Оглавление

[Введение 4](#_Toc106951668)

[Глава 1: Обзор и анализ 7](#_Toc106951669)

[1.1. Анализ области темы 7](#_Toc106951670)

[1.2. Обзор существующих систем. 9](#_Toc106951671)

[1.2.1. Обзор Google Classroom 9](#_Toc106951672)

[1.2.2. Обзор Moodle 11](#_Toc106951673)

[1.2.3. Обзор WebTutor 12](#_Toc106951674)

[1.2.4. Выводы 13](#_Toc106951675)

[1.3. Функции системы 14](#_Toc106951676)

[1.3.1. Основные функции. 14](#_Toc106951677)

[1.3.2. Подсистемы и их функции 15](#_Toc106951678)

[1.4. Постановка задач 16](#_Toc106951679)

[1.4.1. Основной алгоритм системы 17](#_Toc106951680)

[Глава 2: Разработка Электронного Ассистента Академического Советника 18](#_Toc106951681)

[2.1 Модификация стандартной установки 18](#_Toc106951682)

[2.2 Реализация разбития пользователей на группы. 19](#_Toc106951683)

[2.3 Реализация входа в систему. 22](#_Toc106951684)

[2.4 Реализация проверки типа пользователя при авторизации. 24](#_Toc106951685)

[2.5 Реализация дополнительных моделей пользователей и групп со студентами. 27](#_Toc106951686)

[2.6 Реализация дополнительных моделей пользователей и групп со студентами. 30](#_Toc106951687)

[2.7 Реализация моделей дисциплин. 33](#_Toc106951688)

[2.8 Реализация модели и функций регистрации на дисциплины. 34](#_Toc106951689)

[2.9 Реализация новостей. 40](#_Toc106951690)

[Глава 3: Руководство 43](#_Toc106951691)

[3.1. Руководство пользователя 43](#_Toc106951692)

[3.2. Руководство разработчику 52](#_Toc106951693)

[Заключение 55](#_Toc106951694)

[Список литературы 56](#_Toc106951695)

# 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, речь не идет 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 данной 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 данной работы является повысить уровень коммуникации между Студентом и Академическим советником.

Исходя из вышеперечисленного можно сделать какие-то выводы. Данную 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 данная работа будет представлена и разрабатываться как общая система состоящая из отдельных подсистем, это может дать большое преимущество в будущем, особенно в том случае если эту систему захотят дополнить новым элементом (подсистемой).  
Системы бывают разных видов, встроенные, программно-насыщенные, вычислительно-ориентированные, в данном 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 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 данном случае было выбрано реальное время.
* В качестве оборудования обычно выступает клиент-серверная архитектура.
* В качестве пользователей выступают руководители и персонал либо же подчиненные, или под ответственные группы.

Часть об «Ассистенте». Если рассматривать данное понятие, не опираясь на слово «Электронный», то можно увидеть 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 данной ВКР, но все же это Электронный Ассистент, поэтому нужно понять какие функции, и вообще в целом, что из себя должен представлять Электронный Ассистент.

Если обратиться к поисковой системе Google, она выдаст в первую очередь информацию о «Виртуальном ассистенте», и почитав статьи о виртуальных ассистентах сразу же станет ясно, 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 данной 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, получено такое определению как «Виртуальный цифровой помощник» его так же упоминают как VDA (от англ. Virtual Digital Assistant). Что же представляет собой VDA? Как говорит 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 данной работе частично, но отмечу что все же, эти определения в какой-то степени подходят для описания данной 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 необходимо 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 Высшего Учебного Заведения.

# Глава 1: Обзор и анализ

## Анализ области темы

Как было описано в ведении, тема данной 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 также 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 позже.

В заключении необходимо 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 выбран иной путь и не будет обращено на такие мнения внимания, т.к. это не то, что нужно, все за развитие IT-индустрии, и далее в данном изложении текст будет опираться 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. данная работа достаточно узконаправленная, сравнение будет проводиться с подсистемами систем по организации учебного процесса.

### Обзор Google Classroom

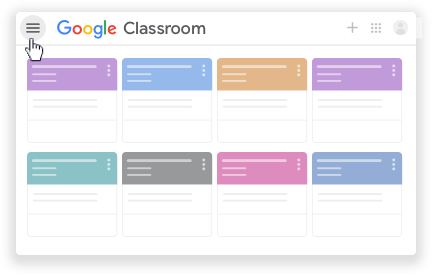


Рис. 1.1  
Главное окно Google Classroom

Начало будет 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 больше интересны нужные 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.

- Уведомления приходят во время публикации задания, и снова это не совсем то, что нужно, но тем не менее можно заметить, что и описывалось 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. данная работа, узконаправленная, будет нужна информация о всех подконтрольных группах.

### Обзор Moodle

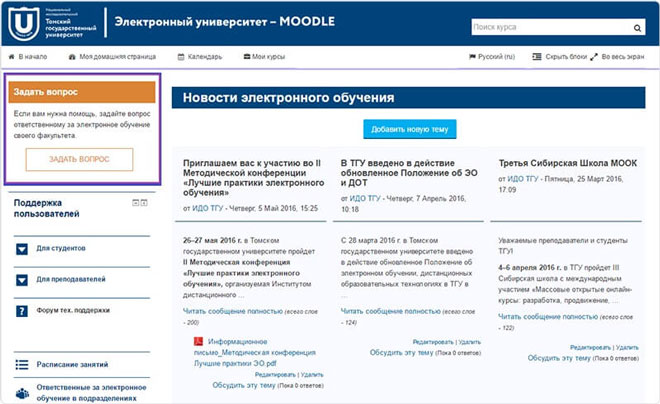


Рис. 1.2  
Главная страница интерфейса Moodle

Далее будет рассмотрена более продвинутая система 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 можно считать 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.

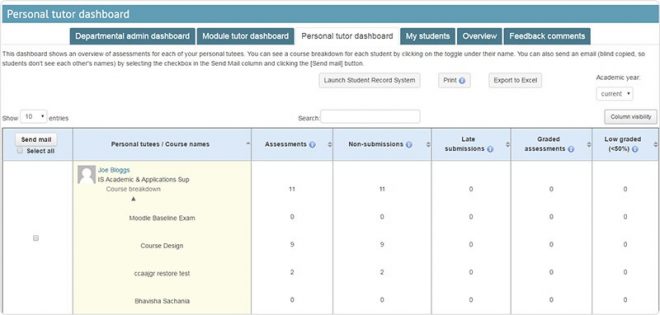


Рис. 1.3  
Подсистема отчетов успеваемости Moodle

### Обзор WebTutor

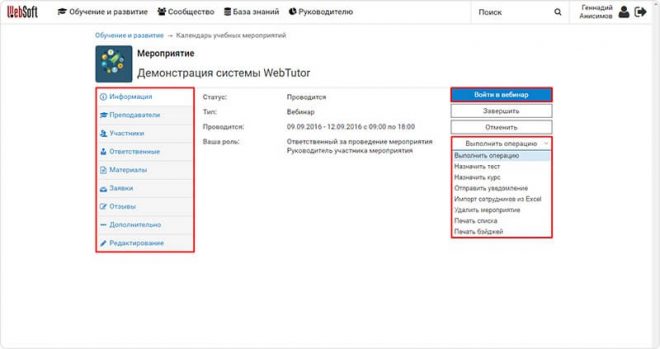


Рис. 1.4  
Главное страница интерфейса WebTutor

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 видна 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 данная 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. можно не считать минусом.

## Функции системы

### Основные функции.

* Сбор имеющихся данных.

В качестве примера представим, что уже имеется 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 данной ВКР, пользователи будут делиться по ролям, в данном 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.

* 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 данном 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 студент имеющий задолженность на втором курсе, будет иметь недостающее количество кредитов, соответственно система этот факт обнаружит, и покажет студенту какой у него недобор кредитов.

С основными функциями системы и ее подсистем было пройдено ознакомление, теперь можно перейти к следующей главе, где будет проходить сравнение со схожими системами.

## Постановка задач

Система будет закрытой, и эту проблему легко решить ограничением доступа к системе посредством входа через личный аккаунт пользователя.

* Создать и добиться грамотного разграничения пользователей.
* Сделать для каждого пользователя свой интерфейс.
* Сделать возможным редактирование профиля, в случае изменения пользовательских данных.

Далее пользователи будут разбиваться на еще более мелкие группы.

* Создать для каждого пользователя свои группы
* Создать для студентов группы
* Создать факультет и кафедру группы
* Создать назначение советника

Далее для реализации на регистрацию на предметы нужно будет:

* Создать предметы
* Создать преподавателя для предмета
* Создать курс для предмета
* Создать регистрации на предметы

Ну и для последнего, чтобы академ. советник мог держать своих студентов в курсе последних событий:

* Создать новостную ленту
* Связать все имеющиеся данные и грамотно построить приложение.

### Основной алгоритм системы

Основной алгоритм системы, должен будет объединить в себе все подсистемы, и обеспечить корректный вывод выходной информации в интерфейсе системы. Ориентировочный основной алгоритм системы показан на рисунке 1.5

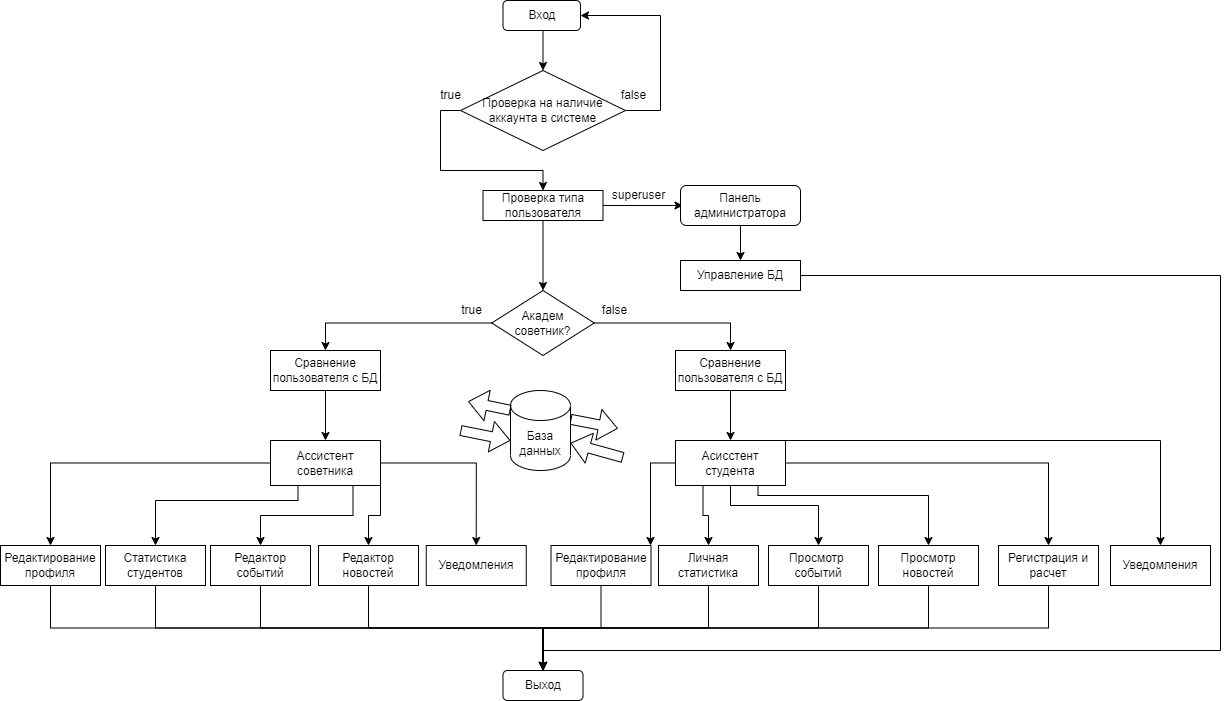


Рис. 1.5  
Основной алгоритм системы

# 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 будет уделяться.

В этом разделе будет больше заострено внимание на некоторых отступлениях от стандартной установки.

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

Придется совершить несколько манипуляций.

Для того чтобы подключить к данному проекту **PostgreSQL** в первую очередь придется установить **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**.   
Следующим шагом, потребуется создать базу данных в **PostgreSQL**.  
В запущенном окне **SQLShell (psql).exe** нужно создать пользователя, БД, и назначить 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 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 возможно благодаря грамотному построению структуры базы данных.

В первую очередь, нужно будет определить главную модель пользователя.

Что и будет сделано для этого необходимо создать новое приложение проекта, можно дать ему название **app\_users**, внутри приложения в файле **models.py**, можно создать новую модель и назвать ее **user** она будет наследоваться от стандартной модели пользователей Django.

В этой модели нужно создать 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

Что и было сделано в следующем фрагменте кода:

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']

Также нужно будет создать обработчик для этой модели:

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)

Далее понадобиться произвести миграцию с базой данных PostgreSQL.

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

Чтобы получить доступ к модели нужно будет создать представление в панели администратора. Для этого нужно открыть/создать файл **admin.py** в папке этого же проекта. И в нем описать отображение, так же сделать поля пригодными для использования, что и было сделано в следующем фрагменте кода:

@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, все еще нет доступа к панели администратора. Т.к. все еще нет созданных пользователей.

Чтобы решить эту проблему, нужно открыть консоль и запустить команду файла Django

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

Появится приписанная регистрация пользователя, которая потребует все необходимые данные. Теперь, можно войти в панель администратора, используя ссылку /admin и данные пользователя. И как можно заметить, появилась панель управления пользователям. При нажатии на кнопку добавить, 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.

Для наглядности будет создано несколько тестовых пользователей.

## 2.3 Реализация входа в систему.

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

Будет создан шаблон для входа, для этого в любом из приложений будет создана папка **template,** в ней будет создана папка **registrations**, в ней будет создан файл с названием login.html. В файле **urls.py** нужно указать путь к данному файлу чтобы он мог корректно работать.

Также нужно указать 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** будет создано условие проверяющее авторизован ли пользователь. Если нет, то будет направлен на представление входа, если авторизован, то будет перенаправлен, на другую страницу, которая будет создана 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, можно 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, можно использовать модуль, меняющий стили форм на стили Bootstrap 5.

pip install crispy-bootstrap5

Теперь нужно вызвать форму в шаблоне, используя фильтр **| crispy.**

<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 веб-сайта можно использовать шаблоны. Существует два способа добавления шаблона на сайт в зависимости от потребностей.

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

Для каждого приложения проекта можно создать отдельный каталог шаблонов.

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

## 2.4 Реализация проверки типа пользователя при авторизации.

Так как в 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).

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

Для этого можно создать 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.

Будет начата реализация. Для этого будут созданы шаблоны для каждого пользователя в папку **templates/app\_assistant**, и в файле **view.py** нужно указать пути и поставить условие проверки пользователя, что будет сделано в следующем фрагменте кода:

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

Будет добавлен этот метод в **urls.py**, чтобы могло работать отображение страниц, что и было сделано в следующем фрагменте кода:

urlpatterns = [

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

]

После проверок, становится ясно, что переадресация в зависимости от типа пользователя работает как положено. Так же нужно добавить ссылку на главную страницу в представление **login**:

        else:

                return redirect('assistant/')

        endif

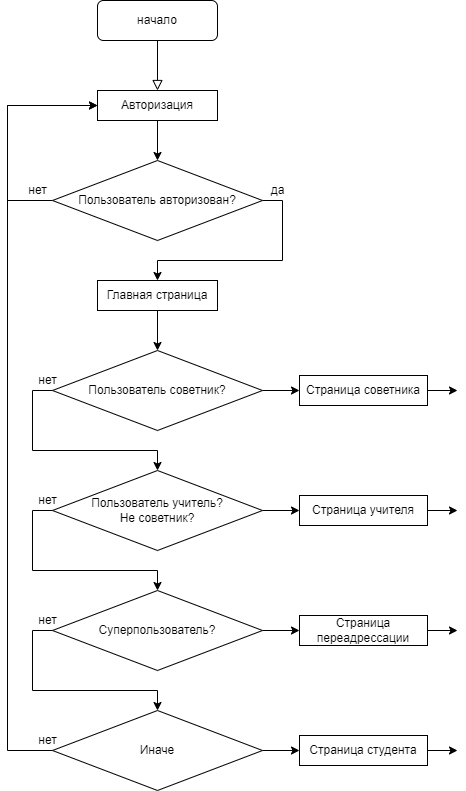


Рис. 2.1  
Алгоритм проверки авторизации

## 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 нужно будет делать много запросов, и полей общей модели просто будет недостаточно.

Для этого в первую очередь будет дополнена 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 будет создана расширяющая 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 система была расширена моделями явно выделяющихся пользователей, можно попробовать создать модель группы для студентов. Но т.к. у групп присутствует иерархия в виде факультетов и кафедр. Для начала нужно создать 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 будет создана форму обучения для групп:

    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, начинается создание модели групп, и соединение 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 будет создана модель пользователя, и соединена 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, будет необходимо добавить отображение модели в панели администратора. После регистрации моделей в панели администратора можно увидеть следующее.

Панели советников и преподавателей схожи, поэтому не будут приведены скриншоты в пример, и просто будет проделано то же самое с панелью преподавателей. Советники будет сделаны преподавателями.

Как можно заметить в вышеприведенных формах, связи в PostgreSQL под управлением Django отрабатывают отлично, как часы.

Будут созданы тестовые группы. Будут назначены для них отделения, и нужно привязать группу к пользователю.

Так же нужно указать к какому отделению привязана группа.

Осталось добавить несколько тестовых студентов.

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

Исходя из созданных моделей. И добавленных данных, можно начать построение главной страницы 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',

        }

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

Во-первых, подключить модели.

И будет добавлен код к имеющемуся условию проверки на академического советника. Через контекст, можно передавать данные из базы данных напрямую в необходимый шаблон, в данном случае это **assistant\_s.html**

Вывод информации будет происходить по алгоритму на рисунке 2.2:



Рис. 2.2  
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 будет 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.

У студента будет выведена информацию схожим образом, относительно предыдущего алгоритма.

## 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 будет создано несколько форм контроля.

Создание модели дисциплины и по внешнему ключу, построение связи с кафедрой, формой контроля 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 будет создано несколько тестовых дисциплин и назначены им преподаватели.

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

Блок схема схожа с последней вышеприведенной схемой на рисунке 2.2.

## 2.8 Реализация модели и функций регистрации на дисциплины.

Далее, к созданию регистраций на дисциплины. Для этого будет создана 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']

Было создано представление в панели администратора, будет добавлено несколько дисциплин, и можно начать работу с формой.

Первым делом регистрация на предметы нужна для студентов.

Будет создано новое представление **dis\_reg,** и будет начата работа над обработкой запроса.

Будет создана форма 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** был сделан обработчик действия.

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

Найти ID студента:

def find\_stud():

        for s in student:

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

                return s.pk

Добавить ID в поле Student:

        initaial\_data = {

            'student': find\_stud,

        }

Создать 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 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

Нужно скрыть форму выбора студента, и указать 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.

**Анализ регистрации,** оценивает и указывает причину невозможности регистрации.

Нужно вывести 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)

Будет создан выбор в принятом запросе преподавателю:

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

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

            for c in conf\_list:

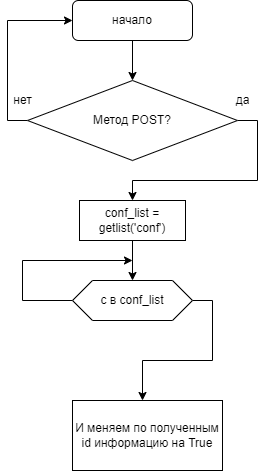
                dis\_reg.filter(*pk*=int(c)).update(*conf*=True)

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

            for a in abon\_list:

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

            return redirect('dis')

  
Рис. 2.3  
Алгоритм отправки/Возврата предметов.

Теперь в html файле будут обозначены имена, нужно будет собрать токен и открыть 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>

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)

После отправки советнику статус меняется на «В обработке»

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)

Тем временем студенту уже пришли новые значения.

И можно открыть регистрационный слит.

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

## 2.9 Реализация новостей.

Будет создана модель новости.

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

    title = models.CharField(("Заголовок"), *max\_length*=50)

    text = models.TextField(("Текст новости"))

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

    advisor = models.ForeignKey("app\_users.Advisor", *verbose\_name*=("Советник"), *on\_delete*=models.PROTECT)

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

    def \_\_str\_\_(*self*):

        return *self*.title

    class Meta:

        verbose\_name = ("Новость")

        verbose\_name\_plural = ("Новости")

        ordering = ['date\_of\_pub',]

Далее, как и в предыдущие разы, будет создано представление в панели администратора.

После чего будет добавлено несколько тестовых новостей.

@admin.register(News)

class CreditsCountAdmin(*admin*.ModelAdmin):

    model = News

    list\_display = ('title', 'text', 'advisor', 'date\_of\_pub',)

    search\_field = ('title', 'advisor',)

Будет создана форма и отображение формы на странице советника

class NewsForm(ModelForm):

    class Meta:

        model = News

        fields = ('title', 'text', 'advisor')

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

Далее будут созданы данные для отображены и публикации новости, опубликованной советником.

    def find\_advisor():

        for a in advisor:

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

                return a.pk

    if *request*.user.is\_advisor == True:

        initaial\_data = {

            'advisor': find\_advisor,

        }

        submitted = False

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

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

            if form.is\_valid():

                form.save()

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

        else:

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

            if 'submitted' in *request*.GET:

                submitted = True

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

        context = {

            'form': form,

            'users': user,

            'advisor': advisor,

            'groups': groups,

            'student': student,

            'news': news,

            'title':'Управление новостями',

        }

        return render(*request*, 'app\_news/news\_a.html', *context*=context)

В результате проектирования и моделирования всех таблиц в базе данных, получилась следующая структурная схема базы данных с присутствующими в ней связями между таблицами, это показано на рисунке 2.4:

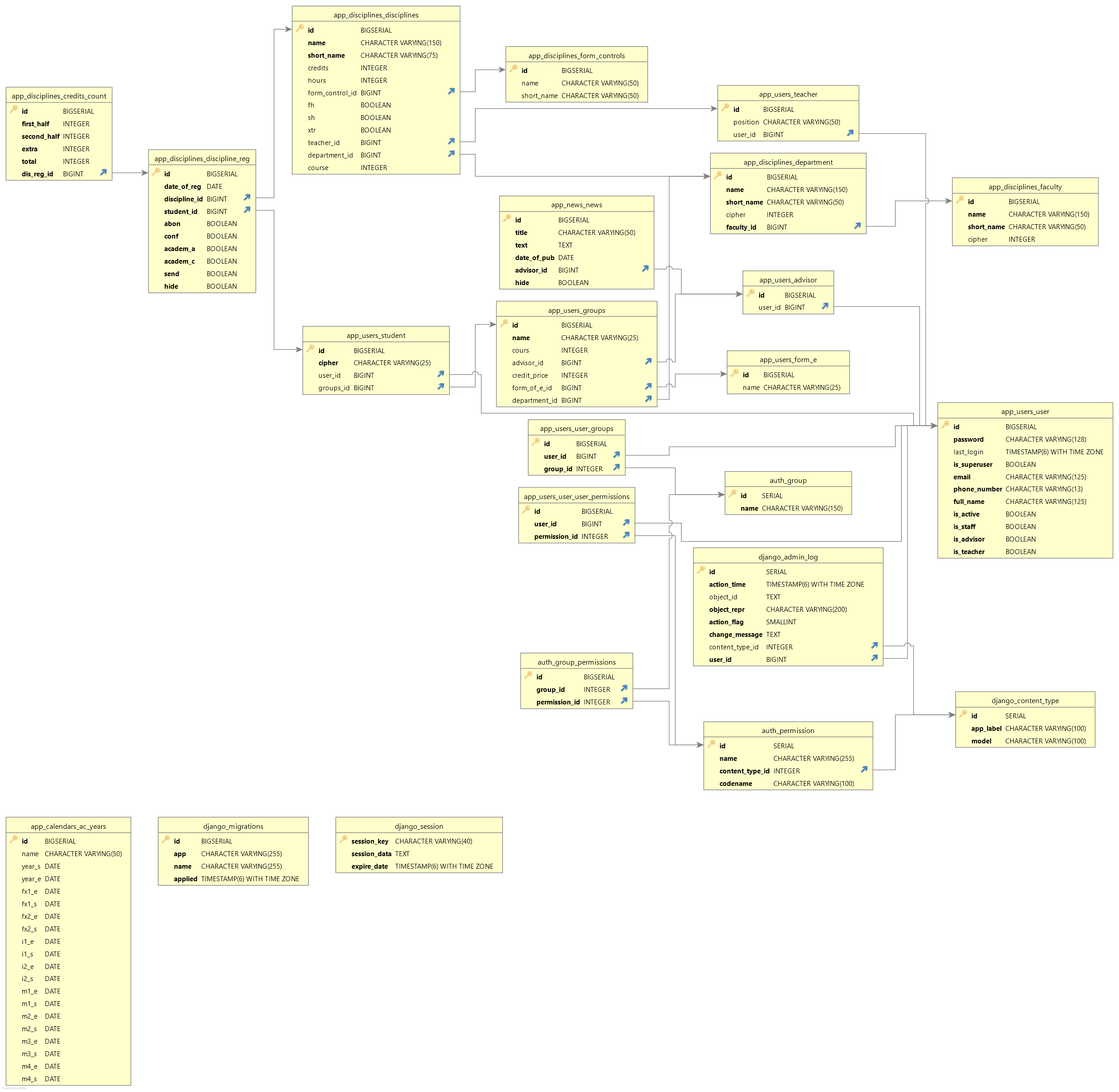
****

Рис. 2.4  
Структурная схема базы данных

# Глава 3: Руководство

### Руководство пользователя

Данное руководство предназначено для людей, которые разбираются в система AVN и в принципе в том, как устроена система регистраций и дисциплин в КГТУ им. И. Раззакова. Система E3A ориентирована на взаимосвязи между академическими советниками и студентами, и преподавателями.

Первоначальный пользователь и пользователь, который будет настраивать и добавлять основные данные в систему это будет Администратор.

Поэтому описание начнется с интерфейса **Администратора**.

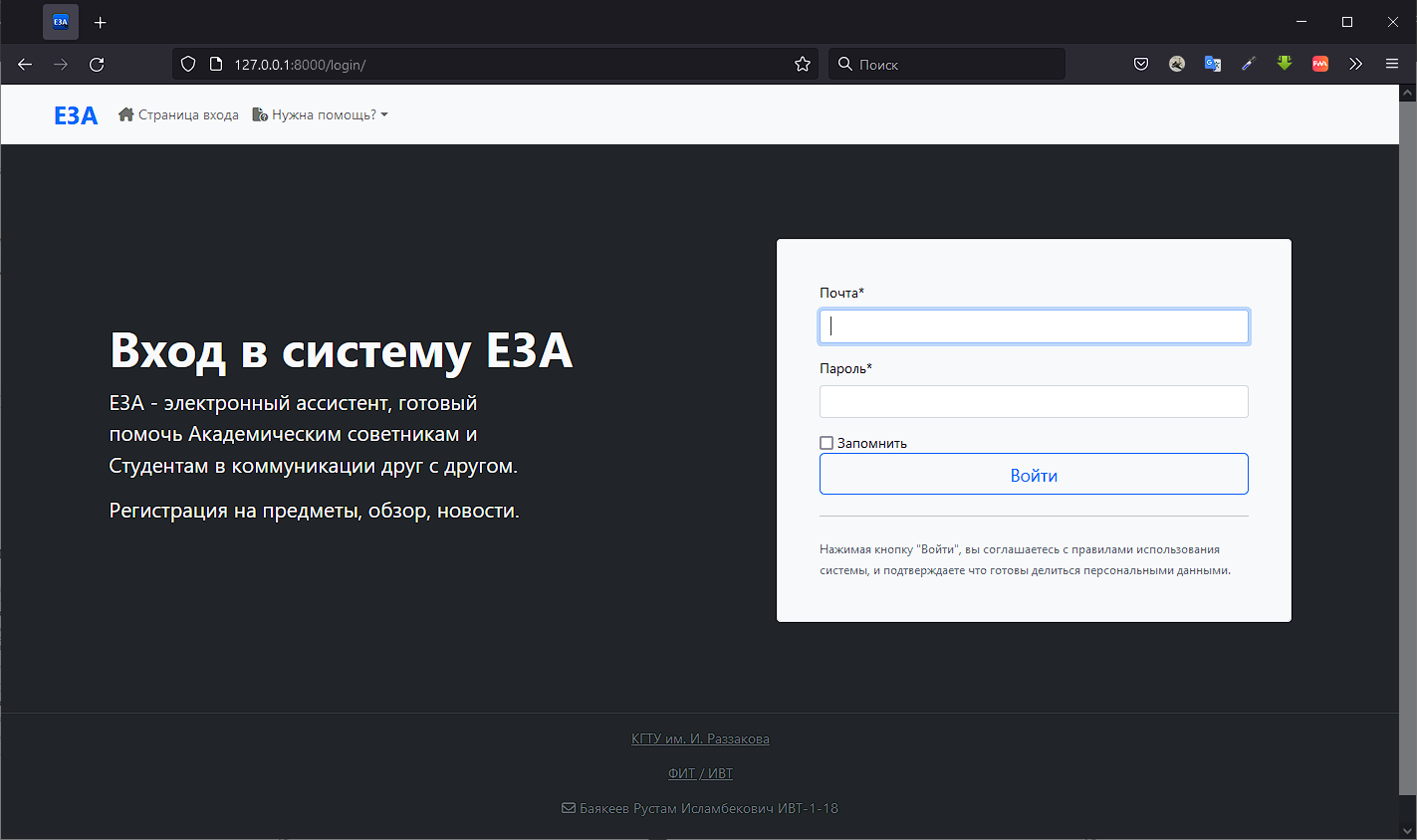


Рис. 3.1  
Окно входа в систему.

В окне, указанном на рисунке 3.1, показано окно входа в систему, 2 поля кнопок для ввода данных и кнопка для подтверждения действия.

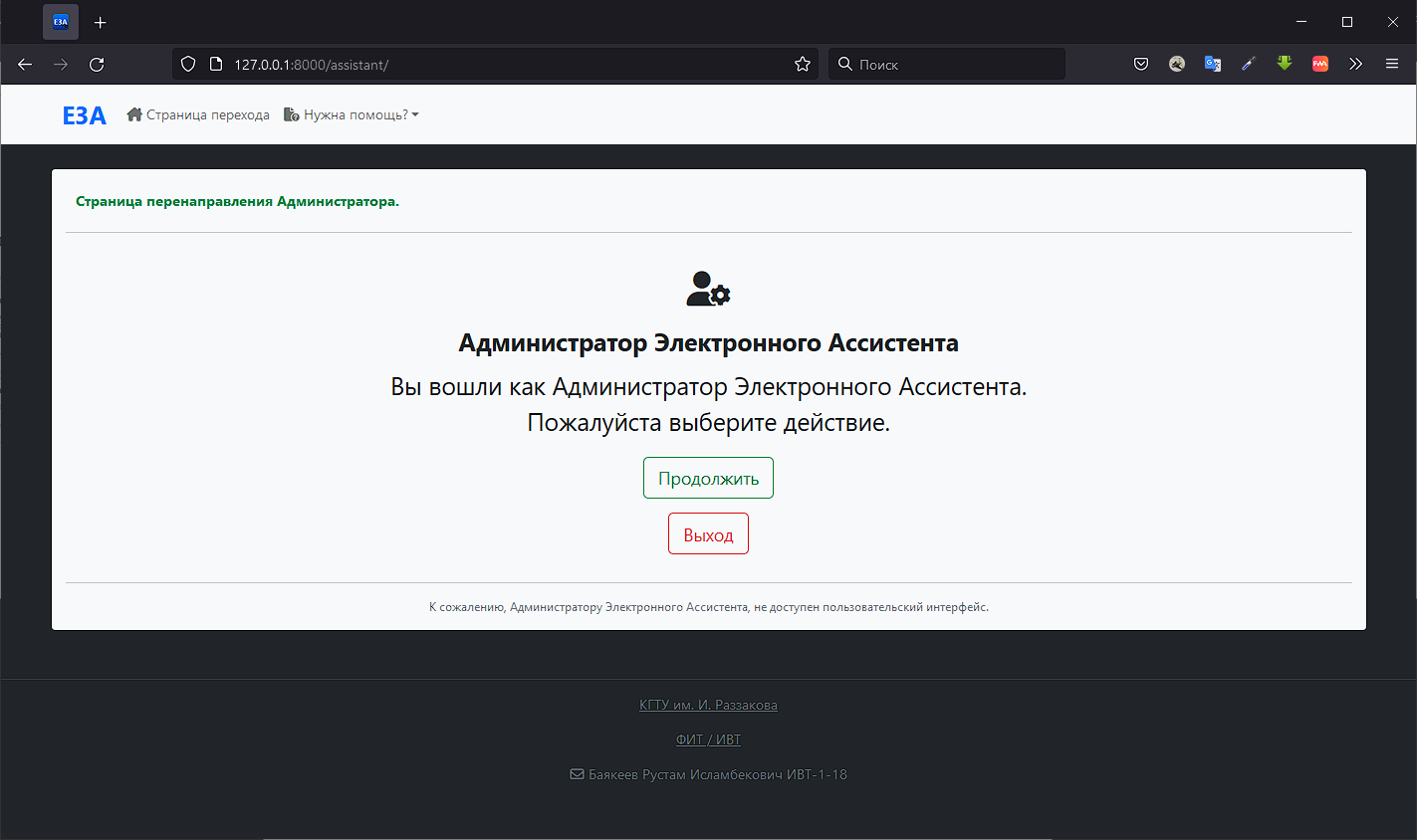


Рис. 3.2  
Окно переадресации в панель администратора.

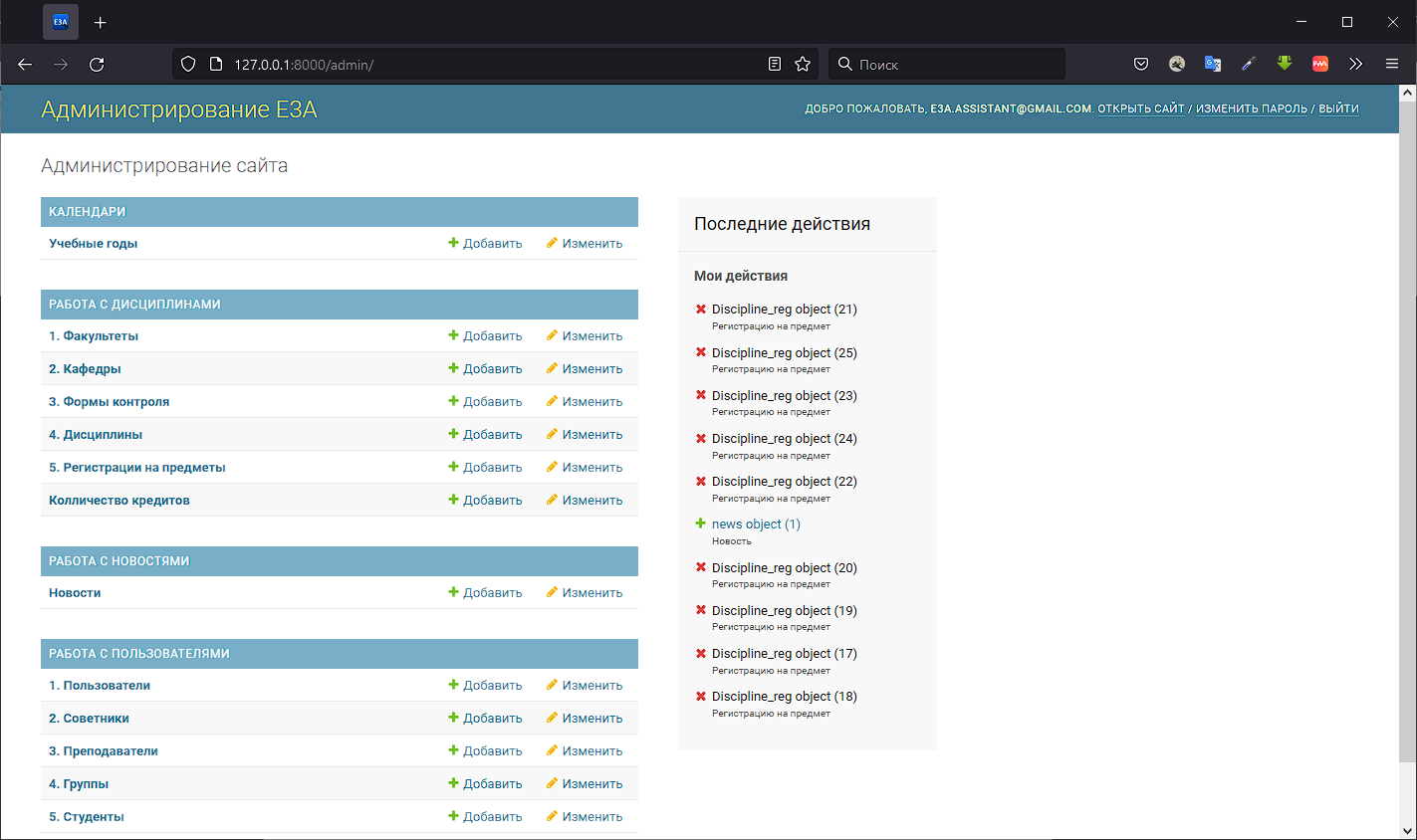


Рис. 3.3  
Панель администратора.

Основная работа в панели администратора завязана на контроле, поступления новых пользователей, изменения рабочей программы и т.п. происходит, в обведенном к красную рамку поле, на рисунке 3.4.

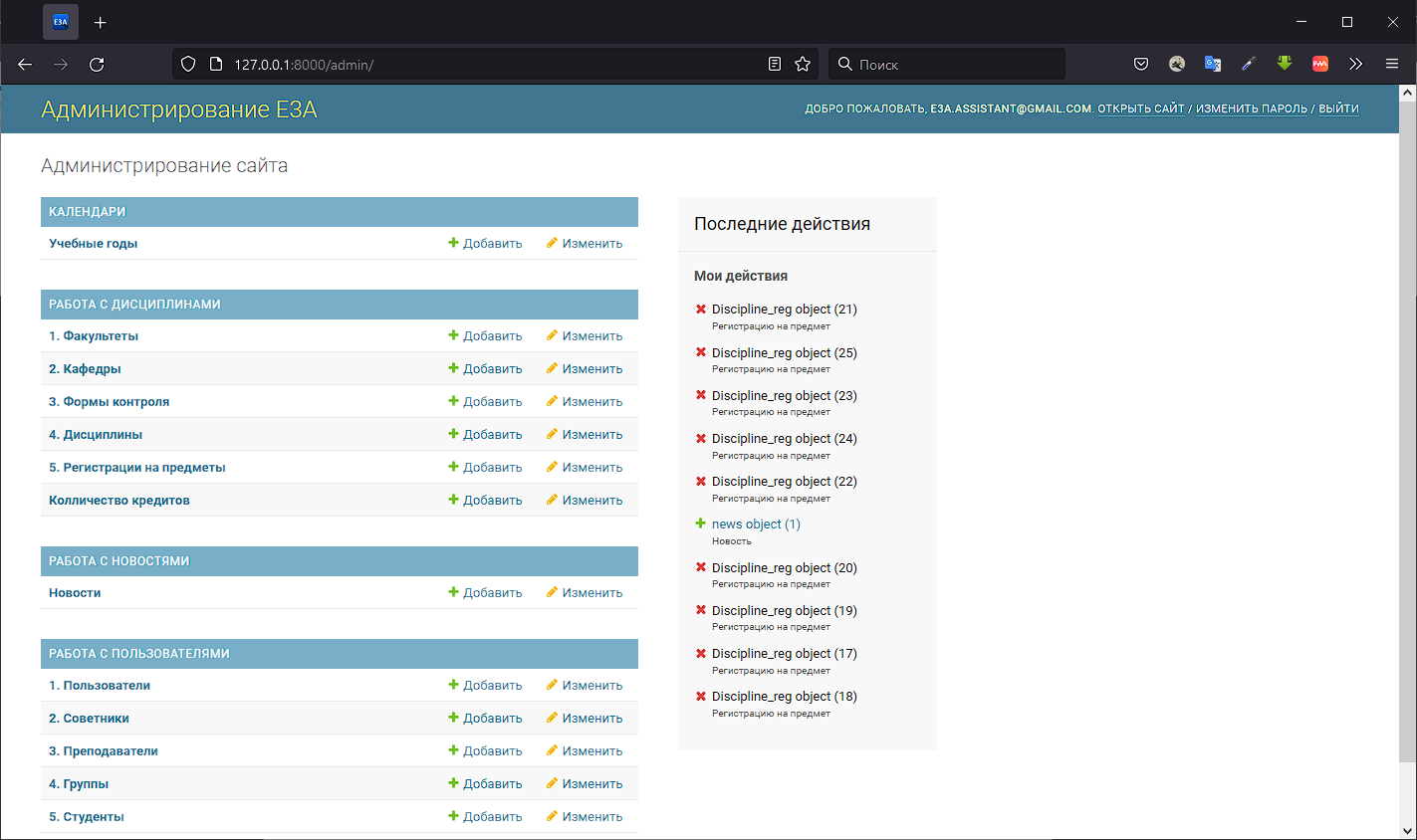


Рис. 3.4  
Основные рабочие поля в панели администратора.

Далее начиная с рисунка 3.5, обзор **интерфейса советника**.

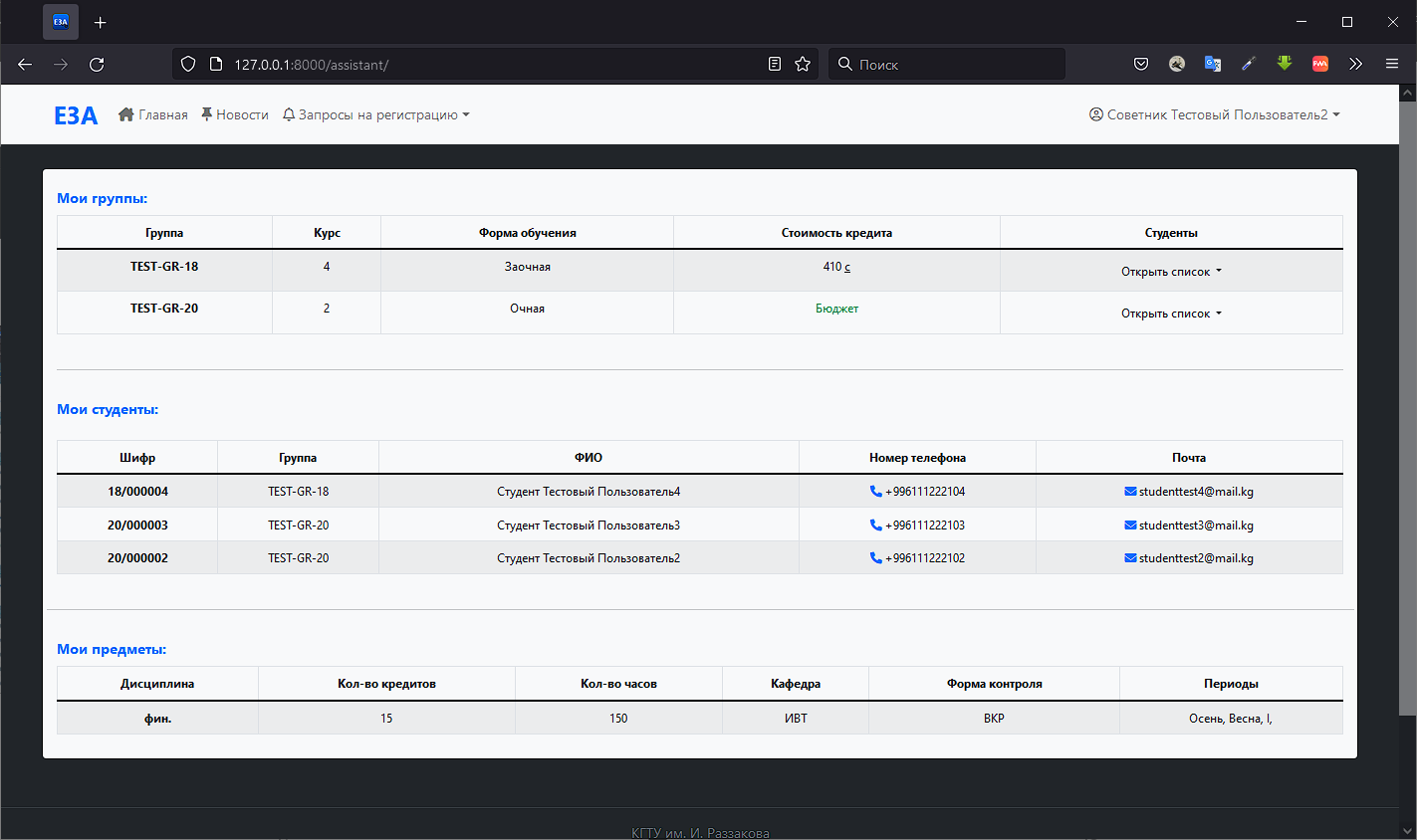


Рис. 3.5  
Основные рабочие поля.

В интерфейсе пользователя отображается вся необходимая информация, о своих группах, своих студентах, и своих дисциплинах (если они есть) на рисунке 3.6.

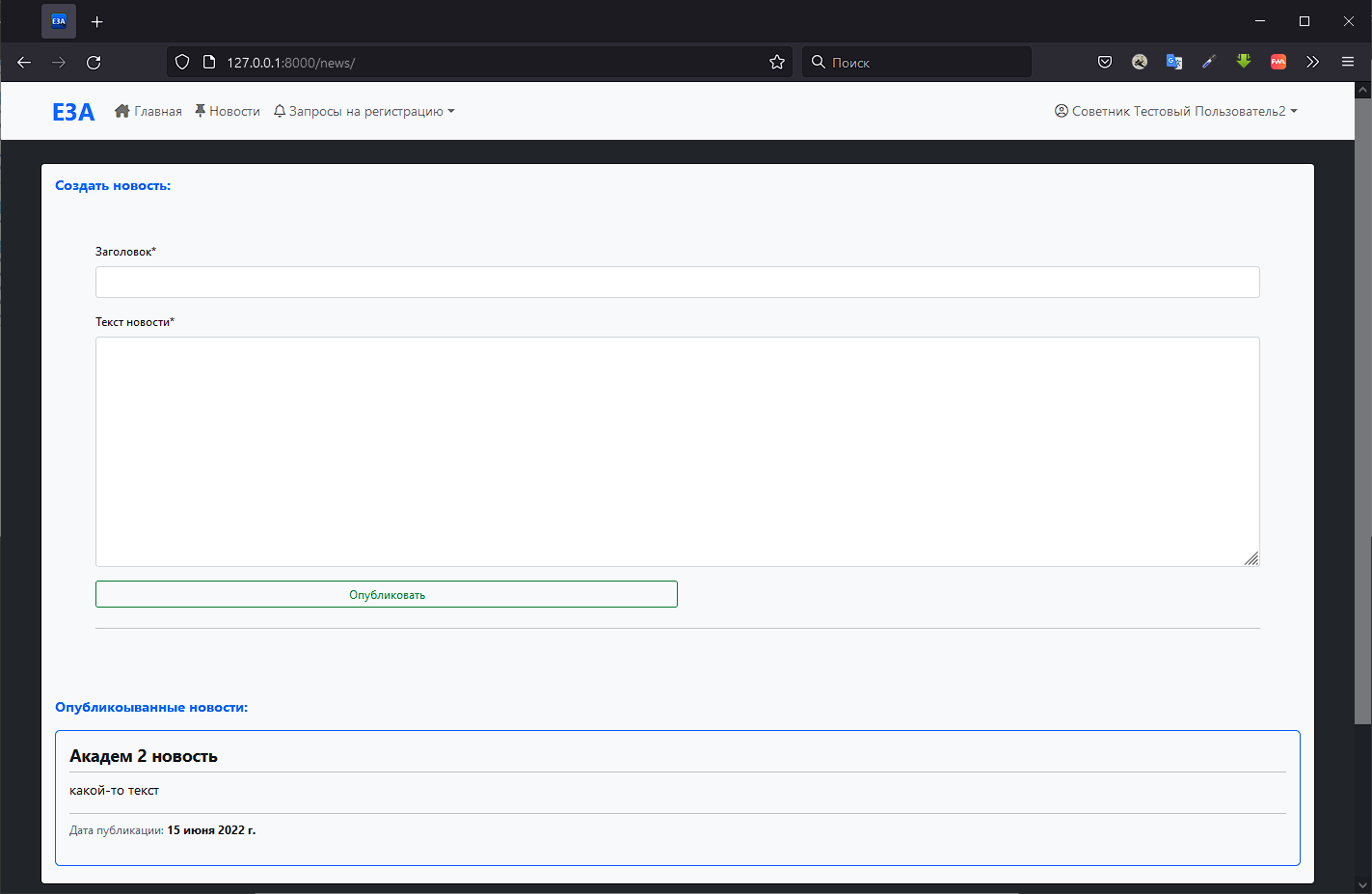


Рис. 3.6  
Окно публикации новостей.

Так же в данном окне можно увидеть все свои новости на рисунке 3.7.

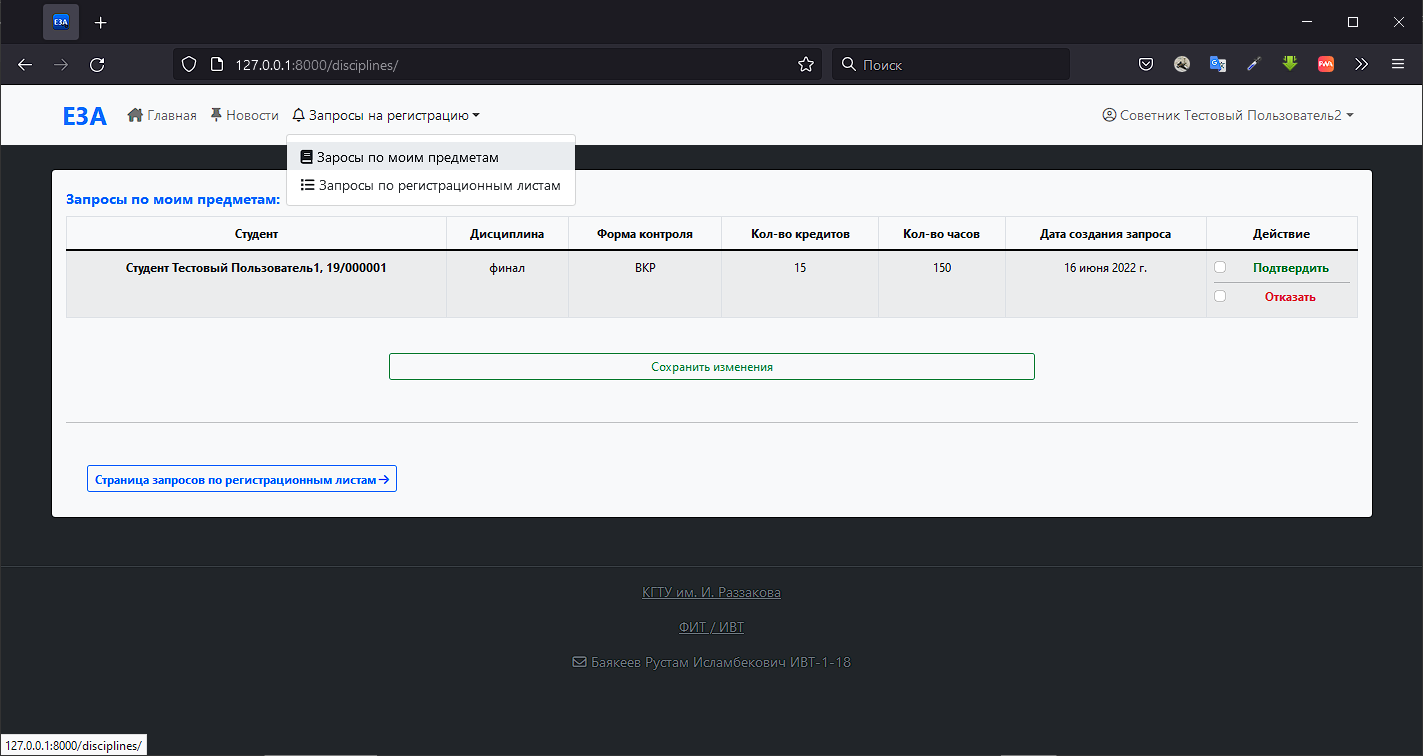


Рис. 3.7  
Окно подтверждения регистраций студентам по своим предметам.

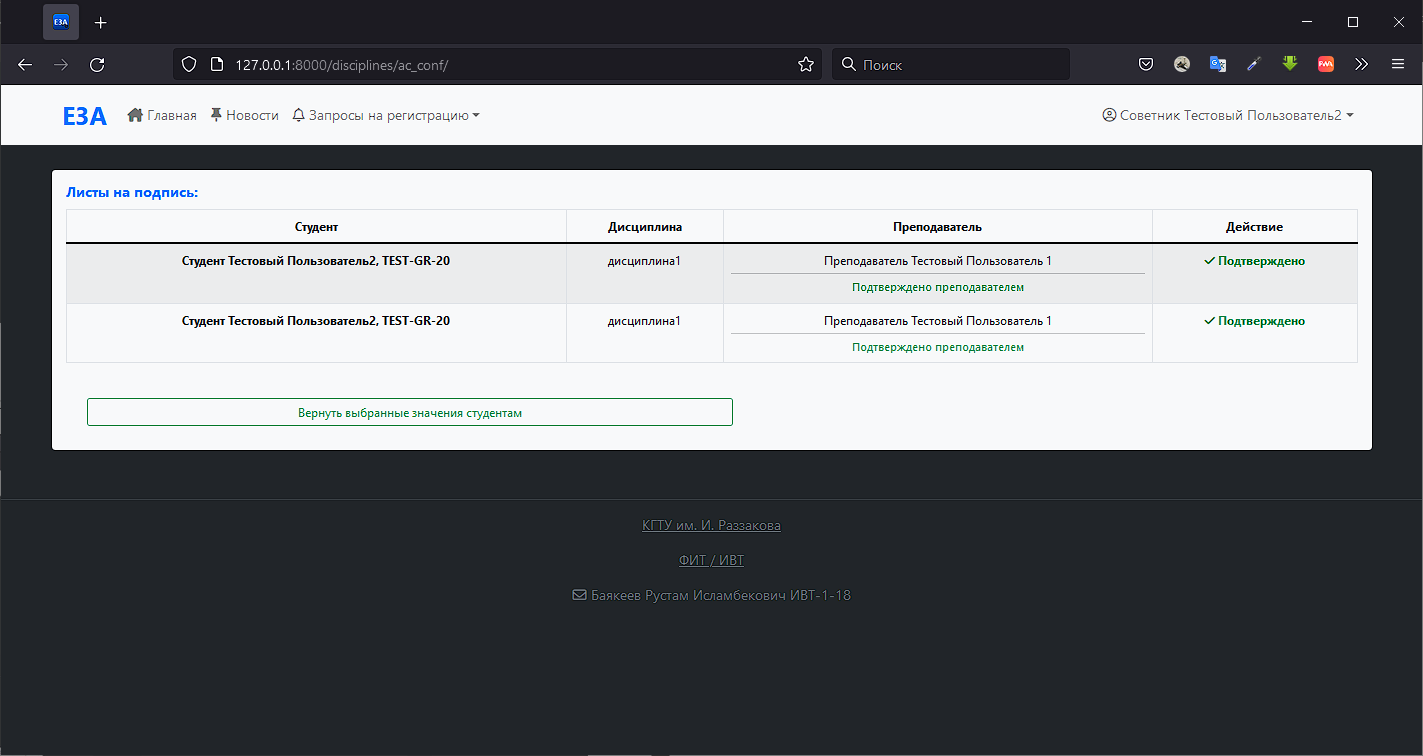


Рис. 3.8  
Окно подтверждения регистраций студентам по предметам из регистрационных листов.

**Интерфейс преподавателя.**

Интерфейс преподавателя самый простой, в нем имеется лишь окно с дисциплинами этого преподавателя на рисунке 3.9.

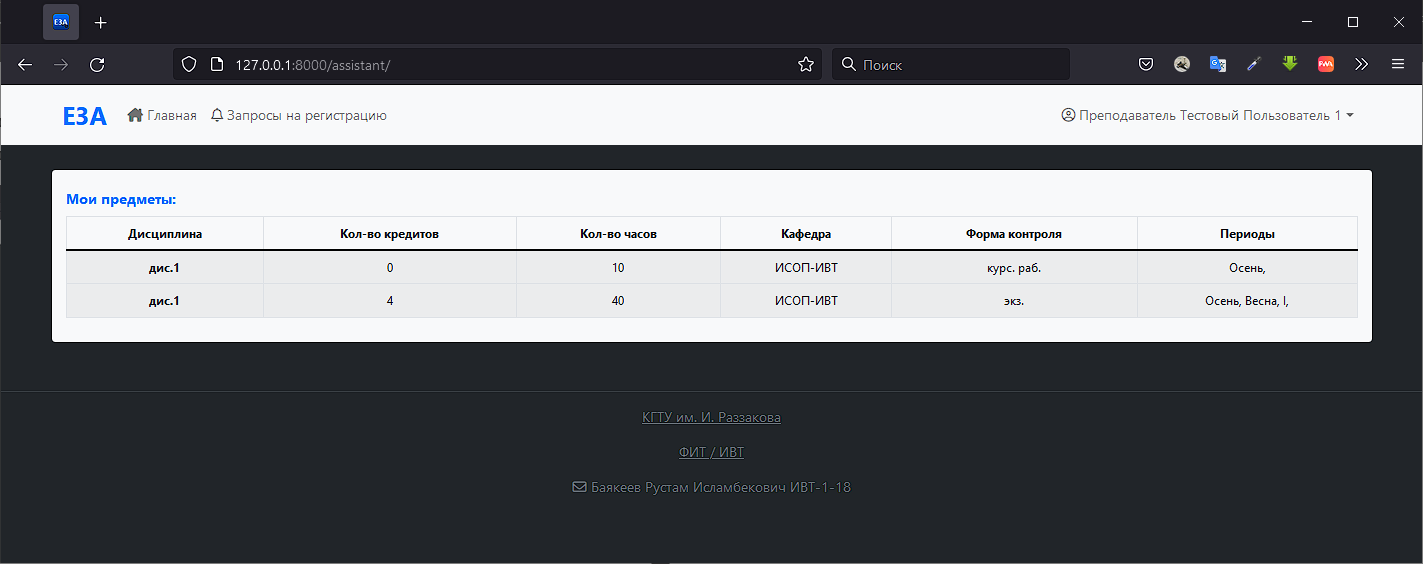


Рис. 3.9  
Окно главной страницы преподавателя.

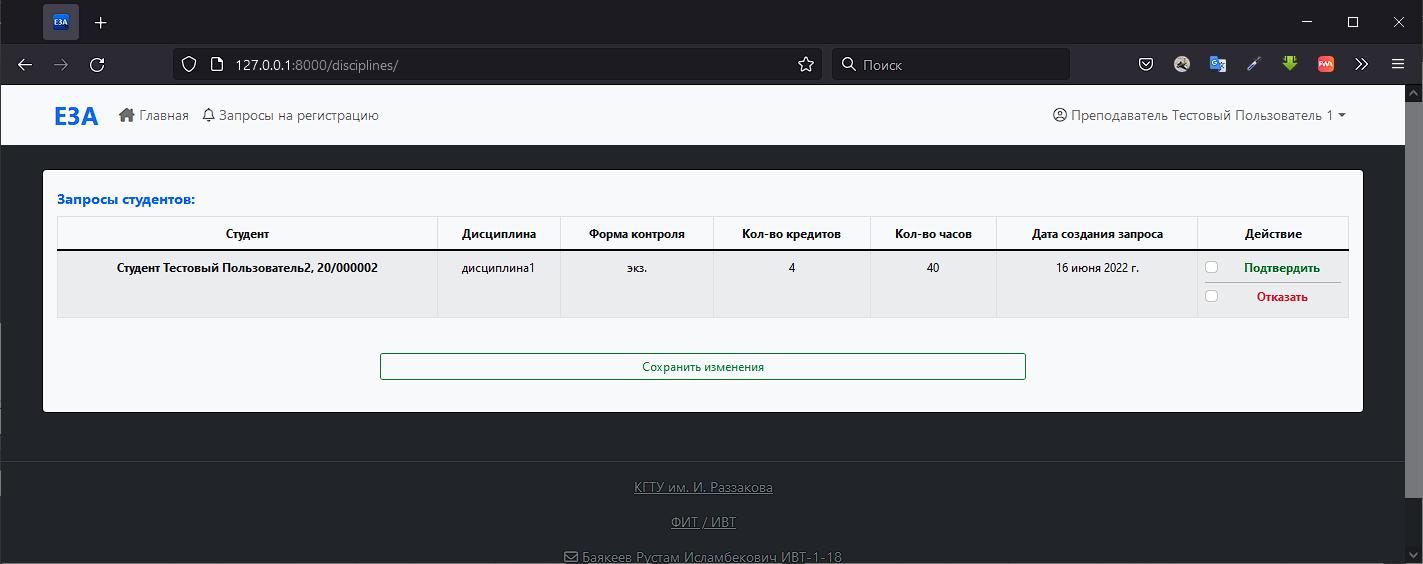


Рис. 3.10  
Окно подтверждения регистраций студентам по своим предметам

**Интерфейс студента.**

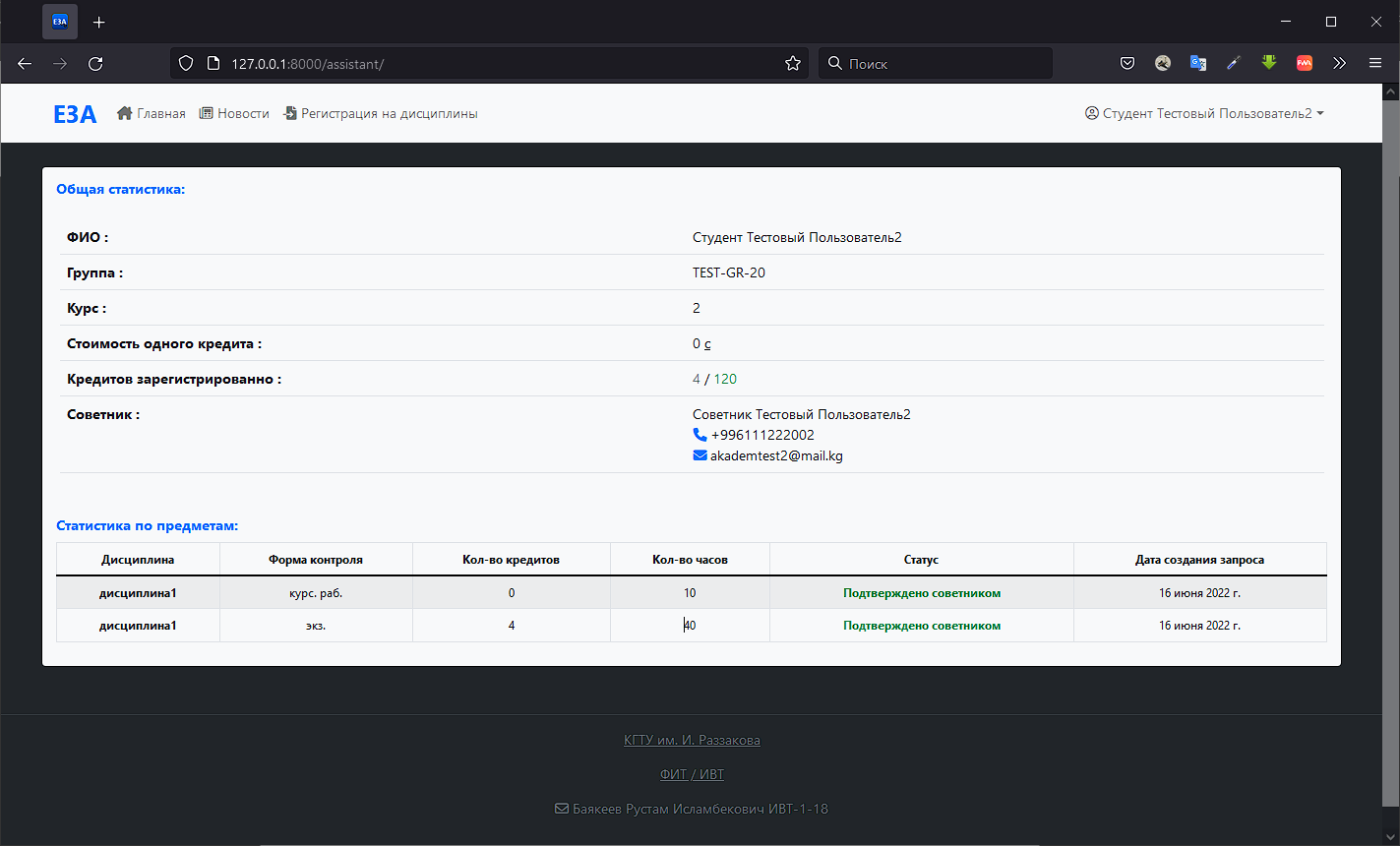


Рис. 3.11  
Главная страница

Выведена основная, необходимая студенту информация. Такая как общее описание, возможность связаться с советником, и зарегистрированные дисциплины, на рисунке 3.12.

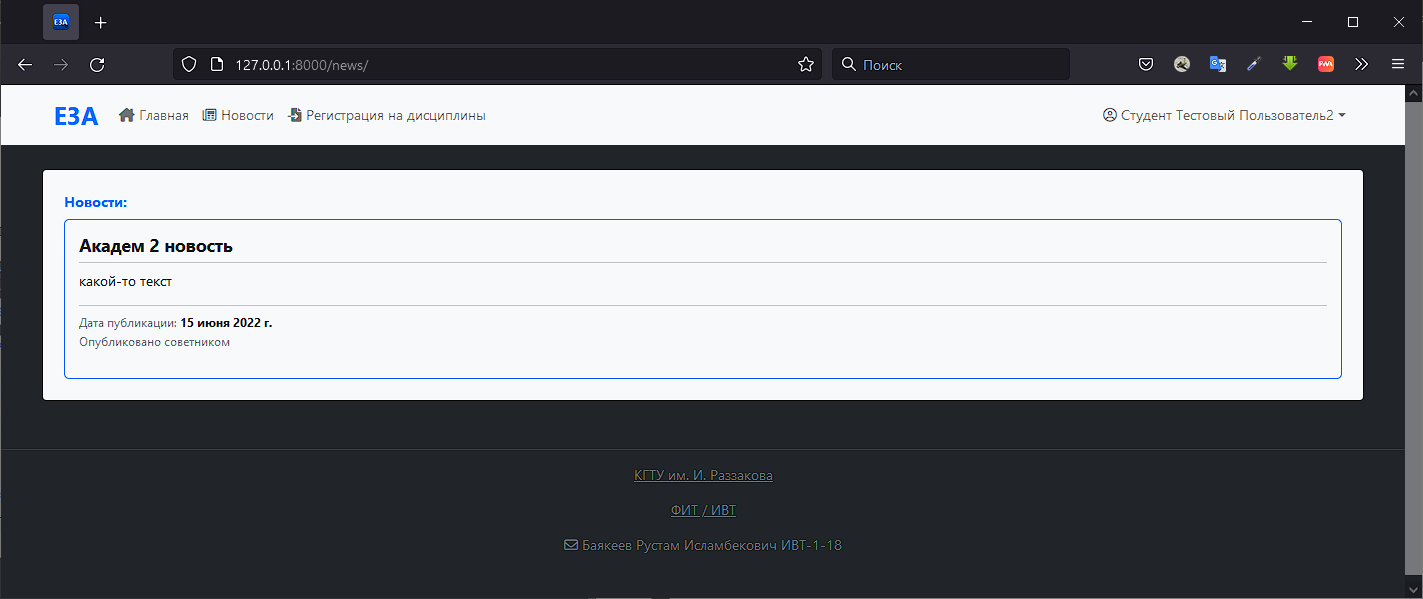


Рис. 3.12  
Новости/События, созданные академическим советником.

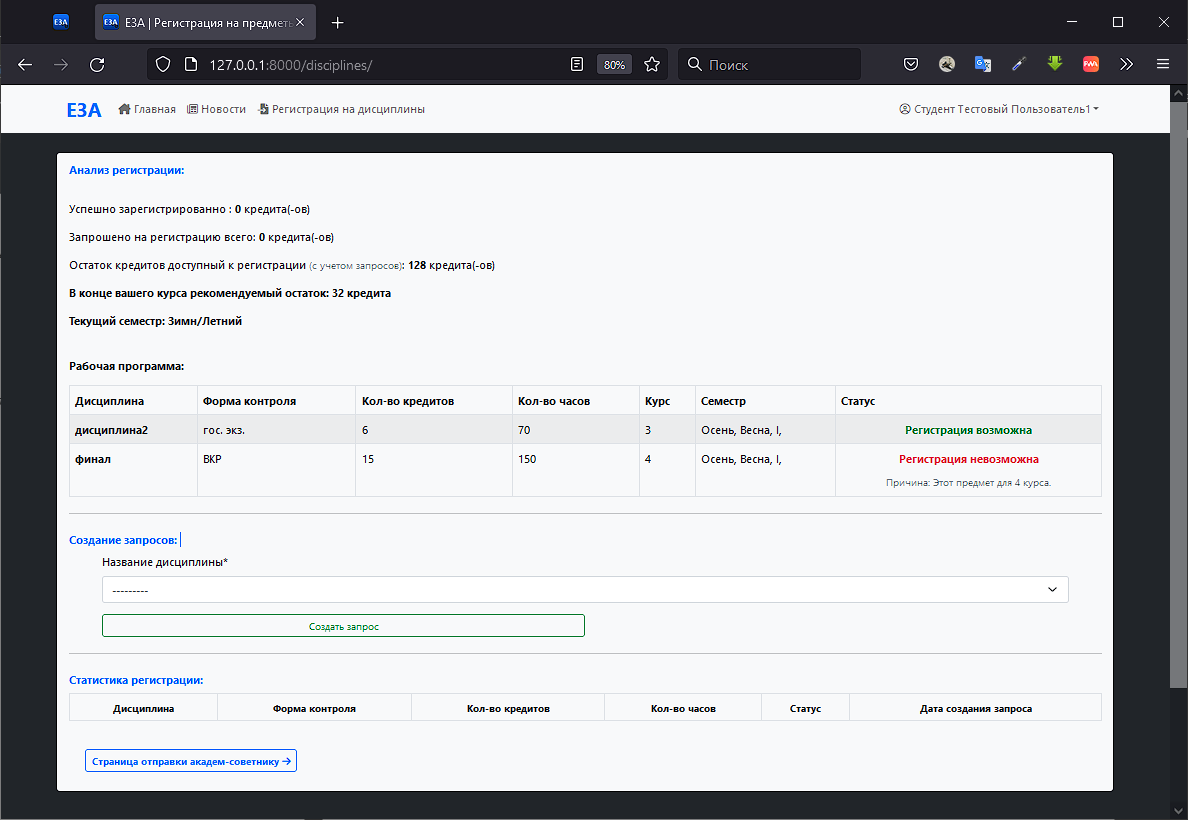


Рис. 3.13  
Окно регистрации на дисциплины + Анализ.

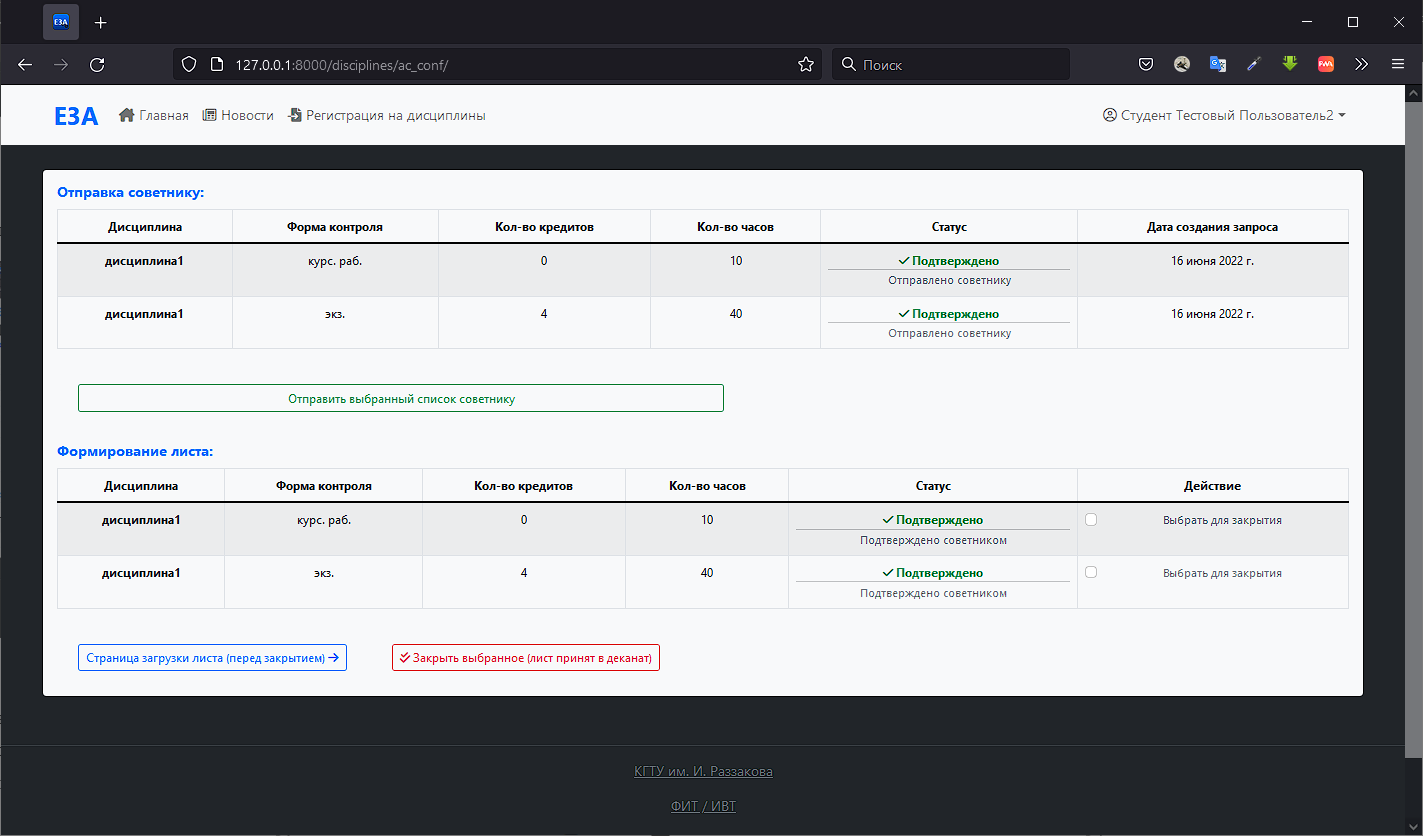


Рис. 3.14  
Окно согласования с советником и формированием регистрационного листа.

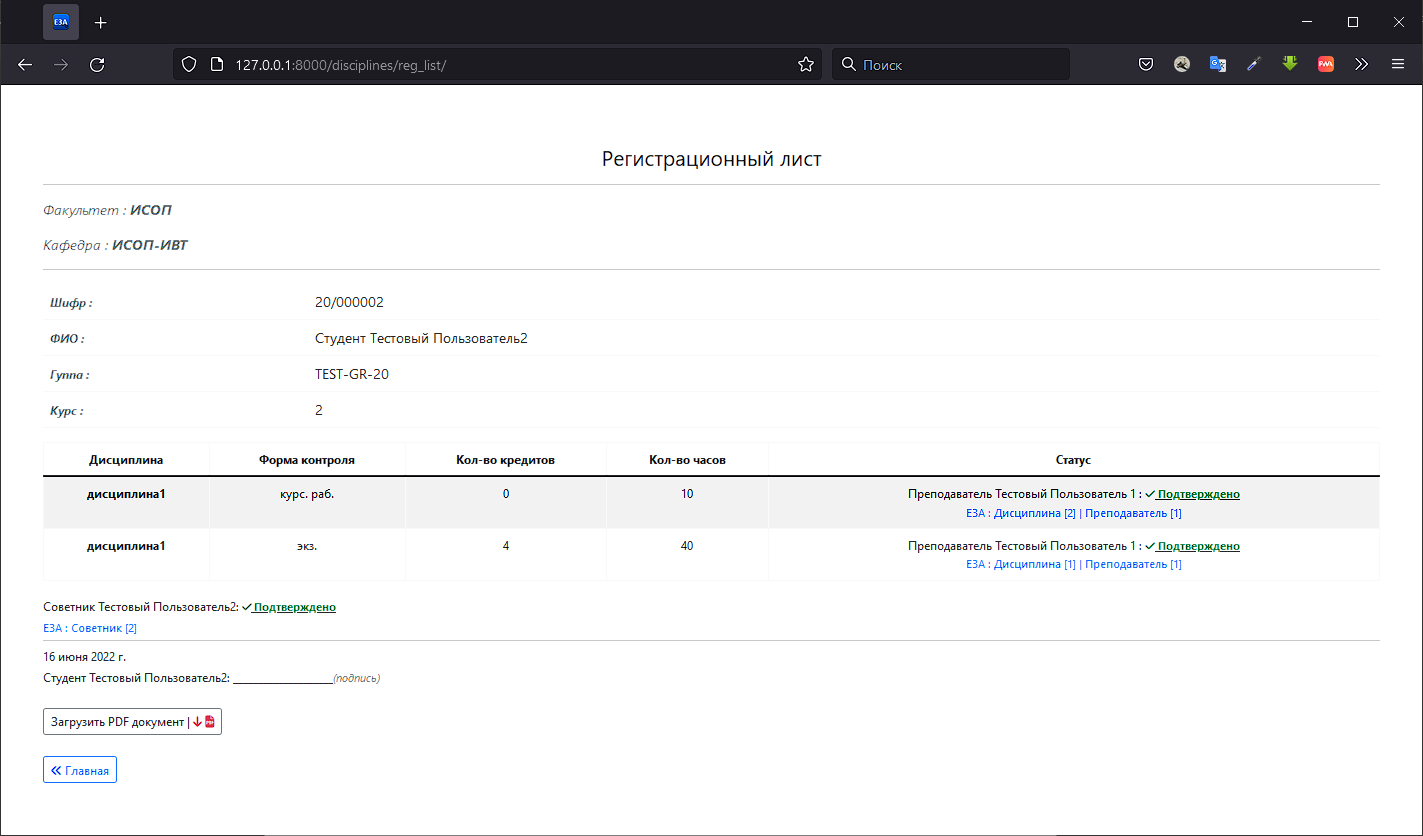


Рис. 3.15  
Регистрационный лист (страница загрузки)

**Общий интерфейс.**

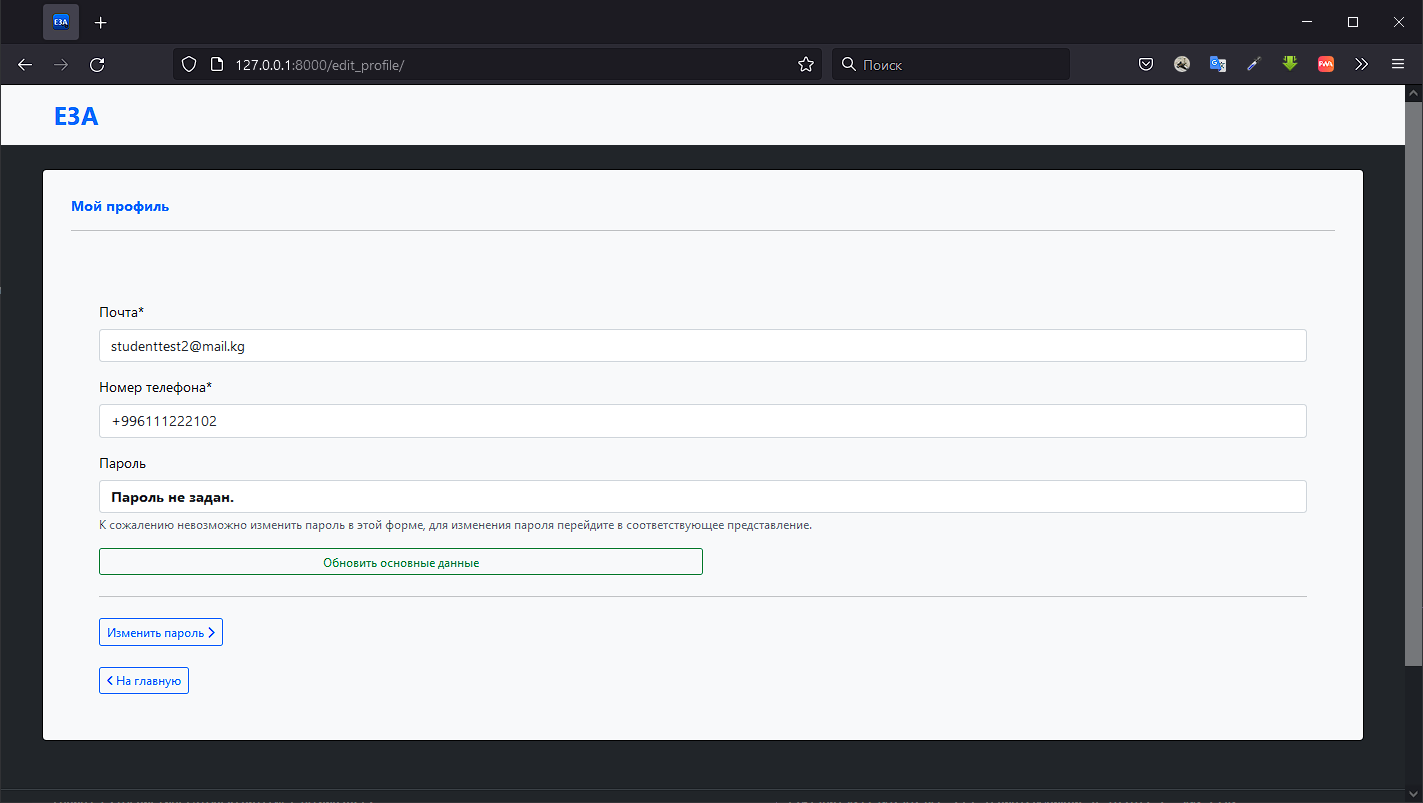


Рис. 3.16  
Редактирование профиля

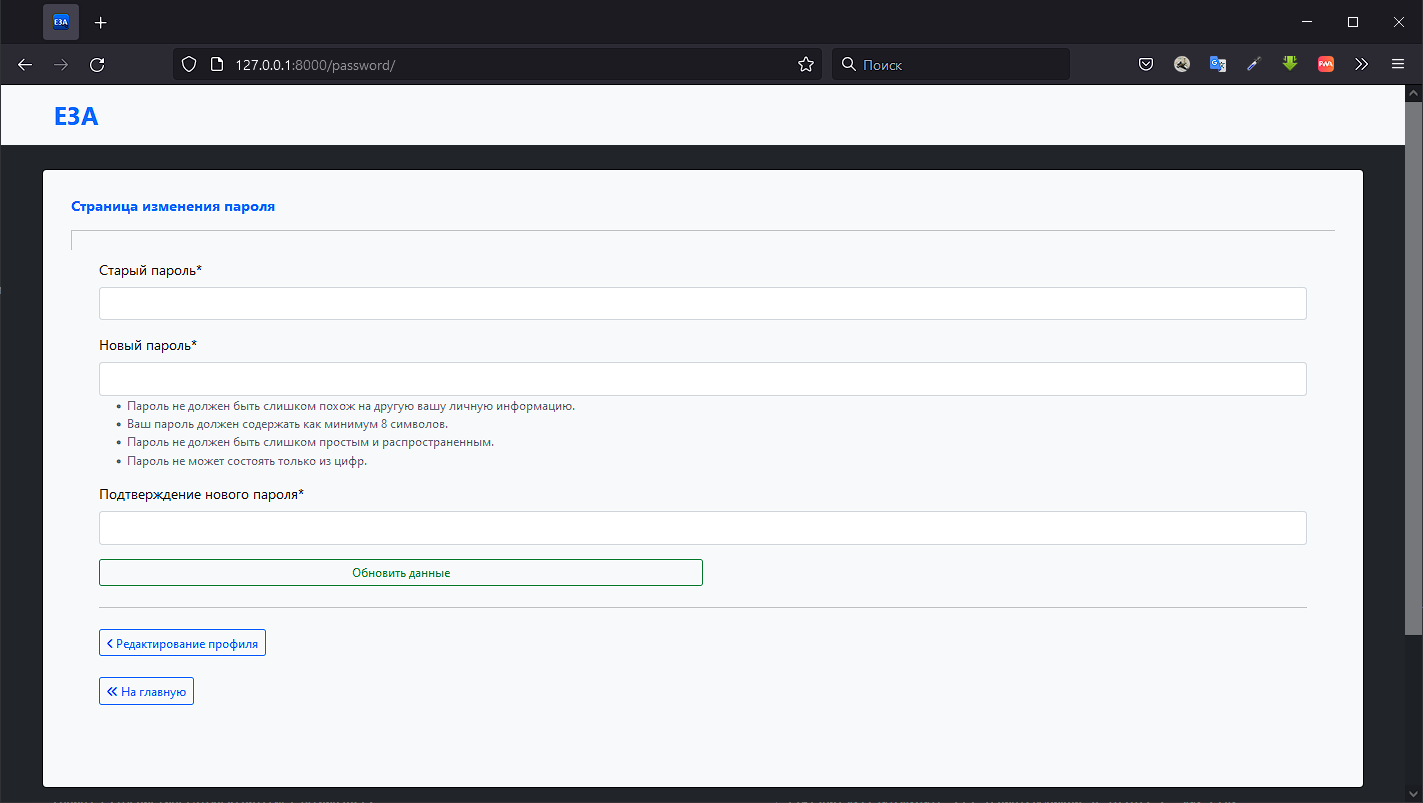


Рис. 3.17  
Изменение пароля

### Руководство разработчику

Запуск приложения:

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

|    Комманды (CMD)                      |

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

|    .\venv\Scripts\activate             |

|    py .\project\manage.py runserver    |

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

Суперпользователь (/admin):

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

|    Почта             |    e3a.assistant@gmail.com                  |

|    ФИО               |    Администратор Электронного Ассистента    |

|    Номер телефона    |    +996111222222                            |

|    Пароль            |    lkd9j7WrAtSHBM4xIb8cBac4LkolOqvs         |

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

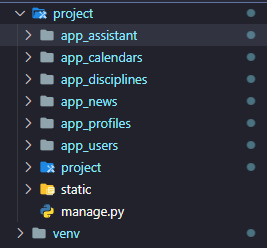


Рис. 3.18  
Приложения проекта

1. **app\_assistant** – Отвечает за работу главной страницы и всех алгоритмов связанных с ней.
2. **app\_disciplines** – Отвечает за работу с моделями, представлениями, ссылками, формами и т.п. связанными с дисциплинами.
3. **app\_news** – Отвечает за работу с моделями, представлениями, ссылками, формами и т.п. связанными с новостями.
4. **app\_profiles** – Перенаправляет на редактирование профиля.
5. **app\_users** – Хранит в себе все что связанно с пользователями и группами.

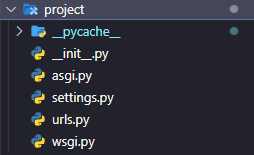


Рис. 3.19  
Проектная папка

В проектной папке самыми важными файлами являются **urls.py**, и **settings.py.**

В них хранятся все настройки приложения включая подключение всех остальных модулей.

В остальном же во всех папках приложений файлы расположены в схожем составе.

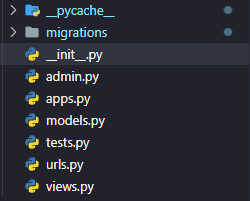


Рис. 3.20  
Проектная папка

* Папка **migrations**, содержит в себе файлы с историей миграций.
* **admin.py** хранит в себе данные регистрации моделей БД к шаблонизатору панели администратора.
* **apps.py** хранит информацию о приложении в целом. Включая название как к приложению, могут обращаться другие модули и т.п.
* **models.py** хранит в себе все модели которые как-то связаны с тематикой приложения.
* **urls.py –** дает представлением ссылки, и помогает связываться по ссылкам во всем проекте.
* **views.py –** хранит в себе представления, передающие свои функции и т.п. в приложение.
* **forms.py –** хранит в себе формы, данные форм и все что только может быть связанно с формами.

# Заключение

В период разработки выпускной квалификационной работы, была разработана система, позволяющая коммуницировать студентам и академическим советникам, а также регистрироваться на предметы, формировать регистрационные листы, получать подтверждения удаленно.

**Достоинства приложения:**

* Простота и логичность интерфейса.
* Быстрая, безошибочная обработка данных.
* Уникальность благодаря узко направленности.
* Модульность, благодаря которой систему можно доработать и превратить в полноценный образовательный портал.

**Недостатки приложения:**

* Отсутствует доступ к AVN.
* Из-за отсутствия доступа к AVN, неоткуда брать информацию об оплате.
* Снова, из-за отсутствия доступа к AVN, не была получена достоверная информация о структуре базы данных.

В качестве дополнительной опции, было введено несколько полезных решений, таких как вызов по мобильному номеру или написание письма на электронную студенту или академическому советнику.

Так же из дополнительных опций, были введены новости, позволяющие создавать новости и информировать студентов о важных событиях.

**Подведение итога.**

В процессе разработки, я научился использовать инструменты для разработки web-приложений, такие как Django в связке с PostgreSQL, на достаточно уверенном уровне, разобрался в некоторых частях учебного процесса, и сделал для себя выводы, о дальнейшем пути развития.

# Список литературы

**Бэкенд:**

* <https://djbook.ru/rel3.0/>
* <https://docs.djangoproject.com/en/3.2/>
* https://pypi.org/project/psycopg2/
* <https://github.com/parallax/jsPDF>
* <https://html2canvas.hertzen.com/>

**Фронтенд:**

* <https://getbootstrap.com/docs/5.2/getting-started/introduction/>
* <https://django-crispy-forms.readthedocs.io/en/latest/>
* <https://fontawesome.com/docs>
* <https://pypi.org/project/django-pwa/>