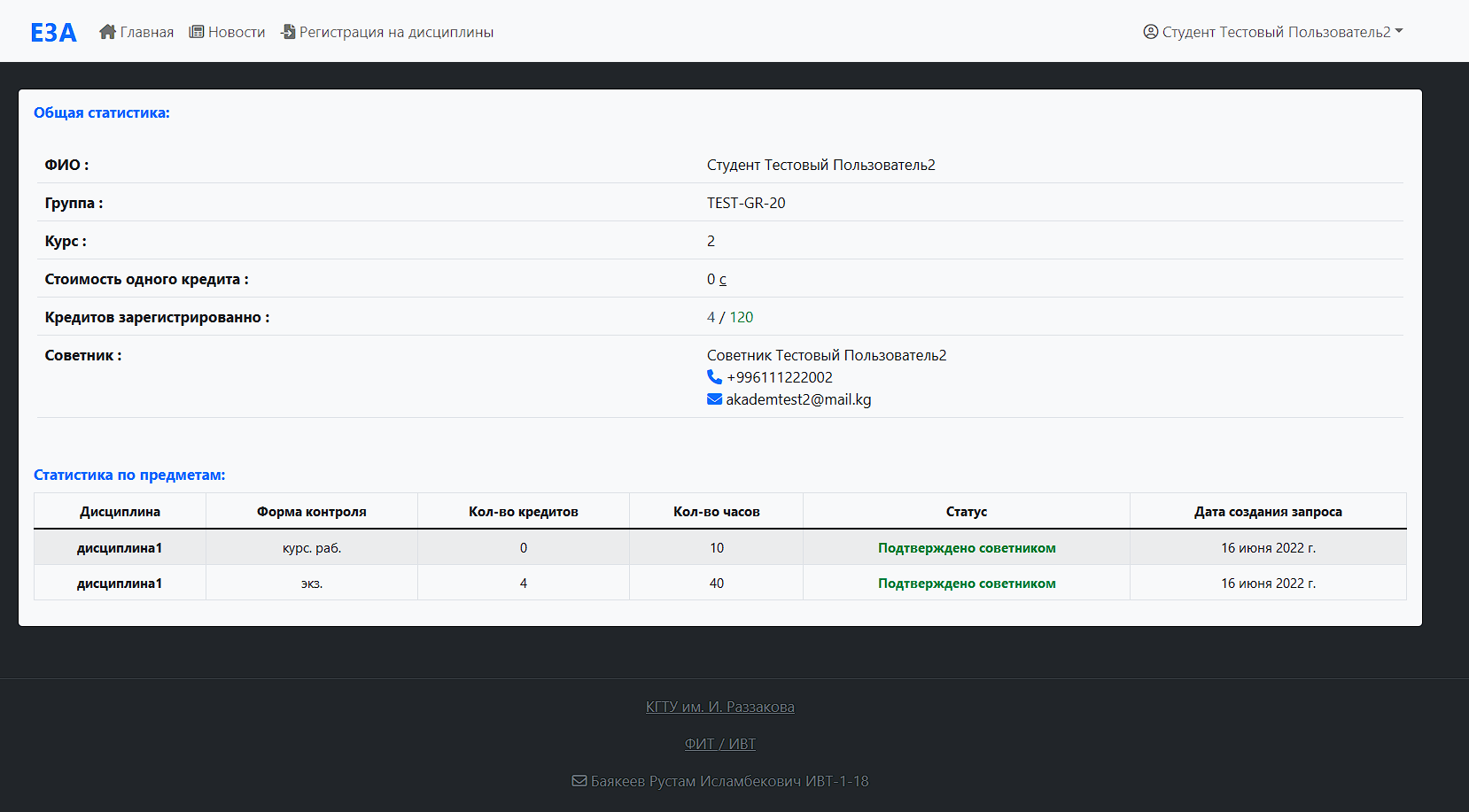
            count\_r = 0

            for a in all\_data\_r:

                count\_r = count\_r + a

            count\_r = 41 - count\_r

            return count\_r



## 2.7 Реализация новостей.