

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Волгоградский государственный технический университет»

Факультет электроники и вычислительной техники

Кафедра электронных вычислительных машин и систем

Согласовано

(должность гл. специалиста предприятия)

(подпись)

(инициалы, фамилия)

«_____» 2023 г

Утверждаю
заведующий кафедрой
ЭВМ и систем

А.Е. Андреев

(подпись) (инициалы, фамилия)

«_____» 2023 г

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к выпускной работе бакалавра на тему
(наименование вида работы)

Разработка распределенной системы сбора данных для рейтинга
преподавателей

Автор Резников Никита Викторович
(подпись и дата подписания) (фамилия, имя, отчество)

Обозначение ВРБ-40461806-10.44-16-23 ПЗ
(код документа)

Группа ИВТ-460
(шифр группы)

Направление 09.03.01 Информатика и вычислительная техника
(код по ОКСО, наименование направления, программы)

Руководитель работы А.Е. Андреев
(подпись и дата подписания) (инициалы и фамилия)

Консультанты по разделам:

(краткое наименование раздела)

(подпись и дата подписания)

(инициалы и фамилия)

(краткое наименование раздела)

(подпись и дата подписания)

(инициалы и фамилия)

Нормоконтролер

(подпись и дата подписания)

Г.А. Кашина

(инициалы и фамилия)

Волгоград 2023

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Волгоградский государственный технический университет»

Кафедра электронных вычислительных машин и систем

Утверждаю
зав. кафедрой

А.Е. Андреев
(подпись) (инициалы, фамилия)

«____» 2023 г.

Задание на выпускную работу бакалавра
(наименование вида работы)

Студент Резников Никита Викторович
(фамилия, имя, отчество)

Код кафедры 10.44 Группа ИВТ-460

Тема Разработка распределенной системы сбора данных для рейтинга преподавателей

Утверждена приказом по университету от «18» января 2023 г. № 74-ст

Срок представления готовой работы 16.06.23 /
(дата, подпись студента)

Исходные данные для выполнения работы

- 1) Положение о рейтинговой оценке деятельности преподавателей
- 2) Документация Django;
- 3) Документация Django-Rest-Framework;
- 4) Документация React;
- 5) Документация Redux;

Содержание основной части пояснительной записи

- 1) Анализ процесса рейтинговой оценки
- 2) Проектирование системы
- 3) Разработка серверной части системы
- 4) Разработка клиентской части системы

Перечень графического материала

- 1) Схема отношений сущностей серверной части

- 2) Схема интеграции с системой 1С
- 3) Диаграмма последовательности взаимодействия с системой
- 4) Диаграмма прецендентов пользователей системы

Руководитель работы _____
(подпись и дата подписания)

А. Е. Андреев
(инициалы и фамилия)

АННОТАЦИЯ

В данной работе описаны этапы проектирования и разработки системы рейтинговой оценки преподавателей, состоящей из серверной части, реализующей API, и клиентской части, представляющей собой веб приложение. Объем работы составляет 83 страницы и включает 68 рисунков. При написании работы использовалось 20 источников.

ABSTRACT

This paper describes the stages of the design and development of the educator rating system, consisting of a server side that implements the API and the client side that is a web application. The volume of the work is 83 pages, includes 66 figures. The 20 sources were used in writing this paper.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	7
1 АНАЛИЗ ПРОЦЕССА РЕЙТИНГОВОЙ ОЦЕНКИ	9
1.1 Основные задачи	9
1.2 Участники оценки	9
1.3 Собираемые данные	10
1.4 Текущая методика проведения	11
1.5 Внедряемое решение	13
1.6 Поиск и обзор аналогов	15
2 ПРОЕКТИРОВАНИЕ СИСТЕМЫ	18
2.1 Детализация общих требований	18
2.2 Пользовательские сценарии	18
2.3 Выбор общей архитектуры	22
2.4 Выделение сервисов	24
2.5 Проектирование базы данных	25
3 РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ СИСТЕМЫ	28
3.1 Выбор инструментов разработки	28
3.2 Подготовка рабочего окружения	31
3.3 Создание моделей данных	44
3.4 Регистрация моделей в панеле администратора	48
3.5 Создание сериализаторов моделей	49
3.6 Создание представлений	50
3.7 Создание пользовательских разрешений	51
3.8 Подключение представлений к маршрутизатору	51
3.9 Интеграция с системой 1С	52
3.10 Настройка почтовых оповещений	57
3.11 Настройка CORS политик	57
3.12 Интернационализация проекта	58
3.13 Средства предзаполнения базы данных	59
3.14 Кастомизация панели администратора	60
3.15 Настройка инструментов работы с API	61
4 РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ СИСТЕМЫ	64

4.1	Выбор инструментов разработки	64
4.2	Подготовка рабочего окружения	65
4.3	Работа с Redux	69
4.4	Определение маршрутов	72
4.5	Создание основных компонентов	74
4.6	Обзор полученного интерфейса	79
	ЗАКЛЮЧЕНИЕ	81
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	82

ВВЕДЕНИЕ

Основным предназначением университета как учреждения является подготовка специалистов к профессиональной деятельности. Процесс подготовки подразумевает под собой освоение студентом конкретной учебной программы в ходе всего периода его обучения. Очевидным фактом является прямое влияние качества как самих программ обучения, так и способов их реализации на результаты, выражаемые в знаниях и навыках, полученных студентом к моменту его выпуска.

Главным ресурсом, который задействуется в процессе обучения студентов, являются преподаватели. К основным обязанностям преподавателя может относиться организация и проведение учебной, воспитательной и учебно-методической работы по преподаваемой дисциплине или отдельным видам учебных занятий, участие в научно-исследовательской работе кафедры, участие в разработке методических пособий, обеспечение выполнения учебных планов и программ, контроль и проверка выполнения обучающимися выданных заданий. От качества выполнения преподавателем вышеперечисленных обязанностей сильно зависит эффективность его профессиональной деятельности.

Из высказанного становится понятно, что одним из способов повышения качества образования внутри университета является осуществление деятельности, направленной на повышение мотивации и вовлеченности преподавателей в рабочий процесс, а также стимулирование их к росту квалификации и профессионализма.

Согласно исследованию Омского государственного университета, проведенного в 1999 году и имеющего цель определить пути преодоления мотивационного кризиса вузовских работников и совершенствования системы оплаты их труда, на вопрос о том, какие стимулы в большей степени влияют на интенсивность и качество их труда, 48% преподавателей ответили: “Общественное признание”, в том числе и со стороны студентов, 27% – “Присвоение ученой степени, звания”, 25% – “Премии, надбавки, соответствующая заработка плата”. Более современные исследованиям, например, исследование Мининского государственного университета от 2018 года имеет схожие результаты с незначительными различиями. В нем показано что весомую роль играет совокупность из финансового фактора и фактора статуса.

Вопрос о модернизации системы оценки и поощрения деятельности преподавательского состава не обошёл стороной многие университеты России. На текущий момент активно вводятся положения о рейтинге профессорско-преподавательского состава. Сами по себе положения носят общий характер в которых устанавливается регламент проведения оценки, уточняются собираемые данные, приводятся формулы расчета и прочее.

Устанавливаемые университетами положения не являются полным решением вышеописанной проблемы, так как необходим еще и инструмент, который будет сопровождать процесс рейтинговой оценки. Внедрение такого инструмента в рабочий процесс позволит автоматизировать и упростить проведение мероприятий связанных со сбором и подсчетом рейтинга, а также повысит вовлеченность преподавателей университета через использование ими этого инструмента.

Таким образом объектом исследования данной работы является рейтинговая оценка деятельности преподавателей университета, а предметом исследования – автоматизация этого процесса посредством использования информационных технологий.

Цель работы – создание инструмента для сбора данных об эффективности деятельности преподавателей университета.

Для достижения поставленной цели определен следующий ряд задач:

- 1) анализ и изучение существующего регламента университета относительно анализа деятельности преподавателей;
- 2) составление списка требований к разрабатываемой системе;
- 3) поиск и анализ схожих систем, и выявление их особенностей;
- 4) проектирование системы с соблюдением установленных требований;
- 5) реализация системы.

Данная работа состоит из введения, четырёх глав, заключения и списка литературы. Первая глава посвящена анализу предметной области, изучению требований и существующих положений, поиску и обзору аналогов. Во второй главе рассматривается проектирование системы. Третья глава описывает процесс разработки серверной части системы. В четвёртой главе показан процесс разработки клиентской части системы.

1 АНАЛИЗ ПРОЦЕССА РЕЙТИНГОВОЙ ОЦЕНКИ

1.1 Основные задачи

Перед тем как приступать к подробному изложению описания процесса необходимо определить какие задачи, решаемые использованием рейтингового анализа, являются основными. В большинстве положений издаваемых университетами, в том числе ВолгГТУ, устанавливаются четыре основные задачи:

- 1) создание фактографической информационной базы, всесторонне отражающей деятельность преподавателей;
- 2) совершенствование деятельности и развитие университета через критический анализ коллективом результативности собственного труда;
- 3) стимулирование видов деятельности, способствующих повышению рейтинга университета в целом;
- 4) получение единых комплексных критериев для оценки и контроля уровня и эффективности работы преподавателей.

Из вышеперечисленных задач можно понять что подход к процессу анализа имеет как индивидуальный, так и коллективный характер. Это примечание необходимо учитывать при проработке поведения системы на этапе проектирования. Предполагается наличие возможностей как для индивидуального так и относительного (коллективного) учёта достижений.

1.2 Участники оценки

Основными участниками рейтинговой оценки являются сами преподаватели. При таком подходе существует ряд преимуществ, а именно:

- преподаватель заинтересован в своевременном внесении значений своих показателей;
- преподаватель заинтересован поддерживать значение своих показателей в актуальном состоянии;
- работа по внесению данных равномерно распределена между персоналом университета.

Однако существуют и некоторые недостатки:

- необходимо наличие проверяющих, удостоверяющих корректность внесенных данных;
- новым сотрудникам необходимо потратить некоторое время на ознакомление с системой и источниками собираемых с них данных.

При этом второй недостаток возможно частично решить на этапе проектирования системы, путем упрощения пользовательского опыта.

Предполагается, что за каждой кафедрой закрепляется отдельный проверяющий. Обычно в качестве проверяющего на кафедре назначается её заведующий.

Также в рейтинговом анализе принимает участие комиссия, которая занимается подведением итогов и назначением, основанных на них, надбавок для преподавателей.

1.3 Собираемые данные

Не существует строго установленного набора показателей общего для всех университетов России. Каждый университет устанавливает его самостоятельно. Так например, в ВолгГТУ, согласно изданному положению, показатели разделяют на шесть категорий:

- 1) потенциал – фиксирует квалификационный потенциал преподавателя, накопленный им за все время работы;
- 2) научная и творческая деятельность – характеризует эффективность работы преподавателя в научной, инновационной и творческой сфере;
- 3) образовательная деятельность – характеризует эффективность работы преподавателя в сфере предоставления образовательных услуг (педагогическая, научно-методическая, организационно-педагогическая работа);
- 4) финансовые показатели – отражают результаты работы преподавателя при привлечении дополнительного финансирования;
- 5) административная деятельность – характеризует вклад в работу ВУЗа преподавателя, выполняющего дополнительные административные функции;
- 6) международная деятельность – отражает эффективность работы преподавателя в области международной деятельности.

В каждую из категорий входит множество показателей, которые также опре-

делены в положении университета. Количество этих показателей достаточно велико (более ста пунктов). В тексте работы они не приводятся за ненадобностью уточнения.

По каждому конкретному показателю преподавателем с определенной периодичностью заносится значение, достигнутое индивидуально им.

1.4 Текущая методика проведения

Существующий регламент ВолгГТУ определяет следующую цепочку действий по проведению рейтинговой оценки. В конце каждого года выпускается указ о начале рейтинговой оценки профессорско-преподавательского состава. За этим следует выдача кафедрам анкет для заполнения, в которые вносятся данные в течении некоторого срока. После чего происходит сбор заполненных анкет и перенос данных в цифровую базу. Затем выполняется проверка и возможные исправления данных, уже за которыми следует подведение предварительных итогов. Заключительным этапом становится утверждение итогов и выпуск приказов.

При этом за достоверность предоставленных данных полную ответственность несут преподаватели и заведующий кафедрой в роли проверяющего.

Если возникает необходимость в корректировке данных, кафедрам представляются таблицы, содержащие данные, введенные с анкет преподавателей в последнем отчетном году. Корректировка проводится исключительно на основании служебной записи заведующего кафедрой, в которой указываются причина возникновения ошибки и новое значение показателя после исправления.

1.4.1 Недостатки текущей методики

Как можно понять, текущая методика проведения рейтинговой оценки имеет ряд недостатков.

Первая проблема заключается в необходимости переноса данных с анкет в цифровую базу. Возникает необходимость в использовании человеческих ресурсов в задаче, имеющей примитивный характер. При этом такая деятельность затрачивает значительно большее количество времени при выполнении человеком нежели машиной. Помимо этого в процессе самого переноса возникает тот или иной шанс допущения человеком ошибки в исходно верные данные.

Вторая проблема связана с тем, что большинство операций в описанном процессе подразумевает необходимость взаимодействия с учебно-методическим управлением. Получение анкет, отправка анкет на подтверждение, корректировка данных – основные действия, которые должны протекать быстро и без лишних утруждений с чьей либо стороны. Сюда же можно отнести отсутствие прямого доступа у преподавателей своих анкет за прошлые периоды. В исходной методике описывается, что анкеты за предыдущий отчетный год выдаются в случае необходимости. Однако прямой доступ к анкетам за прошлые периоды поможет преподавателям не только при внесении корректировок, но и при заполнении новых анкет с нуля и прочих случаях.

Третья проблема выражена в необходимости ручной проверки вводимых данных. Введенные преподавателям данные нельзя считать заведомо верными не зависимо от того, насколько тщательно они заносились. Помимо очевидных причин, таких, как человеческий фактор, можно выделить и некоторые другие. Во-первых, необходимо учитывать вопрос умышленной фальсификации с целью увеличения личного рейтинга. Во-вторых, преподаватель может не иметь всех актуальных сведений, которые так или иначе могут повлиять на значения. Сверка данных занимает ключевую позицию в процессе проведения рейтинговой оценки. Необходимо понимать, насколько здесь важно обеспечить точность заполняемых данных, при этом, по возможности, максимально автоматизировав процесс. Оптимальное соблюдение этих условий позволит снизить использование человеческих ресурсов и шанс возникновения ошибки.

1.4.2 Способы улучшения методики

В соответствии с каждой из выявленных проблем существующей методики проведения оценки необходимо определить рабочее решение, реализуемое при помощи имеющихся практических возможностей.

Рассматривая все проблемы в совокупности, необходимо понимать какие части процесса представляется возможным автоматизировать, а какие все еще должен контролировать человек.

Так например, первая проблема не требует задействования человеческих ресурсов и решается правильной организацией системы в совокупности с наличием необходимых пользовательских инструментов. Если у кафедры есть возможность

внесения данных по каждому из преподавателей непосредственно в информационную базу, то отбрасывается необходимость в ручном переносе данных с анкет.

Решение второй проблемы подразумевает упрощение процесса обмена данными между кафедрами и УМУ. Достичь этого можно формализацией каждого действия в виде отдельного пользовательского сценария с его последующей автоматизацией. Важно уделить внимание деталям внутри каждого сценария чтобы обеспечить обработку всевозможных возникающих ситуаций. Договоренность между кафедрами и УМУ об использовании такого формата взаимодействия значительно упростит процесс обмена информацией.

Третья проблема относится к тому виду задач, в которых на текущий момент сложно достигнуть полной автоматизации и избежать человеческого участия. Дело в том, что для каждого значения имеется собственный источник данных или набор правил, определяющих его корректность. Так, например, для проверки корректности значений, связанных с финансовыми показателями, необходимы актуальные данные из тех источников, которые в большинстве случаев не предоставляют их в открытом виде. Однако есть и показатели, значения которых представляется возможным проверить в открытых источниках. В качестве примера таких значений можно привести количество публикаций преподавателя. Но даже здесь не всегда получается автоматизировать процесс в силу отсутствия у некоторых источников программных интерфейсов. Резюмируя решение данной проблемы, можно сказать, что на данный момент затруднительно полностью возложить процесс проверки на программные алгоритмы, однако имеется возможность частичной автоматизации в некоторых областях. Для полной автоматизации необходима глобальная модернизация всех связанных систем, которая в том числе подразумевает внедрение программных интерфейсов для внешнего доступа. В перспективе это может полностью избавить человека от необходимости участия в описываемом процессе.

1.5 Внедряемое решение

Для практического решения каждой из проблем предлагается разработка вышеописанных программных инструментов, которые в совокупности составляют единую информационную систему.

После определения частных решений по каждой из проблем, необходимо

конкретизировать требования, которые будут предъявляться системе при её проектировании. Помимо этого, также важно учитывать замечания, приведенные при обсуждении основных задач рейтинговой оценки.

Исходя из основных задач, система должна иметь следующие базовые свойства:

- 1) хранение данных;
- 2) сбор данных;
- 3) анализ данных;
- 4) отчетность;
- 5) интеграция.

Система должна быть способна хранить данные, так как это необходимо для формирования фактографической базы, которая, помимо функций архива, может также использоваться и для аналитических расчетов с выборкой за разные периоды времени.

Система должна иметь механизмы для сбора данных о показателях эффективности преподавателей. Это может быть достигнуто, например, путём представления преподавателям электронных форм.

Система должна быть способна анализировать собранные данные для выявления различных метрик и проблемных областей. Это может включать расчет средних значений, статистический анализ, сравнение результатов с другими преподавателями и другие методы обработки данных.

Система должна предоставлять возможность создания отчетов о показателях эффективности преподавателей. Эти отчеты могут использоваться администрацией университета для принятия решений, развития преподавательского состава и оценки эффективности образовательных программ.

Наличие свойства интегрируемости является необходимым, так как подразумевается взаимодействие с другими системами университета, такими как учебно-методическое управление и прочими. Также в целом это обеспечит систему свойством свободной расширяемости и повысит эффективность обмена информацией внутри университета.

Важно также понимать, что свойство интегрируемости системы позволит вынести в отдельные модули инструменты анализа и отчётности, указанные в пунктах 3 и 4, что упростит разработку системы на начальных стадиях.

После обозначения базовых свойств, требуется сформулировать требова-

ния, относительно выявленных решений частных проблем. Все эти требования можно выразить в пользовательских сценариях, так как все они подразумевают какую-либо операцию при использовании системы. Основные сценарии приведены в виде следующего списка:

- 1) закрытая регистрация пользователей в системе;
- 2) просмотр преподавателем собственных анкет;
- 3) создание преподавателем новых анкет с внесением данных о его показателях;
- 4) просмотр проверяющим контролируемых им анкет;
- 5) утверждением проверяющим контролируемых им анкет.

Детальное рассмотрение каждого из сценариев приводится в следующей части работы, в которой описывается процесс проектирования системы.

Отметим, что здесь не приводятся сценарии, связанные с анализом данных, получением отчётности и прочими возможностями системы. Данная работа фокусируется на центральном модуле – системе, которая будет предоставлять интерфейс для внешних модулей. При таком подходе внешние модули могут разрабатываться уже независимо.

1.6 Поиск и обзор аналогов

Обозначив основные требования, открывается возможность поиска и обзора существующих систем, имеющих схожие возможности.

Перед поиском и обзором аналогов важно понимать, что система сбора данных о показателях эффективности преподавателей разрабатывается по заказу конкретного университета и имеет специфические требования, соответствующие его потребностям и контексту. Однако, разработанная система может быть адаптирована и использована и другими университетами.

Тем не менее, изучение существующих решений и учёт их особенностей может быть полезным при разработке системы. Сюда можно отнести различные функции, архитектуру, пользовательский опыт и прочие аспекты.

Для начала необходимо определить, каким образом будет выполняться поиск таких систем.

Первый способ заключается в изучении сайтов университетов и поиска на них информации о рейтинговой оценке профессорско-преподавательского соста-

ва, кафедр и факультетов. Следующим шагом является попытка определения используемых при построении рейтинга инструментов.

Второй способ состоит в поиске схожих проектов на хостингах публичных репозиториев. Самыми популярными такими хостингами являются GitHub, GitLab, GitBucket. Поиск осуществляется посредством использования ключевых понятий, таких, как “рейтинг”, “университет”, “оценка” и прочих.

Результаты первого способа показали, что существует следующий ряд факторов, противодействующих анализу:

- университеты могут в целом не использовать схему рейтинговой оценки;
- университеты могут не публиковать информацию о деталях проведения рейтинговой оценки;
- университеты публикуют только результаты рейтинговой оценки в виде электронного документа;
- университеты публикуют результаты рейтинговой оценки, полученные использованием ручного расчета;
- университеты используют инструменты для автоматического расчета, однако они являются закрытыми.

Были изучены сайты более 30 самых известных университетов России. Среди всего числа проанализированных, есть пара интересных для рассмотрения решений.

Одной из схожих систем стала ИАС «ИСТИНА», разрабатываемая МГУ, которая начала недавно вводиться в процесс рейтинговой оценки. Система связана с информационными базами научных публикаций и позволяет автоматизировать процесс получения данных о научной деятельности преподавателей. Однако на текущей стадии своего существования система не отражает все стороны, учитываемые при расчете общего рейтинга преподавателя. Исходный код разработки авторами не предоставляется, однако регистрация является открытой.

ВШЭ предлагает решение, которое носит название «Smart LMS», которое стоит рассматривать уже как целую экосистему. Платформа включает в себя различные конструкторы курсов и образовательных программ, электронные реестры (дипломы, зачетные книжки, ведомости), инструменты по работе с расписанием и другое. Среди перечисленного имеется система опроса участников учебного процесса, включающая студентов, с использованием которой формируют рейтинг преподавательского состава. Доступ предоставляется только персоналу универси-

тета и его учащимся.

Остальные же найденные системы носят недостаточно открытый для полноценного анализа характер. Таким образом, на основании всего числа проанализированных ресурсов, можно сделать заключение по первому из способов. Достаточно малый процент университетов, проводящих рейтинг ППС, использует или раскрывает публике автоматизированные системы учёта рейтинга. Если же университет имеет такую систему, то в большинстве случаев она является частично или полностью закрытой. На основании изученных положений университетов, стоит отметить, что есть активная тенденция к разработке и вводу таких систем в эксплуатацию.

Переходя к результатам второго способа, можно также выделить некоторые проблемы:

- отсутствие какой-либо документации или описания проекта;
- остановка проекта на стадии разработки;
- неполадки при сборке проекта.

Было изучено более 300 репозиториев, выданных по запросу с использованием следующих ключевых фраз:

- рейтинг университета (university rating);
- рейтинг преподавателей (educators rating, professors rating);
- ППС, профессорско-преподавательский состав (university staff);
- оценка деятельности преподавателей.

Фактическое число аналогов, найденных в открытых репозиториях и отвечающих установленным требованиям равно нулю. Среди организаций с проектами для университета, можно отметить «TNEU DEV TEAM». Среди их проектов числится и рейтинговая система, но ориентированная на студентов университета. С 2018 года команда прекратила активную разработку.

В заключении по двум способам можно сказать, что на текущий момент количество проектов связанных с рейтинговой оценкой преподавателей не велико. Выдвигаемые требования делают систему узкоспециализированной, не имеющей полноценных аналогов. В отношении этого можно сказать, что актуальность разработки системы, в дополнении к уже имеющейся, становится еще больше.

2 ПРОЕКТИРОВАНИЕ СИСТЕМЫ

2.1 Детализация общих требований

На текущий момент уже установлены некоторые базовые требования, а также набор сценариев, который система должна реализовывать. Теперь необходимо провести технический анализ по перечисленным пунктам со всеми его вытекающими. Результатом такого анализа станет набор уже технических требований, которые будут учитываться при реализации системы.

Наиболее важными при анализе стали пункты 1 и 5 – хранение данных и интегрируемость системы.

Хранение данных очевидным образом требует наличие в системе базы данных. Полноценное управление данными должно поддерживаться выбранной системой управления базами данных.

Интегрируемость подразумевает наличие публичного интерфейса. Таким образом, можно рассматривать систему как отдельный модуль, который представляет определенный набор функций, выполняющихся по внешнему запросу.

2.2 Пользовательские сценарии

Рассмотрим каждый сценарий по отдельности на основе диаграммы прецедентов показанной на рисунке 2.1

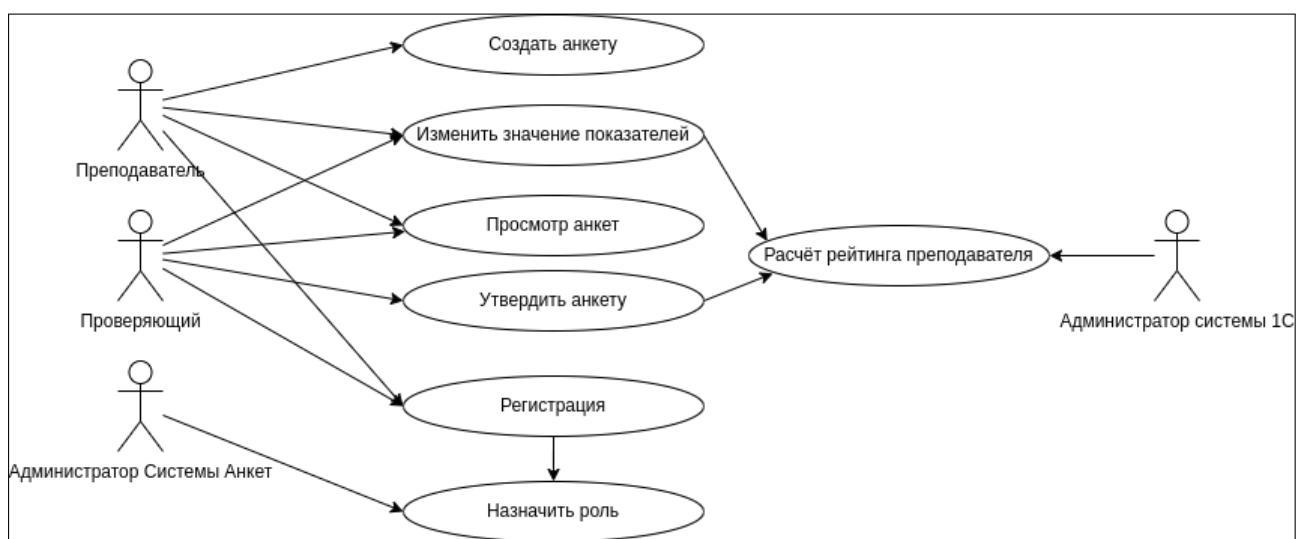


Рисунок 2.1 – Диаграмма прецедентов

2.2.1 Регистрация пользователей

Система должна предоставлять функции закрытой регистрации пользователей. Необходим механизм который бы запрещал регистрироваться пользователям, которые не имеют отношения к деятельности университета. Для того, чтобы не переусложнять этот механизм предлагается следующая стратегия. Администраторы системы заранее вносят электронные почты сотрудников университета в реестр доступных к регистрации. При регистрации, сотрудник указывает свою электронную почту и, если она находится в этом реестре, то регистрация проходит успешно, иначе пользователь информируется о том, что в регистрации ему отказано. К плюсам такого метода можно отнести следующие особенности:

- отсутствие необходимости сложных механизмов идентификации пользователей;
- изначальная привязка учётной записи сотрудника к его личной почте, что даёт ему возможность, например, восстановить пароль в случае, если он его забыл.

Также важно отметить следующие особенности, которые не относятся к плюсам, но их нельзя счесть и за минусы:

- необходимость добавления администрацией почты сотрудника в реестр, для его дальнейшей регистрации.
- в случае, если доступ к почте у сотрудника по каким либо причинам утерян, то для сохранения функций смены пароля и прочих ему необходимо также обратиться к администраторам или уполномоченным лицам, которые закрепят за профилем новую электронную почту.

Впрочем, изложенное в первом пункте можно автоматизировать путём добавления почты сотрудника в реестр при его устройстве на должность. Касательно второго пункта, можно сказать что, это скорее путь решения потенциальной ситуации, нежели недостаток метода.

Стоить отметить, что почта сотрудника может добавляться и из информационной базы сотрудников в дополнении с идентификатором сотрудника, по которому в дальнейшем может происходить получение данных о преподавателе.

2.2.2 Просмотр собственных анкет от лица преподавателя

Один из ключевых сценариев системы - это возможность просмотра анкет. С точки зрения данного сценария система должна обеспечивать пользователя следующими возможностями:

- получение списка собственных анкет с общей информацией, такой, как год анкеты, принадлежность, статус подтверждения;
- получение данных о конкретной собственной анкете, а именно списка показателей и соответствующих им значений.

При этом для пользователя в лице преподавателя запрещен просмотр анкет, не принадлежащих ему.

Полезной функцией при просмотре данных будет наличие возможность сортировки и фильтрации списка анкет.

2.2.3 Просмотр анкет от лица проверяющего

Отличием данного сценария от предыдущего является то, что помимо уже перечисленных возможностей проверяющий также должен быть обеспечен следующими возможностями:

- доступ к анкетам всех преподавателей на кафедре, за которой закреплен данный проверяющий, подразумевающий наличие возможностей преподавателей (получение списка анкет и данных по конкретным анкетам);
- возможность подтверждения и отмены подтверждения анкет.

Проверяющий не имеет доступа вообще ко всем анкетам, занесенных в систему. Его доступ ограничивается кафедрой, за которой он закреплен. Однако стоит учесть следующие ситуации:

- пользователь одновременно является и преподавателем и проверяющим;
- пользователь является проверяющим сразу на нескольких кафедрах.

Обе этих ситуации помогут понять, что роль пользователя в системе определяется составным способом. Предлагается решение при котором существует набор сущностей, в которые пользователь может быть независимо включен. Для сущности “Преподаватель” логично, что включение конкретного пользователя допустимо единожды. А сущность “Проверяющий” должна позволять закреплять за одним пользователем несколько связей. Здесь под связью понимается кафедра

и соответствующие права по осуществлению деятельности проверяющего внутри этой кафедры. Такой подход позволит гибко настроить то, что пользователю допустимо, а что запрещено.

2.2.4 Создание и редактирование анкет

Неразрывно с просмотром анкет связано их создание и редактирование. Пользователь должен иметь возможность создавать анкеты с указанием года, который характеризует отчетный период, и преподавателя, указывающего кому принадлежит данная анкета.

При указании преподавателя при создании анкеты:

- преподавателям разрешается указывать в качестве владельца только себя;
- проверяющим разрешается указывать в качестве владельца любого преподавателя с кафедры, за которую несет ответственность проверяющий.

Год должен быть уникальным среди существующих анкет для конкретного преподавателя. Таким образом, за год может быть подана только одна анкета от лица одного преподавателя. Дополнительно к этому ограничению, также вводятся верхняя и нижняя граница значения года. Нижняя граница задается статическим значением. Например, это может быть год основания университета. Верхняя граница – динамическая. Она всегда соответствует текущему году. Таким образом, при создании анкеты невозможно указать отчетный период, который на временном интервале находится в будущем.

Также с анкетой связан еще один параметр – статус подтверждения. При создании он всегда должен быть установлен в отрицательное значение. Попытка создать заведомо подтвержденную анкету должна приводить к отказу в доступе к такому действию.

Редактирование анкеты подразумевает под собой изменение значений показателей внутри неё. Для преподавателей существует ряд ограничений при редактировании анкет:

- некоторые показатели, называемые привилегированными, недоступны для редактирования преподавателями, их внесение и изменение происходит исключительно проверяющими (заведующими кафедрами);
- после утверждения анкеты преподавателю закрывается доступ к её редактированию, сам же проверяющий редактировать анкету так же не может,

однако имеет функцию снятия анкеты с подтверждения.

Набор показателей, которые включены в анкету для всех преподавателей должен задаваться администратором системы. Для самих показателей отдельно устанавливается тип их значения. Для каждой отдельной анкеты хранится именно значение показателя, а не критерия, так как один и тот же показатель может включаться сразу в несколько разделов рейтинга, характеризуя разные критерии. Следование этому требованию необходимо, так как:

- во-первых, снижается количество хранимых данных, тем самым уменьшается расход вычислительных ресурсов;
- во-вторых, данные обеспечиваются свойством целостности, путём исключения ситуаций, при которых для одного и того же показателя заданы различные значения, полученные от разных критериев.

2.2.5 Утверждение анкет

Заключительным сценарием является подтверждение анкет проверяющим. Проверяющему должна предоставляться функция подтверждения анкет. После подтверждения анкеты, её редактирование закрывается как для самого проверяющего, так и для преподавателя. Проверяющий имеет право снять анкету с подтверждения, однако у него должно быть основание для такого действия, коим может являться корректировка данных, после рассмотрения руководством служебной записи проверяющего, содержащую пояснение причины корректировки. Более специфических деталей для данного сценария не предусматривается.

2.3 Выбор общей архитектуры

До этого в работе уже отмечалось то, что система должна интегрироваться с другими системами. Учитывая набор требований, подходящим решением при реализации системы станет использование микросервисной архитектуры.

2.3.1 Плюсы микросервисной архитектуры

Именно такой подход обеспечит простоту в процессе дальнейшего развития системы на протяжении её эксплуатации внутри университета.

Компоненты микросервисной архитектуры работают независимо друг от

друга, что позволяет вносить изменения и внедрять новые функции без необходимости изменения всей системы. Это дает возможность свободно добавлять новые модули или обновлять существующие, не затрагивая другие компоненты.

Разделение системы на микросервисы упрощает процесс разработки. Каждый микросервис может быть разработан и оттестирован независимо, а затем легко интегрирован в общую систему. Это также позволяет разным разработчикам работать параллельно над различными компонентами системы, повышая эффективность разработки.

Микросервисная архитектура позволяет использовать разные технологии и языки программирования для различных модулей. Это дает возможность выбрать наиболее подходящие технологии и инструменты для каждой задачи. Например, можно использовать удобный при работе с данными Python для модуля аналитики, и высокопроизводительный GoLang для модуля расчёта и обработки собранных данных.

Также важно отметить вопрос оборудования университета, на котором будет производиться развёртывание системы. Не все университеты имеют достаточно мощные сервера для того, чтобы поддерживать работу монолитных систем, которые имеют перспективу роста. Разделение нагрузки при использовании отдельных сервисов решает данный вопрос.

2.3.2 Проблемы микросервисной архитектуры

Для удобного развёртывания сервисов требуются специальные инструменты, которые упрощают этот процесс. Одной из технологий, которая настоятельно рекомендуется к использованию в современной разработке и будет задействована в данном проекте, является контейнеризация. Если все сервисы в рамках системы, контейнеризованы, то управление ими упрощается, а также открывается возможность настройки их оркестрации. Тем не менее вопрос развёртывание всё еще остаётся сложным.

Мониторинг системы имеет важную роль. С ростом количества сервисов его значение усиливается. На начальной стадии проекта с крайне малым числом отдельных сервисов допускается их мониторинг по отдельности. Поэтому, при разработке системы не будут рассматриваться отдельные средства совокупного мониторинга всей системы. Однако важно понимать, что, чем раньше эти сред-

ства будут добавлены в систему, тем надежнее будет её дальнейшее развитие и эксплуатация.

В микросервисной архитектуре каждый микросервис может иметь свою собственную версию. Планирование и координация обновлений микросервисов может быть сложной задачей, особенно при наличии зависимостей между ними.

2.4 Выделение сервисов

После того, как выбрана общая архитектура построения всей системы, необходимо установить её внутренние модули, которые в концепте микросервисной архитектуры именуются сервисами.

2.4.1 Сервис клиента

Данный сервис представляет из себя веб-приложение, которое реализует графический интерфейс для пользователей. Пользователи, открывая сайт в своем веб-браузере, взаимодействуют с интерфейсом приложения именно этого сервиса. Все запросы, которые отправляет пользователь, сначала поступают на обработку в этот сервис. После получения и обработки запроса от пользователя, сервис клиента взаимодействует с сервисом API. Получив ответ от сервиса API, сервис клиента обрабатывает полученные данные и отображает их пользователю.

2.4.2 Сервис API

Сервис API является серверной частью системы. Его основной задачей является обработка запросов связанных с манипуляцией данными. Сервис API предоставляет REST API для взаимодействия с другими сервисами. REST API реализует все необходимые процедуры для работы с данными, такие как создание, чтение, обновление и удаление (CRUD-операции). Внешние сервисы могут отправлять HTTP-запросы на определенные конечные точки сервиса API для выполнения соответствующих операций с данными.

Сервис API также отвечает за обработку ошибок и возврат соответствующих сообщений. Сюда же относятся механизмы регистрации и авторизации, которые обеспечивают безопасный доступ к данным.

Важно отметить, что сам сервис API не хранит данные на своей стороне.

Он взаимодействует с сервисом базы данных, который отвечает за хранение и управление данными. Когда сервис API получает запросы, он выполняет соответствующие операции с данными, подключаясь к сервису базы данных.

2.4.3 Сервис базы данных

Внутри данного сервиса находится СУБД и сама база данных. Выделение базы данных в отдельный сервис является хорошей практикой, которая обеспечивает расширяемость, повышает безопасность и упрощает архитектуру системы.

2.4.4 Сервис брокера сообщений

При подтверждении анкеты её данные должны отправляться в систему 1С, управляемую УМУ. Для того чтобы обеспечить надёжную доставку этих данных сервис API отправляет данные не напрямую, а через брокер сообщений. К этому же сервису подключен внешний модуль для интеграции 1С. Этот модуль прослушивает очередь сообщений и при появлении в ней данных о новых анкетах, он выполняет их доставку в 1С.

2.5 Проектирование базы данных

Для начала предлагается выделить набор основных сущностей и связей между ними в системе. Представим их в виде простого текстового описания.

2.5.1 Выделение сущностей

Имеется «Пользователь», который может зарегистрироваться только если адрес его электронной почты находится в списке «Разрешённых Электронных Адресов». Он может быть добавлен в группу «Преподавателей» или «Проверяющих». Каждый преподаватель имеет «Квалификацию» и «Кафедру» на которой он преподаёт. Каждая «Кафедра» имеет свой «Тип» (общенаучная, выпускающая и т. д.) и «Факультет» на котором она расположена. Ежегодно «Преподаватель» создает «Анкеты». За всеми анкетами закрепляется общий набор «Критериев» по которым оцениваются преподаватели. «Критерий» подразумевает под собой «Показатель», включённый в определенный «Раздел Рейтинга». У каждого показателя есть свой «Тип Значения». «Преподаватели» вводят «Значения Показателей».

После такого описания можно набор сущностей можно представить в виде следующего списка: пользователь, разрешенные почтовые адреса, преподаватель, квалификация, проверяющий, кафедра, факультет, анкета, критерий, показатель, раздел рейтинга, тип значения показателя, значение показателя.

2.5.2 Определение атрибутов сущностей

Теперь определим основные атрибуты для каждой из сущности.

- пользователь (user profile) – фамилия, имя, отчество, электронная почта, пароль, статус персонала (администратора);
- разрешенные почтовые адреса (allowed email) – адрес электронной почты, табельный номер сотрудника;
- преподаватель (educator) – ссылка на пользователя, ссылка на квалификацию, ссылка на кафедру;
- квалификация (qualification) – название;
- проверяющий (educator report controller) – ссылка на пользователя, ссылка на кафедру;
- кафедра (department) – название, ссылка на тип кафедры, ссылка на заведующего (пользователя), ссылка на факультет;
- тип кафедры (department type) – название;
- факультет (faculty) – название, ссылка на декана (пользователя);
- анкета (educator report) – глобально уникальный идентификатор, год, ссылка на преподавателя, статус подтверждения;
- раздел рейтинга (educator rating partition) – название, шифр;
- показатель (indicator) – название, ссылка на тип значения, аннотация (пояснение), статус привилегированности;
- критерий (criterion) – ссылка на раздел рейтинга, ссылка на показатель, пункт в номере, подпункт в номере, вес критерия для раздела;
- значение показателя (educator indicator value) – ссылка на показатель, значение, ссылка на анкету.

2.5.3 Схема отношений сущностей

На рисунке 2.2 представлена итоговая схема отношений между сущностями.

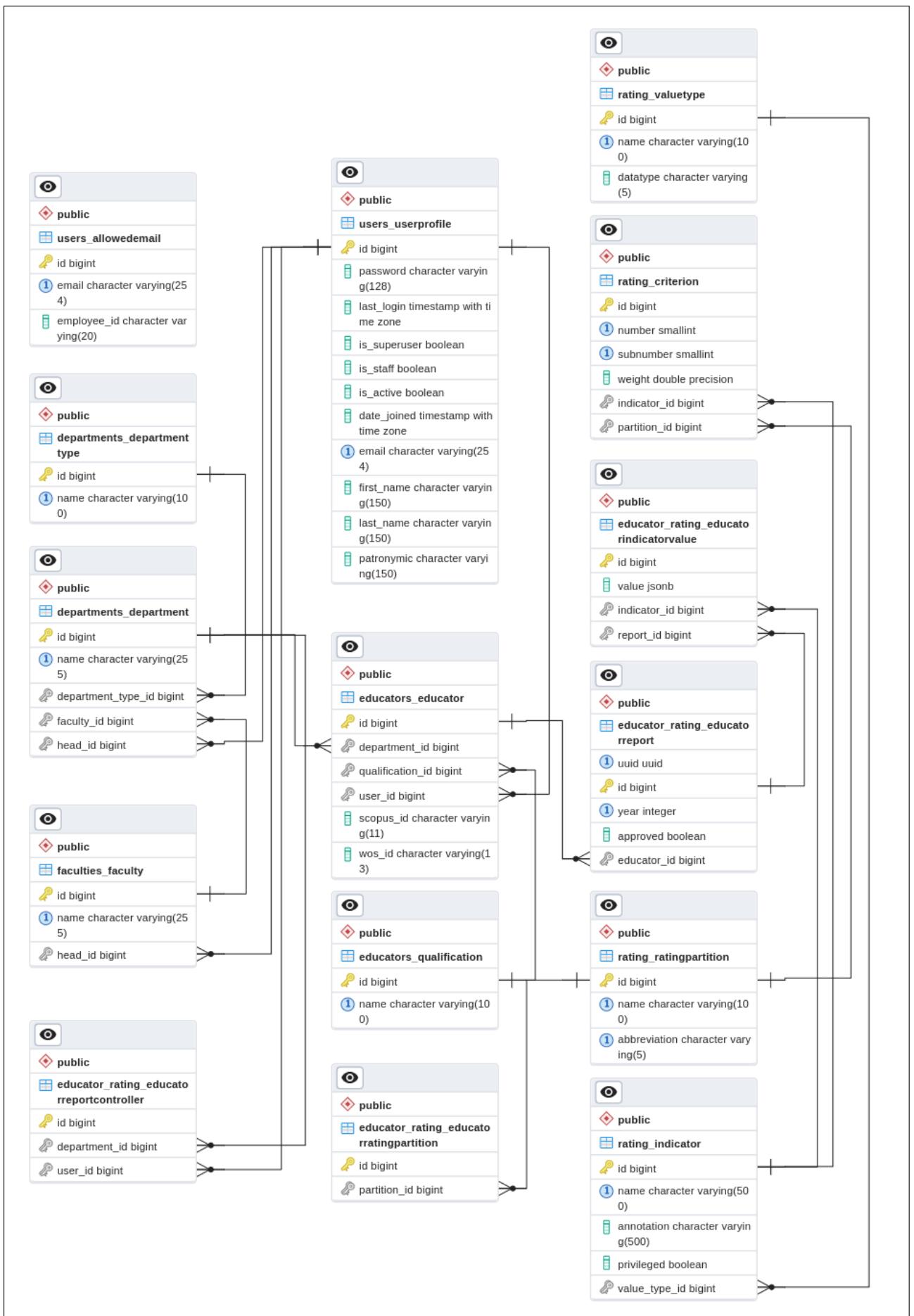


Рисунок 2.2 – Схема отношений сущностей

3 РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ СИСТЕМЫ

3.1 Выбор инструментов разработки

3.1.1 Выбор языка программирования

В качестве языка программирования для реализации серверной части системы, то есть для создания веб-API, выбран Python. Этот выбор можно обосновать по следующим причинам:

- Простота и читаемость кода – Python известен своей простотой и понятностью. Его синтаксис и структура делают код легким для чтения и понимания. Это особенно важно при разработке API, поскольку хорошо структурированный и читаемый код облегчает сопровождение и развитие системы.
- Большое сообщество и обширная экосистема – Python имеет одно из самых активных сообществ разработчиков. Большой выбор готовых решений позволяет сэкономить время и усилия при разработке.
- Множество библиотек и фреймворков – Python имеет обширную библиотеку стандартных модулей, а также множество сторонних библиотек и фреймворков, которые упрощают разработку API. Например, Flask и Django являются популярными фреймворками для разработки веб-приложений на Python, и они предлагают множество инструментов для создания и развертывания API.

3.1.2 Выбор фреймворка

Рассмотрим несколько популярных веб-фреймворков Python:

- 1) Django - это полноценный веб-фреймворк, который предоставляет мощные инструменты для разработки веб-приложений. Он включает ORM, систему маршрутизации, аутентификацию, панель администрирования и многое другое.
- 2) Flask - это легковесный веб-фреймворк, который фокусируется на простоте и гибкости. Он предлагает основные инструменты для разработки веб-приложений, такие как маршрутизация, обработка запросов и шаблоны.

низация. Flask позволяет разработчикам выбирать и интегрировать необходимые компоненты и библиотеки.

- 3) FastAPI - это современный и высокопроизводительный веб-фреймворк Python для создания веб-приложений и API. Он основан на асинхронной модели Python и предлагает простой и интуитивно понятный синтаксис. FastAPI обладает высокой производительностью благодаря асинхронному коду. Он также предоставляет автоматическую валидацию данных, типовую безопасность и генерацию интерактивной документации на основе стандарта OpenAPI. FastAPI имеет обширную экосистему плагинов и интегрируется с популярными инструментами и библиотеками Python, делая разработку веб-приложений более эффективной и удобной.

Выбор Django основывается на следующих факторах:

- 1) Мощность и полнота: Большое количество встроенного функционала Django позволяет быстро создавать сложные и масштабируемые веб-приложения.
- 2) Большое сообщество и поддержка: Django имеет большое и активное сообщество разработчиков, что обеспечивает доступ к множеству ресурсов, документации, обучающих материалов и сторонних пакетов.
- 3) Безопасность: Django предоставляет множество встроенных механизмов безопасности, включая защиту от CSRF (межсайтовая подделка запроса), аутентификацию и авторизацию пользователей, обработку форм и предотвращение уязвимостей. Django активно поддерживается и обновляется, что позволяет быть уверенным в его безопасности и защите от известных уязвимостей.
- 4) Оптимизация: Django имеет встроенные механизмы для оптимизации производительности, такие как кэширование запросов, компиляция шаблонов, ленивая загрузка и другие виды оптимизации. Он также поддерживает масштабирование приложений на различных уровнях, позволяя распределить нагрузку и обеспечить высокую производительность.
- 5) Документация: Django обладает хорошо структурированной и обширной официальной документацией, которая облегчает изучение и использование фреймворка.

3.1.3 Брокер сообщений

В качестве брокера сообщений выбран RabbitMQ. RabbitMQ является одним из наиболее популярных брокеров сообщений и имеет богатый набор клиентских библиотек. Это делает его легко интегрируемым в приложение, независимо от того, на каком языке оно написано.

3.1.4 Среда развёртывания

Как уже было сказано ранее, для упрощения процесса развёртывания используется технология контейнеризации. Стандартом де facto в этой сфере является инструмент под названием Docker.

Приведем основные преимущества использования Docker:

- 1) Универсальность и переносимость: Docker контейнеры предоставляют унифицированное окружение, которое легко переносить между различными системами. Это позволяет обеспечить одинаковую конфигурацию и запуск приложения на разных платформах, включая разработку и тестирование. Docker обеспечивает целостность и позволяет избегать проблем, связанных с совместимостью между окружениями разработчиков и окружениями реального развёртывания, что существенно упрощает процесс развертывания и управления приложением.
- 2) Изолированность: Docker обеспечивает изоляцию приложения и его зависимостей в контейнере. Это означает, что каждое приложение и его компоненты работают в своей собственной виртуальной среде, не влияя на другие приложения или систему в целом. Это повышает безопасность и предотвращает возможные конфликты между зависимостями разных приложений.
- 3) Упрощенное развертывание и масштабирование: Docker позволяет упаковать приложение, его зависимости и конфигурацию в контейнер, который легко развертывать и масштабировать. Контейнеры Docker могут быть легко развернуты на любом хосте, который поддерживает Docker. Это облегчает процесс развертывания приложения как на локальной машине разработчика, так и на серверах в рабочей среде.
- 4) Масштабируемость и гибкость: Docker обладает мощными возможностями для управления контейнерами, что позволяет легко масштабировать приложение в зависимости от нагрузки.

стями масштабирования и оркестрации, такими как Docker Compose и Kubernetes. Это позволяет горизонтально масштабировать приложение, запуская несколько контейнеров и распределяя нагрузку между ними.

В целом, использование Docker в проекте позволяет достичь высокой степени переносимости, изолированности, простоты развертывания и управления зависимостями, а также облегчает масштабирование и оркестрацию приложения. Это упрощает процесс разработки, развертывания и сопровождения приложения, а также способствует его надежности, безопасности и эффективности.

3.2 Подготовка рабочего окружения

3.2.1 Выбор и настройка менеджера пакетов

В качестве менеджера пакетов вместо стандартного Pip выбран Poetry. Ниже перечислены основные причины такого выбора:

- 1) Poetry предоставляет более продвинутый и удобный способ управления зависимостями в Python-проектах. Он использует файл `pyproject.toml` для описания зависимостей и их версий, что позволяет легко контролировать и обновлять пакеты. Poetry также предоставляет возможность создания виртуальных сред, что облегчает изоляцию зависимостей для разных проектов.
- 2) Poetry позволяет сохранять состояние окружения в lock-файле (`poetry.lock`). Это означает, что можно точно воспроизвести окружение с помощью одного файла, который содержит версии всех установленных пакетов и их зависимостей. Это полезно при совместной работе над проектом или при развертывании приложения на другой машине.
- 3) Poetry предоставляет более удобные инструменты для управления версиями пакетов. Имеется возможность определить диапазон версий для каждой зависимости и использовать команды Poetry для обновления или установки конкретной версии пакета. Это помогает избежать проблем совместимости и облегчает обновление зависимостей.
- 4) Poetry упрощает процесс установки зависимостей и сборки проекта. Он автоматически устанавливает все необходимые пакеты из указанных зависимостей и выполняет необходимые шаги сборки (если такие име-

ются) для проекта. Это позволяет сократить время настроек и упростить процесс развертывания проекта.

- 5) В Poetry имеется функциональность группировки зависимостей, которая позволяет легко отделять зависимости, необходимые только во время разработки, от зависимостей, необходимых для работы приложения в режиме реальной эксплуатации. Это полезно, для избегания установки и поддержки ненужных пакетов в рабочем окружении.

Poetry можно установить как пакет при помощи Pip. В создании виртуального окружения на хостовой машине разработчика нет необходимости, однако это даёт большие преимущества в статическом анализе кода. Интегрированная среда разработки, в которую были добавлены сведения о используемом виртуальном окружении, значительно упрощает процесс написания кода.

3.2.2 Используемые пакеты

После установки менеджера пакетов Poetry можно приступить к добавлению в проект внешних зависимостей. Перечислим основные используемые пакеты:

- 1) Django – это высокоуровневый веб-фреймворк, предоставляющий удобные инструменты для разработки мощных и масштабируемых веб-приложений.
- 2) Django REST Framework – это расширение для фреймворка Django, которое предоставляет набор инструментов и функциональность для разработки веб-сервисов API (интерфейсов программирования приложений) на основе принципов REST (Representational State Transfer). Фреймворк упрощает создание, сериализацию и валидацию данных, обработку запросов и ответов API, а также предоставляет множество полезных функций, таких как аутентификация, авторизация, пагинация и фильтрация данных.
- 3) Django REST Framework SimpleJWT – это расширение для Django REST Framework, которое предоставляет простой и удобный способ реализации аутентификации и авторизации с использованием JSON Web Tokens (JWT) в веб-сервисах API.
- 4) Djoser – это библиотека, предназначенная для облегчения и ускорения

процесса разработки аутентификации и управления пользователями в веб-приложениях на основе Django и Django REST Framework. Djoser предоставляет готовые API-эндпоинты для регистрации пользователей, входа, выхода, восстановления пароля и других операций, связанных с управлением пользователями. Библиотека также предоставляет настраиваемые опции для управления электронными письмами, шаблонами, правами доступа и другими функциями, связанными с пользовательскими аккаунтами.

- 5) Django CORS Headers – это пакет расширения для фреймворка Django, который обеспечивает управление CORS (Cross-Origin Resource Sharing) – механизмом безопасности браузера, позволяющим контролировать доступ к ресурсам на сервере из других источников (доменов). При разработке веб-приложений, особенно с использованием клиентской части на другом домене, CORS может представлять преграду для обмена данными между разными источниками. Django Cors Headers упрощает установку и настройку правил CORS в Django-проектах.
- 6) Django Jazzmin – это пакет расширения для фреймворка Django, предназначенный для создания современной и интерактивной панели администрирования для веб-приложений, построенных на Django. По умолчанию Django предоставляет стандартную административную панель, которая является функциональной, но не всегда удобной в использовании и не обладает современным дизайном. Django Jazzmin вносит значительные улучшения в пользовательский интерфейс панели администрирования, предоставляя более современный внешний вид и дополнительные возможности.
- 7) Psycopg2-binary – это бинарная реализация библиотеки Psycopg2, которая является популярным драйвером для работы с базами данных PostgreSQL в языке программирования Python. Psycopg2-binary обеспечивает простую и эффективную связь с PostgreSQL и позволяет разработчикам выполнять различные операции с базой данных.
- 8) Pika – это популярная библиотека Python, которая предоставляет простой и эффективный способ для взаимодействия с RabbitMQ. Pika имеет полную поддержку протокола AMQP (Advanced Message Queuing Protocol), который является стандартом для систем брокеров сообщений.

9) Gdown – это инструмент командной строки и библиотека Python, предназначенные для загрузки файлов с Google Drive. Он облегчает процесс загрузки файлов, предоставляя простой и удобный способ скачивания файлов с платформы Google Drive на локальную машину. В проекте данный пакет используется для предзагрузки записей в базу данных.

К зависимостям, используемым в среде разработки относятся следующие пакеты:

- 1) Flake8 – это инструмент статического анализа кода для языка программирования Python. Он предоставляет возможность автоматической проверки и обнаружения различных стилевых и синтаксических ошибок в исходном коде Python.
- 2) Drf-yasg (Django Rest Framework Yet Another Swagger Generator) – это пакет расширения для Django REST Framework, который обеспечивает автоматическую генерацию документации API в формате Swagger/OpenAPI на основе кода проекта Django REST Framework.
- 3) Django-Extensions – это популярное расширение для фреймворка Django, которое предоставляет дополнительные функциональные возможности и инструменты разработки для упрощения создания веб-приложений на базе Django.

3.2.3 Определение переменных окружения

Для конфигурирования среды используются файлы, содержащие определения переменных окружения, разделенные на две категории. Первая категория используется на этапе разработки, вторая – при развёртывании в режиме эксплуатации. Далее эти категории будут называться как dev-конфигурация и prod-конфигурация. Внутри каждой категории содержатся файлы, каждый из которых объединяет в себе набор переменных окружения, относящихся к какому-то одному модулю. Разбиение производилось по принципу — отдельный файл на каждый сервис. Рассмотрим переменные окружения, определенные для сервиса API и базы данных.

Для сервиса API имеется следующий список:

- DEBUG – режим эксплуатации системы (1 - dev, 0 - prod);
- SECRET_KEY – секретный ключ django проекта;

- ALLOWED_HOSTS – список хостов, с которых допустимы запросы;
- DB_ENGINE – используемая СУБД;
- DB_NAME – название сервера СУБД;
- DB_USER – имя пользователя в СУБД;
- DB_PASSWORD – пароль пользователя в СУБД;
- DB_HOST – адрес сервера СУБД;
- DB_PORT – порт СУБД;
- DJANGO_SUPERUSER_EMAIL – имя суперпользователя при первом запуске;
- DJANGO_SUPERUSER_PASSWORD – пароль суперпользователя при первом запуске;
- EMAIL_HOST – адрес почтового сервера ;
- EMAIL_PORT – порт почтового сервера;
- EMAIL_USE_TLS – безопасное соединение для почтового сервера;
- EMAIL_HOST_USER – почтовый ящик для рассылки;
- EMAIL_HOST_PASSWORD – пароль к почтовому ящику для рассылки;
- CORS_ALLOWED_ORIGINS – список хостов, с которых допустимы междоменные запросы;
- CLIENT_DOMAIN – адрес хоста с сервисом клиента (для почтовых уведомлений);
- SITE_NAME – название сайта клиента (для почтовых уведомлений);
- RABBITMQ_HOST – адрес хоста брокера сообщений;
- RABBITMQ_EXCHANGE – название обменника брокера сообщений;
- RABBITMQ_QUEUE – название очереди брокера сообщений;
- RABBITMQ_ROUTING_KEY – название ключа брокера сообщений;
- EMPLOYEES_API_URL – адрес API сервиса сотрудников;
- EMPLOYEES_API_USERNAME – логин авторизации сервиса сотрудников;
- EMPLOYEES_API_PASSWORD – пароль авторизации сервиса сотрудников.

Для сервиса базы данных список состоит из следующих переменных:

- POSTGRES_USER – имя пользователя в СУБД;
- POSTGRES_PASSWORD – пароль пользователя в СУБД;
- POSTGRES_DB – название сервера СУБД.

3.2.4 Написание Docker образа

Для сервиса базы данных используется официальный образ PostgreSQL, получаемый из открытого реестра Docker Hub.

Для сервиса API реализуется собственный образ, который строится из файла Dockerfile. В файле Dockerfile проекта задействуется подход мультиэтапного построения образа. Определено три основных этапа: base, builder и final. Разделение процесса построения на стадии позволяет снизить размер итогового образа, а также ускорить перепостроение образа. Рассмотрим структуру каждого этапа.

На первом этапе, код которого представлен на рисунке 3.1 происходит следующая цепочка событий:

- 1) указывается образ Python на базе Alpine Linux;
- 2) определяется аргумент DEPLOY_ENV, который отвечает за тип среды развёртывания (dev или prod);
- 3) устанавливаются переменные окружения, который влияют на процесс построения окружения.

```
1 FROM python:3.11-alpine as base
2
3 ARG DEPLOY_ENV
4
5 ENV DEPLOY_ENV=${DEPLOY_ENV} \
6     PYTHONUNBUFFERED=1 \
7     PYTHONFAULTHANDLER=1 \
8     PYTHONHASHSEED=random \
9     PIP_NO_CACHE_DIR=off \
10    PIP_DEFAULT_TIMEOUT=100 \
11    PIP_DISABLE_PIP_VERSION_CHECK=on \
12    POETRY_VERSION=1.4.0
13
```

Рисунок 3.1 – Первая стадия построения Docker образа

На втором этапе, код которого представлен на рисунке 3.2 определяется следующая последовательность действий:

- 1) создается рабочая директория;
- 2) копируются файлы со сведениями об окружении из директории хоста в директорию контейнера;
- 3) устанавливаются зависимости для сборки пакетов Python. Выполняются установка компилятора GCC, установка инструмента управления зависи-

мостями Poetry и установка зависимостей проекта, при этом учитывается значение переменной DEPLOY_ENV для определения необходимости пакетов для разработки.

```
14
15 FROM base as builder
16
17 WORKDIR /app/
18
19 COPY src/pyproject.toml src/poetry.lock /app/
20
21 RUN apk add gcc --no-cache gcc musl-dev python3-dev
22
23 RUN pip install poetry==${POETRY_VERSION} && \
24     poetry config virtualenvs.in-project true && \
25     poetry install $(test $DEPLOY_ENV = prod && echo "--without dev") --no-root
26     --no-interaction --no-ansi
```

Рисунок 3.2 – Вторая стадия построения Docker образа

Третий этап представлен на рисунке 3.3.

```
27
28 FROM base as final
29
30 # we need this for django-admin compilemessages
31 RUN apk add gettext
32
33 COPY --from=builder /app/.venv/ /app/.venv/
34 COPY src/ /app/src/
35
36 WORKDIR /app/src/
37
38 RUN chmod +x *.sh
39
40 RUN addgroup -S app && \
41     adduser -S app -G app -h /app -DH
42
43 USER app
44
45 ENTRYPOINT [ "./entrypoint.sh" ]
46
```

Рисунок 3.3 – Третья стадия построения Docker образа

Здесь происходит следующее:

- 1) устанавливается пакет gettext для использования локализации;
- 2) копируется виртуальное окружение из этапа builder;

- 3) копируется весь код проекта из директории хоста в директорию контейнера;
- 4) добавляются права на выполнение для всех сценариев оболочки;
- 5) создаются пользователь и группа с именем “app”;
- 6) пользователь “app” устанавливается в качестве пользователя по умолчанию для контейнера;
- 7) указывается точка входа в контейнер.

По результатам прохождения этапов создаётся оптимизированный Docker-образ, включающий зависимости проекта и настроенное окружение.

3.2.5 Написание сценария точки входа

Рассмотрим сценарий точки входа в контейнер. Весь сценарий можно условно разделить на пять этапов. Рассмотрим каждый из них.

- 1) Ожидание доступности базы данных (рисунок 3.4):
 - в цикле происходит проверка доступности базы данных с использованием команды “nc -z” для проверки соединения с хостом базы данных;
 - если соединение с базой данных не установлено, сценарий ожидает 1 секунду и повторяет проверку.
- 2) Активация окружения (рисунок 3.5):
 - проверяется наличие директории “.venv/” в родительской директории;
 - если директория “.venv/” существует, активируется виртуальное окружение с помощью команды “source ../venv/bin/activate”;
 - если директория “.venv/” не найдена, выводится сообщение об ошибке и сценарий завершается с кодом 1, что означает остановку контейнера.
- 3) Применение миграций базы данных (рисунок 3.6):
 - выполняется команда “python manage.py migrate” для применения миграций базы данных.
- 4) Создание суперпользователя (рисунок 3.7):
 - выполняется команда python “manage.py shell < create_superuser.py”, которая создает суперпользователя на основе кода, указанного в файле “create_superuser.py” (код файла указан на рисунке 3.8);
 - если команда выполнена успешно (код возврата 0), выводится сообщение об успешном создании суперпользователя;

- если суперпользователь уже существует (код возврата не равен 0), выводится сообщение о том, что суперпользователь уже существует.
- 5) Запуск сервера (рисунок 3.9):
- проверяется значение переменной DEPLOY_ENV;
 - если значение DEPLOY_ENV равно ”prod” выполняется проверка безопасности с помощью команды “python manage.py check –deploy” и сценарий завершается с кодом 0, по скольку данный режим на момент написания работы еще не реализован;
 - если значение DEPLOY_ENV не равно ”prod” выполняется команда “python manage.py runserver 0.0.0.0:8000” для запуска сервера разработки на порте 8000.

```

1  #!/bin/sh
2
3  echo "Waiting for database..."
4
5  while ! nc -z $DB_HOST $DB_PORT; do
6      sleep 1
7  done

```

Рисунок 3.4 – Фрагмент кода для ожидания доступности базы данных

```

9  echo "Activation environment..."
10
11 if [ -d "../.venv/" ]; then
12     source ../.venv/bin/activate
13 else
14     echo "No .venv/ found in the /app directory"
15     exit 1
16 fi
17

```

Рисунок 3.5 – Фрагмент кода для активации окружения

```

18 echo "Applying database migrations..."
19 python manage.py migrate
20 echo "You can use load-fixture.sh script to prepopulate database with some values"
21 echo "You can use localize.sh script to switch to presented locale"
22

```

Рисунок 3.6 – Фрагмент кода для применения миграций

```

23 echo "Creating superuser..."
24 python manage.py shell < create_superuser.py
25
26 if [ $? = 0 ]; then
27 |   echo "Superuser '$DJANGO_SUPERUSER_EMAIL' created successfully"
28 else
29 |   echo "Superuser '$DJANGO_SUPERUSER_EMAIL' is already exists"
30 fi
31

```

Рисунок 3.7 – Фрагмент кода для создания суперпользователя

```

1 # instead of using python manage.py createsuper --no-input
2 # we use this script because we need to add superuser email
3 # to allowed list (create AllowedEmail instance) initially
4
5 import os
6
7 from apps.users.models import AllowedEmail, UserProfile
8
9
10 email = os.environ.get('DJANGO_SUPERUSER_EMAIL')
11 password = os.environ.get('DJANGO_SUPERUSER_PASSWORD')
12
13 try:
14     AllowedEmail.objects.get(email=email)
15     exit(1)
16 except AllowedEmail.DoesNotExist:
17     AllowedEmail.objects.create(email=email)
18     UserProfile.objects.create_superuser(
19         email=email,
20         password=password,
21         is_superuser=True,
22         is_staff=True,
23     )

```

Рисунок 3.8 – Сценарий создания суперпользователя

```

32 echo "Running server ..."
33
34 if [ "$DEPLOY_ENV" = "prod" ]; then
35     echo "Using production mode"
36     echo "Performing security checks..."
37     python manage.py check --deploy
38     echo "Not implemented"
39     exit 0
40 else
41     echo "Using development mode"
42     python manage.py runserver 0.0.0.0:8000
43 fi
44

```

Рисунок 3.9 – Фрагмент кода для запуска сервера

3.2.6 Написание вспомогательных сценариев

Рассмотрим вспомогательные сценарии, используемые внутри контейнера.

Для активации виртуального окружения используется сценарий “activate.sh”, код которого показан на рисунке 3.10.

```
1  #!/bin/sh
2
3  source ../venv/bin/activate
```

Рисунок 3.10 – Сценарий для активации виртуального окружения

Для применения имеющейся локализации к проекту используется сценарий “localize.sh”, код которого показан на рисунке 3.11.

```
1  #!/bin/sh
2
3  source activate.sh
4  django-admin compilemessages
```

Рисунок 3.11 – Сценарий для применения локализации

Для предзаполнения базы данных записями используется сценарий “load-fixture.sh”, код которого показан на рисунке 3.12.

```
1  #!/bin/sh
2
3  if [ $1 ]; then
4      source activate.sh &&
5      gdown $1 -O fixture.json &&
6      python manage.py loaddata fixture.json
7  else
8      echo "Get fixture file from google drive and load it to database."
9
10     echo "Use following format: ./load-fixture.sh <fixture file id>"
11 fi
```

Рисунок 3.12 – Сценарий для предзаполнения базы данных

3.2.7 Написание конфигурации docker-compose

Так при разработке необходимо запускать одновременно несколько контейнеров, а также развёртывание системы потенциально может осуществляться на одном хосте, принято решение использовать в проекте инструмент docker-compose.

Docker-compose – это инструмент, который используется для определения и управления множеством контейнеров Docker. Он позволяет объединить несколько контейнеров в единую группу, которая может быть запущена и остановлена одновременно с помощью простых команд.

Основная цель docker-compose – облегчить развертывание и управление сложными приложениями, состоящими из нескольких сервисов и компонентов.

Docker-compose предоставляет простой способ описания и настройки многоконтейнерных приложений. Вместо того чтобы создавать и настраивать каждый контейнер отдельно, имеется возможность определить все контейнеры, их зависимости, настройки и сетевые параметры в файле `docker-compose.yaml`. Затем, используя интерфейс утилиты docker-compose, можно легко управлять этой группой контейнеров.

Используются две разные конфигураций “`docker-compose.prod.yaml`” и “`docker-compose.dev.yaml`”. Каждая из этих конфигураций предназначена для работы в разных окружениях: эксплуатации (production) и разработки (development). Между этими конфигурациями имеются следующие отличия:

1) конфигурация эксплуатации:

- использование оптимизированных образов Docker;
- настройки масштабирования и балансировки нагрузки, чтобы обеспечить отказоустойчивость и эффективное использование ресурсов;
- конфигурация сети для обеспечения изоляции и безопасности;

2) конфигурация разработки:

- использование образов Docker с инструментами разработки;
- монтирование локальной файловой системы в контейнеры, чтобы обеспечить мгновенные изменения кода без пересборки образа;
- включение дополнительных сервисов, таких как мониторинговые инструменты, полезных для разработки и отладки.

На момент написания работы используется конфигурация “`docker-compose.dev.yaml`”, рассмотрим её структуру в качестве примера:

1) сервис `api` (рисунок 3.13):

- `image`: задает название docker-образа для сервиса;
- `container_name`: задает имя контейнера для сервиса;
- `build`: определяет настройки для сборки docker-образа: контекст сборки указан как “`./api/`”, аргумент `DEPLOY_ENV` установлен в “`dev`”;

- ports: определяет проброс портов между контейнером и хостом. В данном случае, порт 8000 контейнера пробрасывается на порт 8000 хоста;
 - volumes: монтирует локальную директорию “./api/src/” внутрь контейнера, чтобы обеспечить синхронизацию кода между хостом и контейнером;
 - env_file: указывает на файл с переменными среды, которые будут доступны внутри контейнера;
 - depends_on: задает зависимость от другого сервиса с именем db.
- 2) сервис db (рисунок 3.14, параметры аналогичные сервису api);
- 3) сервис pgadmin (рисунок 3.15):
- restart: always: Задает автоматическую перезагрузку контейнера в случае его остановки.
- 4) сервис rabbitmq (рисунок 3.16, параметры аналогичные сервису api);
- 5) сервис client (рисунок 3.17, параметры аналогичные сервису api);

```

1 version: '3'
2
3 services:
4   api:
5     image: university-rating-system-api:latest
6     container_name: urs-api-dev
7     build:
8       context: ./api/
9       args:
10      - DEPLOY_ENV=dev
11     ports:
12      - "8000:8000"
13     volumes:
14      - ./api/src/:/app/src/
15     env_file:
16      - ./env/dev/api.dev.env
17     depends_on:
18       - db

```

Рисунок 3.13 – Конфигурация сервиса api

```

19   db:
20     image: postgres:15.2-alpine
21     container_name: postgres-dev
22     volumes:
23       - pg_data:/var/lib/postgres/data/
24     env_file:
25       - ./env/dev/postgres.dev.env

```

Рисунок 3.14 – Конфигурация сервиса db

```

26 pgadmin:
27   container_name: pgadmin-dev
28   image: dpage/pgadmin4
29   restart: always
30   environment:
31     PGADMIN_DEFAULT_EMAIL: admin@example.com
32     PGADMIN_DEFAULT_PASSWORD: admin
33   ports:
34     - "5050:80"

```

Рисунок 3.15 – Конфигурация сервиса pgadmin

```

35 rabbitmq:
36   image: rabbitmq:3.12.0-management
37   container_name: rabbitmq-dev
38   ports:
39     - 5672:5672
40     - 15672:15672

```

Рисунок 3.16 – Конфигурация сервиса rabbitmq

```

35 client:
36   image: university-rating-system-client:latest
37   container_name: urs-client-dev
38   build:
39     context: ./client/
40     args:
41       - DEPLOY_ENV=dev
42   ports:
43     - "3000:3000"
44   volumes:
45     - ./client/src/:/app/src/
46     - ./client/public/:/app/public/
47   env_file:
48     - ./env/dev/client.dev.env
49
50   volumes:
51     pg_data:

```

Рисунок 3.17 – Конфигурация сервиса client

3.3 Создание моделей данных

После настройки рабочего окружения можно приступить к написанию кода. Было принято решение начать разработку с программного описания моделей

данных. Для этого используется ORM встроенное в Django. Структура Django-проекта разделена на 8 приложений: “core”, “users”, “educators”, “departments”, “faculties”, “rating”, “educator-rating”, “integration_1c”. Для каждого из приложений за исключением “core” и “integration_1c”, определяется собственный набор моделей.

3.3.1 Примеры моделей данных

Так как все модели описываются одинаковым способом, и имеют различие лишь в наборе полей, то как пример рассмотрим лишь модели из приложения “educators”. Их описание показано на рисунках 3.18 и 3.19.

```
8  class Qualification(models.Model):
9      """Model represents educator qualification."""
10
11     name = models.CharField(
12         verbose_name=_('qualification name'),
13         max_length=100,
14         unique=True
15     )
16
17     You, 2 months ago | 1 author (You)
18     class Meta:
19         verbose_name = _('qualification')
20         verbose_name_plural = _('qualifications')
21
22     def __str__(self) -> str:
23         return self.name
24
```

Рисунок 3.18 – Модель данных квалификации

```
25    class Educator(models.Model):
26        """Model represents university educator."""
27
28        user = models.OneToOneField(
29            verbose_name=_('educator profile'),
30            to=UserProfile,
31            on_delete=models.PROTECT
32        )
33        qualification = models.ForeignKey(
34            verbose_name=_('educator qualification'),
35            to=Qualification,
36            on_delete=models.PROTECT
37        )
38        department = models.ForeignKey(
39            verbose_name=_('educator department'),
40            to=Department,
41            on_delete=models.PROTECT
42        )
```

Рисунок 3.19 – Модель данных преподавателя

3.3.2 Переопределение стандартной модели пользователя

Так как Django определяет стандартную модель пользователя с набором полей, который не всегда соответствует нуждам, предлагается её переопределение. Рассмотрим код самой модели и её менеджера, представленные на рисунках 3.20 и 3.21 соответственно.

```
59  class UserProfile(AbstractUser):
60      """Custom user model which uses email as username and has `patronymic`  

61      `first_name`, `last_name` mandatory fields.  

62      """  

63      username = None  

64  

65      email = models.EmailField(  

66          verbose_name=_('email address'),  

67          unique=True,  

68          validators=[AllowedEmail.validate_email]  

69      )  

70      first_name = models.CharField(verbose_name=_('first name'), max_length=150)  

71      last_name = models.CharField(verbose_name=_('last name'), max_length=150)  

72      patronymic = models.CharField(  

73          verbose_name=_('patronymic'),  

74          max_length=150,  

75          blank=True  

76      )  

77  

78      USERNAME_FIELD = 'email'  

79      REQUIRED_FIELDS = []  

80  

81      objects = UserProfileManager()  

82  

83      def __str__(self) -> str:  

84          return f'{self.email}'
```

Рисунок 3.20 – Переопределение модели данных пользователя

В данном фрагменте кода показано переопределение стандартной модели пользователя Django путём наследования от абстрактного класса “AbstractUser”. В унаследованную стандартную модель пользователя вносятся изменения, описанные ниже.

- Стока “username = None” указывает, что поле “username” должно быть отключено для данной модели пользователя. Вместо этого будет использоваться поле email в качестве уникального идентификатора пользователя.
- Поле email определено как “models.EmailField” и используется в качестве уникального идентификатора пользователя. Оно также имеет параметр “unique=True”, чтобы гарантировать, что каждый пользователь имеет уникальный адрес электронной почты.

- Поля “first_name” и “last_name” определены как “models.CharField” и представляют собой обязательные поля для имени и фамилии пользователя.
- Поле “patronymic” определено как “models.CharField” и представляет собой дополнительное поле для отчества пользователя. Оно не является обязательным и может быть пустым (параметр “blank=True”).
- Стока “USERNAME_FIELD = ‘email’ ” указывает, что поле “email” будет использоваться в качестве уникального идентификатора пользователя при аутентификации.
- Список “REQUIRED_FIELDS” пуст, что означает, что при создании пользователя обязательными полями будут только “email”, “password” и “is_staff”.
- Поле “objects” указывает на менеджер объектов “UserProfileManager”, который будет рассмотрен позже.
- Метод “__str__” переопределен для модели пользователя, чтобы при преобразовании объекта в строку выводился адрес электронной почты.

```

30 class UserProfileManager(BaseUserManager):
31     def _create_user(self, email, password, **extra_fields):
32         """Create and save a user with the given email, and password."""
33         if not email:
34             raise ValueError("The given email must be set")
35         email = self.normalize_email(email)
36         user = self.model(email=email, **extra_fields)
37         user.password = make_password(password)
38         user.save(using=self._db)
39
40         return user
41
42     def create_user(self, email, password=None, **extra_fields):
43         extra_fields.setdefault('is_staff', False)
44         extra_fields.setdefault('is_superuser', False)
45         return self._create_user(email, password, **extra_fields)
46
47     def create_superuser(self, email, password=None, **extra_fields):
48         extra_fields.setdefault('is_staff', True)
49         extra_fields.setdefault('is_superuser', True)
50
51         if extra_fields.get('is_staff') is not True:
52             raise ValueError("Superuser must have is_staff=True.")
53         if extra_fields.get('is_superuser') is not True:
54             raise ValueError("Superuser must have is_superuser=True.")
55
56         return self._create_user(email, password, **extra_fields)
57

```

Рисунок 3.21 – Переопределение менеджера модели данных пользователя

Класс “UserProfileManager” является пользовательским менеджером модели и используется для управления объектами модели UserProfile. Он наследуется от базового класса “BaseUserManager”, предоставляемого Django, и позволяет

переопределить методы создания и управления пользователями. Для менеджера моделей были переопределены методы из списка ниже.

- Метод “`_create_user`” создает пользователя с указанным “`email`” и паролем. Если “`email`” не указан, вызывается исключение. Метод нормализует “`email`”, создает пользователя, хеширует пароль и сохраняет объект.
- Метод “`create_user`” создает обычного пользователя с указанным `email` и паролем. Он устанавливает значения по умолчанию (если они не заданы) для полей “`is_staff`” и “`is_superuser`” как “`False`” и вызывает “`_create_user`” для создания и сохранения пользователя.
- Метод “`create_superuser`” создает суперпользователя с указанным `email` и паролем. Аналогичен методу “`create_user`”, но устанавливает значения по умолчанию для полей “`is_staff`” и “`is_superuser`” как “`True`” и проверяет, что “`is_staff`” и “`is_superuser`” имеют значения “`True`”.

3.4 Регистрация моделей в панеле администратора

Следующим шагом является регистрация моделей в панеле администрирования. Рассмотрим её также на примере моделей приложения “`educators`”. Код фрагмента показан на рисунке 3.22.

```
7 @admin.register(Educator)
8 class EducatorAdmin(admin.ModelAdmin):
9     list_display = ('__str__', 'user', 'qualification', 'department', )
10    search_fields = (
11        'department__name', 'user__first_name', 'user__last_name',
12        'user__patronymic', 'user__email',
13    )
14    search_help_text = _(
15        ('Department name or educator email, '
16        '| first name, last name or patronymic')
17    )
18    list_filter = ('qualification__name', )
19    autocomplete_fields = ('user', 'qualification', 'department', )
20
21
22 You, 2 months ago | 1 author (You)
23 @admin.register(Qualification)
24 class QualificationAdmin(admin.ModelAdmin):
25     list_display = ('name', )
26     search_fields = ('name', )
27     search_help_text = _('Qualification name')
```

Рисунок 3.22 – Регистрация моделей приложения “`educators`” в панеле администрирования

Свойствами “list_display”, “search_fields” и другими задаются настройки отображения полей модели, поиска, фильтров и прочих элементов интерфейса.

3.5 Создание сериализаторов моделей

Сериализаторы в Django Rest Framework представляют собой ключевой компонент для преобразования сложных типов данных, таких как модели Django, в форматы, подходящие для передачи по сети, например JSON. Они обеспечивают сериализацию (преобразование в данные) и десериализацию (преобразование из данных) объектов, что позволяет удобно работать с данными при разработке веб-сервисов и API.

DRF предоставляет готовые инструменты для простого создания сериализаторов моделей веб-приложений. Он включает базовый класс “serializers.ModelSerializer”, на основании которого создаются сериализаторы, основанные на модели Django. Рассмотрим определение сериализаторов на примере приложения “educators” (рисунок 3.23).

В сериализаторе указывается используемая модель и поля, которые будут задействоваться при преобразовании.

```
1  from rest_framework.serializers import ModelSerializer
2
3  from .models import Educator, Qualification
4
5
6  You, 2 months ago | 1 author (You)
7  class EducatorSerializer(ModelSerializer):
8      You, 2 months ago | 1 author (You)
9          class Meta:
10              model = Educator
11              fields = ('id', 'user', 'qualification', 'department', )
12
13  You, 2 months ago | 1 author (You)
14  class QualificationSerializer(ModelSerializer):
15      You, 2 months ago | 1 author (You)
16          class Meta:
17              model = Qualification
18              fields = ('id', 'name', )
```

Рисунок 3.23 – Сериализаторы приложения “educators”

3.6 Создание представлений

Представления в Django Rest Framework представляют собой компоненты, которые определяют логику обработки запросов API.

Представления позволяют определить различные типы запросов и связать их с соответствующими операциями базы данных или другой логикой приложения. Они обеспечивают уровень абстракции, позволяющий обрабатывать запросы и возвращать соответствующие HTTP-ответы, включая данные в сериализованной форме.

Представления также предоставляют механизмы для авторизации, аутентификации, разрешения доступа и обработки ошибок, что позволяет легко реализовывать функции безопасности и управление доступом в API. Рассмотрим пример всё того же приложения “educators” (рисунок 3.24).

```
12  class EducatorViewSet(ReadOnlyModelViewSet):
13      queryset = Educator.objects.all().order_by('pk')
14      serializer_class = EducatorSerializer
15
16      @action(
17          detail=False,
18          methods=SAFE_METHODS,
19          permission_classes=(IsAuthenticated, IsEducatorUser, )
20      )
21      def me(self, request: Request) -> Response:
22          """Get educator instance of requested user."""
23
24          return Response(
25              EducatorSerializer(
26                  instance=Educator.objects.get(user=request.user)
27              ).data
28          )
29
30
31  You, 2 months ago | 1 author (You)
32  class QualificationViewSet(ReadOnlyModelViewSet):
33      queryset = Qualification.objects.all().order_by('pk')
34      serializer_class = QualificationSerializer
```

Рисунок 3.24 – Представления приложения “educators”

Как видно из рисунка, используются не простые представления, а наборы представлений (viewsets). Они представляют собой высокоуровневый компонент, который объединяет различные операции CRUD для модели в одном месте, что

является удобным и компактным способом определения логики обработки запросов API, минимизируя необходимость явного определения каждого отдельного представления.

Если возникает необходимость в определении для viewset'а дополнительной операции, то имеется возможность его создания путём использования действий (actions). В примере определяется действие “me”, которое возвращает запросившему пользователю его же преподавательские данные.

3.7 Создание пользовательских разрешений

Django Rest Framework уже содержит набор часто используемых разрешений, например, “IsAuthenticated” (пользователь авторизирован). Однако иногда возникает необходимость определить собственные наборы разрешений, которые будут использоваться представлениями. В примере с представлениями использовалось пользовательское разрешение “IsEducatorUser”, рассмотрим его в качестве примера (рисунок 3.25).

```
9  class IsEducatorUser(BasePermission):
10     message = _("You are not an educator.")
11
12     def has_permission(self, request: Request, view: View) -> bool:
13         try:
14             Educator.objects.get(user=request.user)
15             return True
16         except Educator.DoesNotExist:
17             return False
```

Рисунок 3.25 – Разрешение “IsEducatorUser”

Здесь можно наблюдать каким образом выполняется проверка разрешения у пользователя, запросившего ресурс.

3.8 Подключение представлений к маршрутизатору

После того как реализована логика обработки запросов, необходимо для каждого представления назначить собственный путь, по которому оно будет доступно. Для упрощения такой задачи используются маршрутизаторы (routers). Рассмотрим пример создания маршрутизатора и подключения к нему набора пред-

ствлений (рисунок 3.26), а также установки маршрутизатора для обработки маршрутов (рисунок 3.27).

```
17 # users app
18 api_router.register(
19     prefix='allowed-emails',
20     viewset=AllowedEmailViewSet,
21     basename='allowed-email'
22 )
23
24 # educators app
25 api_router.register(
26     prefix='educators',
27     viewset=EducatorViewSet,
28     basename='educator'
29 )
30 api_router.register(
31     prefix='qualifications',
32     viewset=QualificationViewSet,
33     basename='qualification'
34 )
35
```

Рисунок 3.26 – Создание и подключение к маршрутизатору представлений

```
12 from .routers import api_router
13
14 urlpatterns = [
15     path('auth/', include('rest_framework.urls', namespace='rest_framework')),
16     path('', include('djoser.urls')),
17     path('', include('djoser.urls.jwt')),
18     path('', include((api_router.urls, 'api'), namespace='api')),
19 ]
```

Рисунок 3.27 – Установка маршрутизатора для обработки маршрутов

3.9 Интеграция с системой 1С

Ранее описывалось что системы отсылает подтвержденные анкеты в брокер сообщений, а также может получать данные о сотрудниках при регистрации нового пользователя. Рассмотрим более подробно формат взаимодействия с системой 1С. На рисунке 3.28 изображена общая схема взаимодействия.

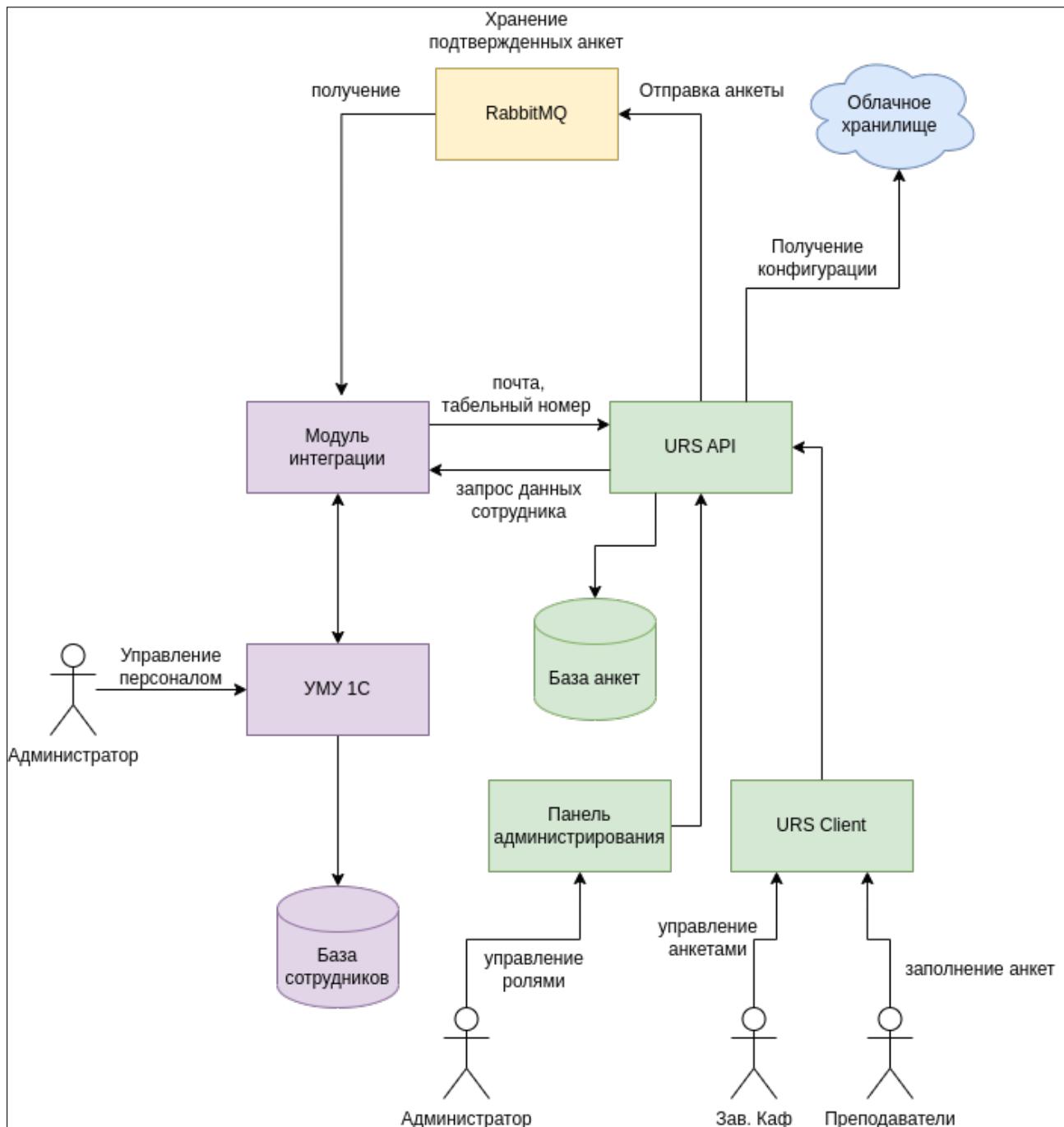


Рисунок 3.28 – Схема взаимодействия с системой 1С

3.9.1 Получение данных о сотруднике

Рассмотрим фрагмент кода, который обрабатывает сигнал создания пользователя. Он представлен на рисунке 3.29. При регистрации пользователя, в первую очередь происходит проверка указанной почты на предмет её присутствия в списке разрешенных. Вместе с этим там может храниться идентификатор сотрудника в системе 1С. Если он указан, то система обращается к модулю интеграции для получения данных, используя функцию “query_employee_data”. Её код показан

на рисунке 3.30. Если данные по указанному идентификатору сотрудника были возвращены, то происходит построение объектов зависимостей – “user_data” и “educator_data”. Если объекты построены успешно, то для пользователя устанавливается его фамилия, имя и отчество, а также в системе создаётся преподаватель с указанной квалификацией и закрепленный на определенной кафедре.

```
102 @receiver(models.signals.post_save, sender=UserProfile)
103 def add_data_from_integration_server(sender, instance, created, *args, **kwargs) -> None:
104     if not created:
105         return
106
107     employee_id = AllowedEmail.objects.get(email=instance.email).employee_id
108
109     if not employee_id:
110         return
111
112     employee_data = query_employee_data(employee_id)
113
114     if employee_data is None:
115         return
116
117     # To avoid circular dependencies imports goes here
118     from ..integration_1c.constructors import (construct_educator_data,
119                                                 construct_user_data,
120                                                 create_educator_hook)
121
122     user_data = construct_user_data(employee_data)
123
124     if user_data:
125         instance.last_name = user_data.last_name
126         instance.first_name = user_data.first_name
127         instance.patronymic = user_data.patronymic
128         instance.save(update_fields=['last_name', 'first_name', 'patronymic'])
129
130     educator_data = construct_educator_data(employee_data)
131
132     if educator_data:
133         create_educator_hook(
134             user=instance,
135             qualification=educator_data.qualification,
136             department=educator_data.department
137         )
```

Рисунок 3.29 – Обработка сигнала создания пользователя

```

17 def query_employee_data(employee_id: str) -> Optional[dict]:
18     """Fetch information about employee from api by his employee id."""
19     try:
20         url = f'{EMPLOYEES_API_URL}/RATING/hs/educatorData/{employee_id}/'
21         response = rq.get(url, headers=_HEADERS)
22         return response.json()
23     except rq.exceptions.RequestException:
24         return None
25
26
27 def query_mock_employee_data(employee_id: str) -> Optional[dict]:
28     return {
29         "fullName": "Тестовый Тест Тестович",
30         "qualificationName": "Доценты",
31         "departmentName": "ЭВМ",
32     }

```

Рисунок 3.30 – Функция получения данных о сотруднике

3.9.2 Отправка данных анкеты

При подтверждении анкеты, предполагается её отправка в брокер сообщений. На рисунке 3.31 показана последовательность действий связанная с анкетами.

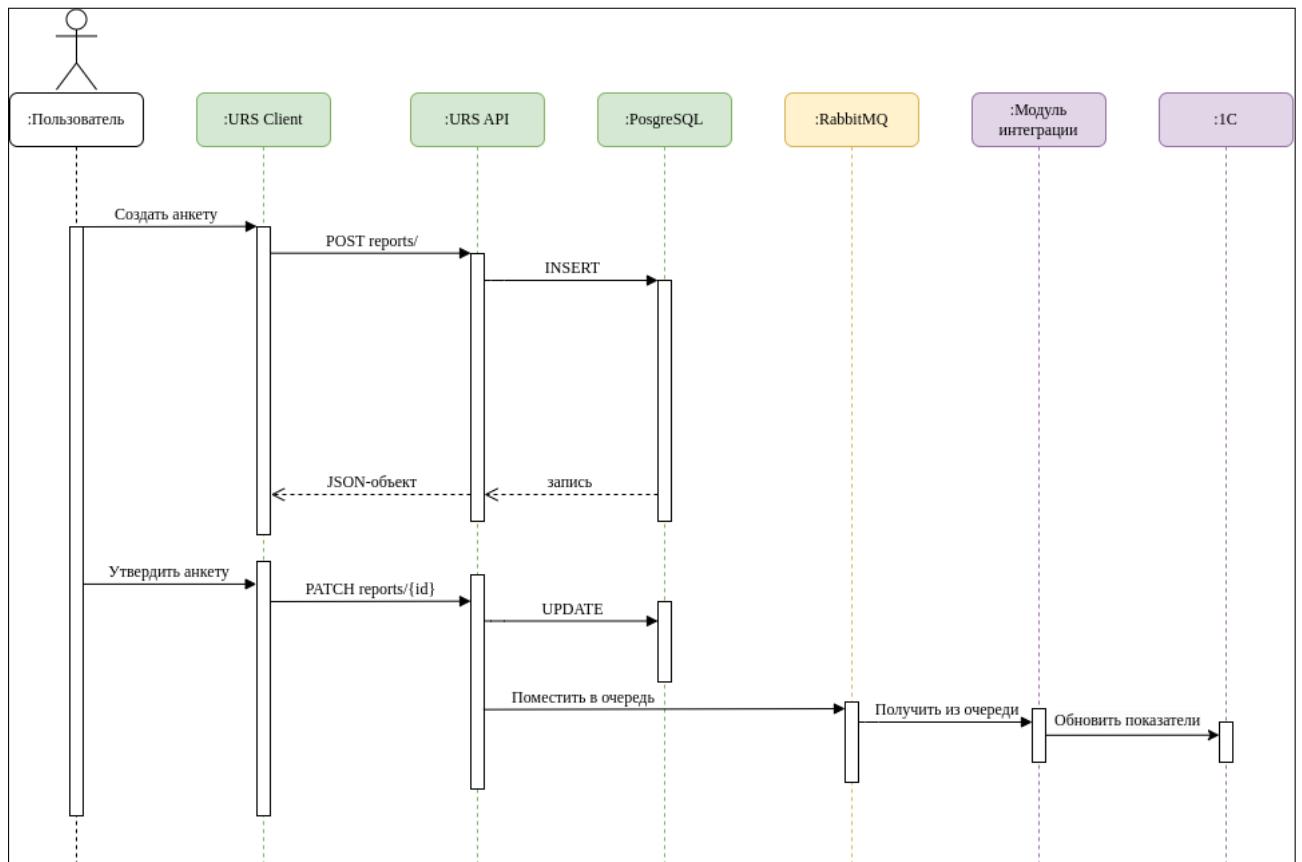


Рисунок 3.31 – Последовательность действий для анкет

Если анкеты подтверждается, то происходит вызов функции, код которой показан на рисунке 3.32. Также рассмотрим 3.33.

```
10  def send_educator_report_to_broker(report: EducatorReport):
11      data = bundle_report(report)
12
13      with pika.BlockingConnection(
14          pika.ConnectionParameters(host=os.getenv('RABBITMQ_HOST'))
15      ) as connection:
16          channel = connection.channel()
17
18          channel.basic_publish(
19              exchange=os.getenv('RABBITMQ_EXCHANGE'),
20              routing_key=os.getenv('RABBITMQ_ROUTING_KEY'),
21              body=json.dumps(data, indent=4, ensure_ascii=False).encode('utf-8')
22          )
```

Рисунок 3.32 – Отправка анкеты в брокер сообщений

```
67      data['year'] = report.year
68
69      employee_record = AllowedEmail.objects.get(
70          email=report.educator.user.email
71      )
72      data['employeeId'] = employee_record.employee_id
73      data['values'] = []
74
75      educator_partitions = EducatorRatingPartition.objects.all(
76          ).values_list('partition')
77
78      report_criterions = Criterion.objects.filter(
79          partition__in=educator_partitions
80      ).order_by(
81          'partition', 'number', 'subnumber'
82      ).select_related('partition', 'indicator')
83
84      indicator_values = EducatorIndicatorValue.objects.filter(
85          report=report
86      ).select_related('indicator')
87
88      for criterion in report_criterions:
89          indicator_value = indicator_values.get(
90              indicator=criterion.indicator
91          ).value
92
93          data['values'].append(
94              {
95                  'criterionKey': get_formatted_criterion_key(
96                      criterion.partition.abbreviation,
97                      criterion.number,
98                      criterion.subnumber
99                  ),
100                 'criterionValue': convert_to_float(indicator_value)
101             }
102         )
103
104     return data
```

Рисунок 3.33 – Сборка данных анкеты для модуля интеграции

Здесь показан фрагмент функции “bundle_report”, которая производит сборку данных по анкете в удобном для модуля интеграции виде.

3.10 Настройка почтовых оповещений

Как было сказано выше, используется библиотека Djoser, которая имеет обширный функционал, в том числе систему почтовых оповещений для операций, связанных с аккаунтом пользователя. Рассмотрим настройки, выполненные для проекта (рисунок 3.34).

```
257 # Djoser
258
259 DOMAIN = os.getenv('CLIENT_DOMAIN')
260
261 SITE_NAME = os.getenv('SITE_NAME')
262
263 SIMPLE_JWT = {
264     'ACCESS_TOKEN_LIFETIME': timedelta(minutes=5),
265     'REFRESH_TOKEN_LIFETIME': timedelta(days=7),
266 }
267
268 DJOSER = {
269     'LOGIN_FIELD': 'email',
270     'SET_PASSWORD_RETYPING': True,
271     'PASSWORD_RESET_CONFIRM_RETYPING': True,
272     'USER_CREATE_PASSWORD_RETYPING': True,
273     'SEND_CONFIRMATION_EMAIL': True,
274     'SEND_ACTIVATION_EMAIL': True,
275     'PASSWORD_CHANGED_EMAIL_CONFIRMATION': True,
276     'ACTIVATION_URL': 'activate/{uid}/{token}',
277     'PASSWORD_RESET_CONFIRM_URL': 'password-reset/confirm/{uid}/{token}',
278     'SERIALIZERS': {
279         'user': 'apps.users.serializers.UserProfileSerializer',
280         'current_user': 'apps.users.serializers.UserProfileSerializer',
281     }
282 }
```

Рисунок 3.34 – Настройки пакета djoser

3.11 Настройка CORS политик

CORS (Cross-Origin Resource Sharing) – это механизм безопасности веб-браузера, который регулирует доступ и обмен ресурсами между разными источниками (доменами) веб-страницы. Он применяется для предотвращения запросов к ресурсам с других доменов, кроме источника, на котором размещена веб-страница.

Когда браузер выполняет запрос к ресурсу, он проверяет политику CORS, отправленную сервером. Если сервер разрешает доступ к ресурсу с определенного домена (или доменов), браузер выполняет запрос и возвращает данные. Если политика CORS запрещает доступ, браузер блокирует запрос и генерирует ошибку.

Политика CORS основана на использовании HTTP заголовков, таких как "Access-Control-Allow-Origin" "Access-Control-Allow-Methods" и прочих. Сервер должен явно установить эти заголовки в ответе, чтобы указать, какие источники, методы и заголовки разрешены для доступа к ресурсам.

Для упрощения работы с этими заголовками, как было сказано выше, используется пакет "django-cors-headers". Рассмотрим настройки данного пакета (рисунок 3.35).

```
284 # CORS
285
286 CORS_ALLOWED_ORIGINS = os.getenv('CORS_ALLOWED_ORIGINS').split(' ')
287
288 CSRF_TRUSTED_ORIGINS = os.getenv('CORS_ALLOWED_ORIGINS').split(' ')
289
290 CORS_ALLOW_METHODS = (
291     *default_methods,
292 )
293
294 CORS_ALLOW_HEADERS = (
295     *default_headers,
296     'JWT',
297 )
298
299 CORS_ALLOW_CREDENTIALS = True
```

Рисунок 3.35 – Настройки CORS политик

После этих настроек, конкретные сервисы (например, сервис клиента), адреса которых указаны в переменной среды "CORS_ALLOWED_ORIGINS", будут иметь возможность обращаться к API.

3.12 Интернационализация проекта

Интернационализация в проекте Django представляет собой механизм, позволяющий адаптировать веб-приложение или сайт для работы на разных языках и с учетом локальных культурных особенностей. Интернационализация позволяет

создавать многоязычные приложения, которые предоставляют контент на разных языках в зависимости от предпочтений пользователей.

Django имеет встроенные средства интернационализации. Все текстовые сообщения, которые в дальнейшем будут видны на пользовательском интерфейсе, обрабатываются в специальную функцию “gettext”. После чего, выполнив команду “python3 manage.py makemessages -l <язык>”, генерируется файл локали (с расширением “.po”) для выбранного языка (рисунок 3.36), который заполняется переводчиками и затем используется в команде “python3 manage.py compilemessages”, которая генерирует конечный файл перевода (с расширением “.mo”).

```
1 # LOCALIZATION FOR UNIVERSITY RATING SYSTEM API.
2 # Copyright (C) 2023 Nikita Reznikov
3 # This file is distributed under the same license as the UNIVERSITY RATING SYSTEM API
4 # Nikita Reznikov <nikita.reznikov.public@mail.ru>, 2023.
5 #
6 #, fuzzy
7 msgid ""
8 msgstr ""
9 "Project-Id-Version: PACKAGE VERSION\n"
10 "Report-Msgid-Bugs-To: \n"
11 "POT-Creation-Date: 2023-04-26 18:10+0300\n"
12 "PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
13 "Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
14 "Language-Team: LANGUAGE <LL@li.org>\n"
15 "Language: \n"
16 "MIME-Version: 1.0\n"
17 "Content-Type: text/plain; charset=UTF-8\n"
18 "Content-Transfer-Encoding: 8bit\n"
19 "Plural-Forms: nplurals=4; plural=(n%10==1 && n%100!=11 ? 0 : n%10>=2 &&
20 "n%10<=4 && (n%100<12 || n%100>14) ? 1 : n%10==0 || (n%10>=5 && n%10<=9) || "
21 "(n%100>=11 && n%100<=14)? 2 : 3);\n"
22
23 #: apps/core/admin.py:6
24 msgid "token"
25 msgstr "токен"
26
27 #: apps/core/admin.py:7
28 msgid "tokens"
29 msgstr "токены"
```

Рисунок 3.36 – Файл локализации

3.13 Средства предзаполнения базы данных

При развертывании проекта, было бы удобно иметь инструменты, для получения удалённых данных и загрузки их в базу данных. Django имеет инструмент, называемый “fixture”.

Fixture – это набор тестовых данных, предназначенных для использования в тестировании или инициализации базы данных. Он представляет собой файл, содержащий предопределенные значения полей моделей Django. Эти наборы могут использоваться для загрузки данных в базу данных при развертывании или для создания набора тестовых данных для автоматического тестирования.

Хранение файла с набором данных предлагается осуществлять в облачном хранилище с ограниченным доступом по ссылке. Для его загрузки используется пакет “gdown”. Установка набора производится вспомогательным сценарием, описанном в пункте “Написание вспомогательных сценариев”.

3.14 Кастомизация панели администратора

По умолчанию панель администрирования Django предоставляет базовый набор функций и внешний вид. Кастомизация позволяет настроить панель в соответствии с требованиями проекта, включая изменение внешнего вида, переименование и переупорядочивание полей и так далее. Для кастомизации используется пакет “django-jazzmin”. На рисунке 3.37 представлены настройки кастомизации, а на рисунках 3.38 и 3.39, внешний вид до и после соответственно.

```
195 JAZZMIN_SETTINGS = {
196     'site_logo': 'core/logo/VSTU-128-for-circle.png',
197     'site_icon': 'core/logo/VSTU-32.png',
198     'copyright': 'Volgograd State Technical University',
199     'welcome_sign': _('API Administration'),
200     'order_with_respect_to': [
201         'users',
202         'educators',
203         'departments',
204         'faculties',
205         'rating',
206         'educator_rating'
207     ],
208     >     'icons': { ...
227 }
```

Рисунок 3.37 – Настройки панели администрирования

Рисунок 3.38 – Базовый вид панели администрирования

Рисунок 3.39 – Обновлённый вид панели администрирования

3.15 Настройка инструментов работы с API

Swagger – это инструмент для описания, документирования и тестирования RESTful веб-сервисов. Он предоставляет интерактивную документацию API, ко-

торая помогает разработчикам быстро понять и использовать API.

ReDoc – это инструмент для генерации интерактивной документации API на основе спецификации OpenAPI. Он предоставляет простую в использовании документацию, которая легко интегрируется в проект.

Для использования Swagger и ReDoc используется пакет “drf-yasg”. Настройки пакета показаны на рисунке 3.40. Интерфейсы Swagger и ReDoc изображены на рисунках 3.41 и 3.42 соответственно.

```
178 SWAGGER_SETTINGS = {
179     'SECURITY_DEFINITIONS': {
180         'Basic': {
181             'type': 'basic'
182         },
183         'Token': {
184             'type': 'apiKey',
185             'name': 'Authorization',
186             'in': 'header'
187         }
188     },
189     'LOGIN_URL': 'rest_framework:login',
190     'LOGOUT_URL': 'rest_framework:logout'
191 }
```

Рисунок 3.40 – Настройки пакета yasg

The screenshot shows the Swagger UI interface for the University Rating System API. At the top, there's a navigation bar with the Swagger logo, the URL 'http://localhost:8000/swagger/?format=openapi', and an 'Explore' button. Below the header, the title 'University Rating System API' is displayed along with the version 'v1'. It includes links for 'Base URL: localhost:8000/' and 'http://localhost:8000/swagger/?format=openapi'. A note says 'API for performing operations on rating reports.' followed by 'Terms of service', 'Contact the developer', and 'MIT License'. On the left side, there's a sidebar with a 'Schemes' dropdown set to 'HTTP' and a 'Filter by tag' input field. The main content area is divided into sections: 'allowed-emails' and 'criterions'. The 'allowed-emails' section contains endpoints for listing, creating, reading, updating, partial updating, and deleting allowed emails. The 'criterions' section contains endpoints for listing and reading criterions. Each endpoint is shown with its method (e.g., GET, POST, PUT, PATCH, DELETE), path, and a small icon indicating its status or security level.

Рисунок 3.41 – Интерфейс Swagger

The screenshot displays the ReDoc API documentation interface. On the left, a sidebar shows a tree structure of API endpoints under the root 'allowed-emails'. The 'allowed-emails_list' endpoint is currently selected, highlighted with a grey background. This endpoint has several methods listed: GET allowed-emails_list, POST allowed-emails_create, GET allowed-emails_read, PUT allowed-emails_update, PATCH allowed-emails_partial_update, and DELETE allowed-emails_delete. Below these methods, sections for 'Responses' and 'Responses' are shown, each containing a '200' status code section with a JSON schema for the response body.

allowed-emails_list

AUTHORIZATIONS: { Basic } OR { Token }

Responses

200

RESPONSE SCHEMA: application/json

Array [

- id integer (ID)
- email string <email> (Email) [1 .. 254] characters

]

allowed-emails_create

AUTHORIZATIONS: { Basic } OR { Token }

REQUEST BODY SCHEMA: application/json

email required string <email> (Email) [1 .. 254] characters

Responses

201

RESPONSE SCHEMA: application/json

id integer (ID)

email string <email> (Email) [1 .. 254] characters

Response samples

200

Content type application/json

```
[{"id": 0, "email": "user@example.com"}]
```

Copy Expand all Collapse all

Request samples

POST /allowed-emails/

Payload

Content type application/json

```
{"email": "user@example.com"}
```

Copy Expand all Collapse all

Response samples

201

Content type application/json

```
{}
```

Copy Expand all Collapse all

Рисунок 3.42 – Интерфейс ReDoc

4 РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ СИСТЕМЫ

4.1 Выбор инструментов разработки

4.1.1 Выбор языка программирования

В качестве языка программирования выбран JavaScript. Выбор обосновывается рядом факторов.

- 1) JavaScript является основным языком программирования для разработки клиентской части веб-приложений. Он поддерживается всеми современными браузерами, что обеспечивает широкую совместимость.
- 2) JavaScript обладает мощными инструментами и библиотеками, такими как React, Angular, Vue.js и прочими, которые значительно упрощают разработку сложных пользовательских интерфейсов. Эти фреймворки предоставляют удобные средства для управления состоянием, роутингом и взаимодействием с сервером.
- 3) JavaScript поддерживает асинхронное программирование, что позволяет эффективно работать с сервером без блокировки пользовательского интерфейса. С помощью технологий, таких как AJAX и Fetch API, можно отправлять асинхронные запросы на сервер и обновлять содержимое страницы без перезагрузки.
- 4) JavaScript является одним из самых популярных языков программирования, что обеспечивает наличие обширного сообщества разработчиков и большого количества готовых решений. Существует огромное количество библиотек, фреймворков и инструментов, которые значительно упрощают и ускоряют процесс разработки.

4.1.2 Выбор фреймворка

Angular, React и Vue – это три наиболее популярных JavaScript инструмента, используемые для разработки пользовательского интерфейса веб-приложений.

- 1) Angular – это полноценный фреймворк, разработанный и поддерживаемый командой разработчиков Google. Он предлагает широкий набор функций, включая управление состоянием приложения, маршрутизацию,

обработку форм и множество других возможностей. Angular использует свой собственный язык разметки шаблонов, называемый Angular Template Language, и позволяет создавать масштабируемые и сложные приложения.

- 2) React – это библиотека JavaScript, разработанная Facebook. Она фокусируется на построении компонентов пользовательского интерфейса, которые могут быть переиспользованы и собраны вместе, чтобы создать приложение. React использует JSX – специальный синтаксис, который позволяет объединять JavaScript и HTML внутри компонентов. React также предлагает Virtual DOM – виртуальное представление DOM, которое позволяет оптимизировать производительность обновления пользовательского интерфейса.
- 3) Vue – это прогрессивный JavaScript фреймворк, который нацелен на простоту использования и интеграцию в существующие проекты. Он предлагает модульную архитектуру и легкую обучаемость, что делает его привлекательным для начинающих разработчиков. Vue использует шаблонизацию и директивы для создания компонентов и обработки событий. Он также имеет удобные инструменты для управления состоянием приложения.

По итогу сравнения выбран React. Выбор React обосновывается его гибкостью, широкой поддержкой и активным сообществом разработчиков. React является очень популярным инструментом в индустрии, и у него огромное количество доступных компонентов, библиотек и ресурсов. Благодаря виртуальному DOM и эффективному обновлению, React обеспечивает высокую производительность при работе со сложными приложениями. Он также имеет хорошую документацию и поддержку со стороны Facebook и активного сообщества, что облегчает изучение и решение проблем.

4.2 Подготовка рабочего окружения

4.2.1 Используемые пакеты

В дополнении к React при реализации приложения использовались также пакеты из списка ниже.

- 1) React Router DOM – это библиотека, специально разработанная для использования с React, которая предоставляет возможности маршрутизации веб-приложений. Она позволяет разработчикам определить маршруты для различных URL-адресов и связать их с соответствующими компонентами React. React Router DOM обеспечивает навигацию между различными представлениями приложения без перезагрузки страницы, что создает более плавное и реактивное пользовательское взаимодействие. Он также предоставляет функциональность, такую как параметры маршрута, вложенные маршруты, защита маршрутов и многое другое.
- 2) Redux - это библиотека управления состоянием, широко используемая в разработке JavaScript-приложений, особенно в связке с фреймворком React. Она помогает управлять и обновлять состояние приложения в предсказуемой и однонаправленной манере.
- 3) Redux Toolkit – это пакет инструментов, разработанный для упрощения использования и управления состоянием в приложениях, использующих Redux. Он предоставляет удобные и простые в использовании абстракции для создания и организации Redux-совместимых хранилищ, управления действиями и редюсерами, а также обработки асинхронных операций. В целом, Redux Toolkit значительно снижает сложность и объем кода, необходимого для работы с Redux.
- 4) Redux Query – это библиотека, которая предоставляет инструменты для управления и нормализации состояния асинхронных запросов в Redux. Она расширяет функциональность Redux, позволяя легко и эффективно управлять состоянием запросов к серверу, кешированием и обновлением данных. Redux Query предлагает удобный способ определения запросов, управления их жизненным циклом, обработки ошибок и синхронизации данных с сервером и локальным состоянием. Она также обеспечивает нормализацию данных, позволяя хранить данные в едином виде и связывать их между разными запросами и сущностями.
- 5) MUI (Material-UI) – это популярная библиотека пользовательского интерфейса для React, которая предоставляет компоненты и стили в соответствии с принципами дизайна Material Design, разработанными Google. MUI предоставляет широкий набор готовых компонентов, таких как кнопки, формы, таблицы, модальные окна и многое другое, которые могут

быть легко настроены и переопределены с помощью тем и стилей. Библиотека имеет модульную архитектуру, что позволяет использовать только нужные компоненты, минимизируя размер конечного бандла.

4.2.2 Написание Docker образа

Для сервиса клиента, аналогично сервису API, реализуется собственный образ, который строится из файла Dockerfile. Так же определено три основных этапа: base, builder и final. Структура каждого этапа схожа с сервисом API, поэтому опустим их описание. Фрагменты кода этапов показаны на рисунках 4.1, 4.2 и 4.3 соответственно.

```
1 FROM node:18-alpine as base
2
3 ARG DEPLOY_ENV
4
5 ENV DEPLOY_ENV=${DEPLOY_ENV}
6
```

Рисунок 4.1 – Первая стадия построения Docker образа

```
7
8 FROM base as builder
9
10 WORKDIR /app/
11
12 COPY . /app/
13
14 RUN npm install
15
```

Рисунок 4.2 – Вторая стадия построения Docker образа

```

16
17 FROM base as final
18
19 COPY --from=builder /app/node_modules/ /app/node_modules/
20 COPY . ./app/
21
22 WORKDIR /app/
23
24 RUN chmod +x ./entrypoint.sh
25
26 RUN addgroup -S app && \
27     adduser -S app -G app -h /app -DH
28
29 RUN mkdir .npm/ node_modules/.cache/ && \
30     chown app:app -R .npm/ && \
31     chown app:app -R node_modules/.cache/
32
33 USER app
34
35 ENTRYPOINT [ "./entrypoint.sh" ]

```

Рисунок 4.3 – Третья стадия построения Docker образа

4.2.3 Написание сценария точки входа

Сценарий точки входа для этого сервиса значительно компактнее в сравнении с сценарием для сервиса API. Здесь лишь выполняется команда запуска приложения. Код сценария показан на рисунке 4.4.

```

1 #!/bin/sh
2
3 echo "Starting client ..."
4
5 if [ "$DEPLOY_ENV" = "prod" ]; then
6     echo "Using production mode"
7     echo "Creating production build ..."
8     npm run build
9     echo "Not implemented"
10    exit 0
11 else
12     echo "Using development mode"
13     npm run start
14 fi

```

Рисунок 4.4 – Сценарий точки входа в контейнер

4.3 Работа с Redux

4.3.1 Определение хранилища

Хранилище в Redux является центральным элементом, где хранится состояние всего приложения. Он предоставляет единое и непротиворечивое представление данных для всего приложения, что облегчает управление состоянием. За счет централизации состояния, компоненты приложения могут получать доступ к состоянию и подписываться на его изменения. Код определения хранилища показан на рисунке 4.5.

```
1 import { configureStore } from "@reduxjs/toolkit";
2 import { apiSlice } from "./api/apiSlice";
3 import authReducer from '../features/auth/authSlice';
4
5
6 const store = configureStore({
7   reducer: {
8     [apiSlice.reducerPath]: apiSlice.reducer,
9     auth: authReducer,
10   },
11   middleware: (getDefaultMiddleware) => getDefaultMiddleware().concat(apiSlice.middleware),
12   devTools: process.env.NODE_ENV === 'development',
13 },
14
15 export default store;
```

Рисунок 4.5 – Определение хранилища

4.3.2 Создание baseQuery

Для того чтобы упростить взаимодействие с API используется Redux Query. Рассмотрим рисунок 4.6 на котором показано определение baseQuery.

```
5 const baseQuery = fetchBaseQuery({
6   baseUrl: process.env.REACT_APP_API_URL,
7   prepareHeaders: (headers, { getState }) => {
8     const token = getState().auth.access
9     if (token) {
10       headers.set('Authorization', `JWT ${token}`);
11     }
12     return headers;
13   }
14});
```

Рисунок 4.6 – Определение baseQuery

Любой запрос к API теперь всегда будет содержать токен аутентификации в своих заголовках. Однако если токен истечёт, то пользователю снова придётся проходить этап авторизации. Учитывая что JWT токен доступа имеет достаточно короткое время жизни, для удобства пользователя необходимо автоматически обновлять его по истечению действия, используя токен обновления (refresh token). Для этого написана функция-обёртка для базового запроса. Её код показан на рисунке 4.7.

```
18  const baseQueryReauth = async (args, api, extraOptions) => {
19    let result = await baseQuery(args, api, extraOptions);
20
21    if (result?.error?.status === 401 && api.getState().auth.refresh) {
22      const refreshResult = await baseQuery(
23        {
24          url: '/jwt/refresh/',
25          method: 'POST',
26          body: { refresh: api.getState().auth.refresh }
27        },
28        api,
29        extraOptions
30      );
31
32      if (refreshResult?.data) {
33        const refresh = api.getState().auth.refresh;
34        const access = refreshResult.data.access;
35        api.dispatch(setCredentials({ refresh, access }));
36        result = await baseQuery(args, api, extraOptions);
37      } else {
38        api.dispatch(logout());
39      }
40    }
41
42    return result;
43 }
```

Рисунок 4.7 – Обёртка для baseQuery

Здесь происходит проверка результата выполнения базового запроса. Если код ответа указывает на то, что токен истёк, то функция выполняет обновление токена и совершает запрос еще раз. Если токен не удалось обновить, то только тогда уже выполняется завершение сессии аутентификации.

4.3.3 Создание слайса аутентификации

Для входа, регистрации и прочих функций, связанных с аутентификацией пользователя определен отдельный слайс. Фрагмент его кода показан на рисунке

4.8. Здесь происходит определение редюсеров и управление локальным хранилищем браузера.

```
4 const authSlice = createSlice({
5   name: 'auth',
6   initialState: { user: null, access: null, refresh: null, isAuthenticated: false },
7   reducers: {
8     setCredentials: (state, action) => {
9       const { access, refresh } = action.payload;
10      state.access = access;
11      state.refresh = refresh;
12      state.isAuthenticated = true;
13
14      localStorage.setItem('access', access);
15      localStorage.setItem('refresh', refresh);
16    },
17    setUser: (state, action) => {
18      state.user = action.payload;
19    },

```

Рисунок 4.8 – Слайс аутентификации

Для дальнейшего обращения к API выполняется вставка эндпоинтов в корневой API слайс (код на рисунке 4.9).

```
4 export const authApiSlice = apiSlice.injectEndpoints({
5   endpoints: build => ({
6     login: build.mutation({
7       query: credentials => ({
8         url: '/jwt/create/',
9         method: 'POST',
10        body: { ...credentials }
11      }),
12      invalidatesTags: ['ReportList', 'ReportController'],
13    }),

```

Рисунок 4.9 – Слайс аутентификации

При входе на сайт хранилище redux будет находиться в начальном состоянии и токен доступа к api не будет пропущен. Для того чтобы восстанавливать сессию пользователя определяется редюсер “restoreSession”. Его код показан на рисунке 4.10.

```

29     restoreSession: (state, _) => {
30       let access = localStorage.getItem('access');
31       let refresh = localStorage.getItem('refresh');
32       state.isAuthenticated = access && refresh ? true : false;
33
34       state.access = access
35       state.refresh = refresh
36     }
37   },
38 }

```

Рисунок 4.10 – Редюсер “restoreSession”

Наличие пары токенов проверяется в хранилище браузера. Если токены найдены, то сессия восстанавливается.

4.3.4 Слайсы для работы с API

Для каждой сущности API, определяется слайс, который образуется вставкой эндпоинтов в корневой API слайс. Рассмотрим пример такого слайса. Фрагмент кода для работы с анкетами преподавателей показан на рисунке 4.11.

```

3  export const reportsApiSlice = apiSlice.injectEndpoints({
4    endpoints: build => ({
5      getMyReports: build.query({
6        query: () => ({
7          url: '/educator-reports/my/',
8          method: 'GET'
9        }),
10       providesTags: ['ReportList'],
11     }),
12     getReport: build.query({
13       query: (id) => ({
14         url: `/educator-reports/${id}/`,
15         method: 'GET'
16       }),
17       providesTags: ['Report'],
18     })
19   })
20 }

```

Рисунок 4.11 – Слайс для работы с анкетами

4.4 Определение маршрутов

Для выделения маршрутов внутри приложения используется компонент “Route” из библиотеки “react-router-dom”. Маршруты приложения показаны на рисунке 4.12.

```

11  export default function App() {
12    return (
13      <Routes>
14        {/* public routes */}
15        <Route path="/login" element={<LoginPage />} />
16        <Route path="/signup" element={<SignupPage />} />
17        <Route path="/activate/:uid/:token" element={<ActivateAccountPage />} />
18
19        {/* protected routes */}
20        <Route element={<RequireAuth />}>
21          <Route path="/reports" element={<ReportsPage />} />
22          <Route path="/reports/:id" element={<ReportDetailsPage />} />
23        </Route>
24        <Route path="*" element={<NotFound />} />
25      </Routes>
26    );
27  }

```

Рисунок 4.12 – Определение маршрутов

Как можно заметить имеется разделение на публичные и защищённые маршруты. Защищенные маршруты включены в компонент “RequireAuth”. Рассмотрим его код, показанный на рисунке 4.13.

```

1  import { useLocation, Navigate, Outlet } from "react-router-dom"
2  import { useSelector } from "react-redux"
3  import { selectIsAuthenticated } from "./authSlice"
4
5  const RequireAuth = () => {
6    const isAuthenticated = useSelector(selectIsAuthenticated)
7    const location = useLocation()
8
9    return (
10      isAuthenticated
11        ? <Outlet />
12        : <Navigate to="/login" state={{ from: location }} replace />
13    )
14  }
15  export default RequireAuth

```

Рисунок 4.13 – Компонент “RequireAuth”

Здесь выполняется проверка состояния аутентификации пользователя. Если он аутентифицирован выполняется отрисовка дочерних элементов, иначе происходит переадресация пользователя на страницу входа.

4.5 Создание основных компонентов

При рассмотрении компонентов будет показываться лишь их основная логика. Фрагменты с JSX разметкой будут опускаться.

4.5.1 Форма регистрации

Данный React компонент представляет собой форму регистрации. Он использует хуки состояния для управления состоянием формы и отображения сообщений об ошибках (рисунок 4.14).

```
8  export default function SignupForm() {
9      const [formData, setFormData] = useState({
10         email: '',
11         password: '',
12         re_password: ''
13     });
14     const [helperMessage, setHelperMessage] = useState("");
15     const [errorStatus, setErrorStatus] = useState(false);
16     const [emailError, setEmailError] = useState({ error: false, message: "" });
17     const [passwordError, setPasswordError] = useState({ error: false, message: "" });
18     const [repasswordError, setRepasswordError] = useState({ error: false, message: "" });
19     const [registrationProcess, setRegistrationProcess] = useState(false);
20     const [signup] = useSignupMutation();
21
22     const { email, password, re_password } = formData;
```

Рисунок 4.14 – Компонент “SignupForm” (фрагмент 1)

В компоненте определены следующие состояния:

- 1) formData – состояние, содержащее значения полей формы (email, password, re_password);
- 2) helperMessage – состояние для отображения сообщения пользователю;
- 3) errorStatus – состояние, указывающее, есть ли ошибка в процессе регистрации;
- 4) emailError, passwordError, repasswordError – состояния, содержащие информацию об ошибках для соответствующих полей формы;
- 5) registrationProcess – состояние, указывающее, выполняется ли процесс регистрации;
- 6) signup – результат вызова пользовательского хука useSignupMutation(), функция для обращения к API.

Компонент имеет две основные функции (рисунок 4.15).

```

24 |     function onChange(e) {
25 |         setFormData({ ...formData, [e.target.name]: e.target.value });
26 |
27 |
28 |     Codeium: Refactor | Explain | Generate JSDoc
29 |     async function onSubmit(e) {
30 |         e.preventDefault();
31 |         setErrorStatus(false);
32 |         setHelperMessage('Регистрация...');
33 |         setRegistarionProcess(true);
34 |         setEmailError({ error: false, message: '' });
35 |         setPasswordError({ error: false, message: '' });
36 |         setRepasswordError({ error: false, message: '' });
37 |
38 |
39 |         let response = await signup(formData);
40 >         if (response?.error?.data) { ...
41 |             }
42 |
43 |             else {
44 |                 setHelperMessage('Письмо с подтверждением отправлено на указанную почту');
45 |                 setFormData({
46 |                     email: '',
47 |                     password: '',
48 |                     re_password: ''
49 |                 });
50 |             }
51 |         } catch (error) {
52 |             setErrorStatus(true);
53 |             setHelperMessage('Ошибка регистрации');
54 |         }
55 |
56 |         setRegistarionProcess(false);
57 |
58 |     }

```

Рисунок 4.15 – Компонент “SignupForm” (фрагмент 2)

Рассмотрим подробнее, какие действия они выполняют:

- 1) onChange – функция, вызываемая при изменении значений полей формы. Она обновляет состояние formData, используя оператор расширения для сохранения предыдущих значений полей и обновления только измененного поля.
- 2) onSubmit – функция, вызываемая при отправке формы. Она выполняет следующие действия:
 - предотвращает обновление страницы при отправке формы;
 - сбрасывает состояния ошибок и вспомогательных сообщений;
 - устанавливает состояние процесса регистрации;
 - выполняет запрос на регистрацию, вызывая функцию signup с использованием данных из formData;
 - проверяет ответ сервера на наличие ошибок и обновляет соответствующие состояния ошибок и вспомогательных сообщений;

- в случае успешной регистрации обновляет вспомогательное сообщение и сбрасывает значения полей формы;
- в случае ошибки регистрации обновляет состояние ошибки и вспомогательное сообщение;
- снимает состояние процесса регистрации.

4.5.2 Форма авторизации

Компонент “LoginForm” аналогичен компоненту регистрации однако дополнительно использует хук “useEffect” для того, чтобы проверять не авторизован ли уже пользователь (код на рисунке 4.16).

```

28  useEffect(() => {
  Codeium: Refactor | Explain | Generate JSDoc
29    async function fetchUser() {
30      dispatch	restoreSession();
31      if (isAuthenticated) {
32        try {
33          const user = await getUser().unwrap();
34          dispatch(setUser(user));
35          navigate('/reports');
36        } catch (error) {
37          setHelperMessage("Сессия истекла");
38        }
39      }
40    }
41    fetchUser();
42  }, [dispatch, getUser, navigate, isAuthenticated]);

```

Рисунок 4.16 – Использование “useEffect” для восстановления сессии пользователя

Сперва происходит попытка восстановить сессию. После чего проверяется статус аутентификации, если он истинный, то происходит обращение к API при помощи функции “getUser” для загрузки пользователя и последующей его установки в состояние хранилища.

4.5.3 Список анкет

Данный компонент получает список анкет от родительского компонента, который в свою очередь использует соответствующий redux запрос. Данный передаётся в функцию map, которая преобразует каждый элемент списка в отдельный

компонент (код на рисунке 4.17).

```
Codeium: Refactor | Explain | Generate JSDoc
7  export default function ReportList({reports}) {
8    return (
9      <Box className={ styles.reportList }>
10     {
11       reports.length
12       ? reports.map(report => <ReportCard key={ report.id } report={ report } />
13       : <Typography align='center' variant="h5" color="text.secondary">Список пуст</Typography>
14     }
15   </Box>
16 )
17 }
```

Рисунок 4.17 – Компонент “ReportList”

Компонент “ReportCard” уже занимается загрузкой зависимых данных, таких, как информация о преподавателе, создавшего анкету. А также определяет функцию для перехода к анкете (код на рисунке 4.18).

```
8  export default function ReportCard({ report }) {
9    const { data: educator, isLoading: skip } = useGetEducatorQuery(report.educator);
10   const { data: user, isLoading } = use GetUserQuery(educator?.user, { skip });
11   const navigate = useNavigate();
12
13   function handleOnOpenReport() {
14     navigate(`/reports/${report.id}`)
15   }

```

Рисунок 4.18 – Компонент “ReportCard”

4.5.4 Просмотр и редактирование анкеты

Компонент “ReportForm” занимается подготовкой данных для анкеты. Он выполняет загрузку показателей из API и создаёт Map объект, который используется для создания дочерних компонентов. Для значений показателей в анкете хранится лишь идентификатор показателя. Для того чтобы отобразить название показателя как раз и используется Map объект в котором идентификаторы сопоставляются с названиями. Фрагмент кода компонента ‘ReportForm’ показан на рисунке 4.19.

```

7  export default function ReportForm({ reportValues }) {
8    const { data, isLoading } = useGetIndicatorsQuery();
9    const indicatorsMap = isLoading ? new Map() : buildIndicatorsMap();
10
11   Codeium: Refactor | Explain | Generate JSDoc
12   function buildIndicatorsMap() {
13     let indicatorsMap = new Map();
14     for (let indicator of data) {
15       indicatorsMap.set(indicator.id, indicator)
16     }
17     return indicatorsMap;
18   }
19
20   return (
21     <Box>
22       { isLoading || !reportValues
23         ? "Загрузка..."
24         : reportValues.map(
25           item => <ReportValue
26             key={ item.id }
27             id={ item.id }
28             valueType={ item.value.type }
29             value={ item.value.value }
30             indicator={ indicatorsMap.get(item.indicator) }
31           />
32         )
33     )
34   }

```

Рисунок 4.19 – Компонент “ReportForm”

Компонент “ReportValue” реализует логику синхронизации изменения значения показателя с сервером. Фрагмент кода показан на рисунке 4.20.

```

7  export default function ReportValue({ indicator, valueType, value, id }) {
8    const [val, setValue] = useState(value);
9    const [patchReportValue] = usePatchReportValueMutation();
10
11   Codeium: Refactor | Explain | Generate JSDoc
12   async function handleOnChange(newValue) {
13     setValue(newValue);
14     try {
15       if (valueType === 'float') {
16         newValue = parseFloat(newValue)
17       } else if (valueType === 'int') {
18         newValue = parseInt(newValue, 10);
19       }
20
21       await patchReportValue(
22         { id, newValue: {value: newValue, type: valueType} }
23       ).unwrap(); // TODO: debounce
24     } catch (error) {
25       setValue(value);
26     }

```

Рисунок 4.20 – Компонент “ReportValue”

4.6 Обзор полученного интерфейса

На рисунках 4.21, 4.22, 4.23 и 4.24 показаны полученные страницы авторизации, просмотра анкет, создания и просмотра анкеты соответственно.

The screenshot shows a registration form titled 'Авторизация'. It contains two input fields: 'Почта' (Email) with the value 'nikita.reznikov.dev@rambler.ru' and 'Пароль' (Password) with the value '.....'. Below the fields are links 'Нет аккаунта?' (No account?) and a blue 'Войти' (Login) button. The background is white with a blue header bar.

Рисунок 4.21 – Страница авторизации

The screenshot shows a list of surveys ('Список анкет') for the user 'Тестовый Тест Тестович'. There are three entries, each with a status of 'Не подтверждена' (Not confirmed). The first entry is dated 'Год: 2022'. The second is 'Год: 2021'. The third is 'Год: 2020'. Each entry has a 'View' button ('открыть') and a 'Delete' button ('удалить'). The top right of the page has buttons for 'Create survey' ('Создать анкету') and 'Logout' ('Выход'). The background is white with a blue header bar.

Рисунок 4.22 – Страница просмотра анкет

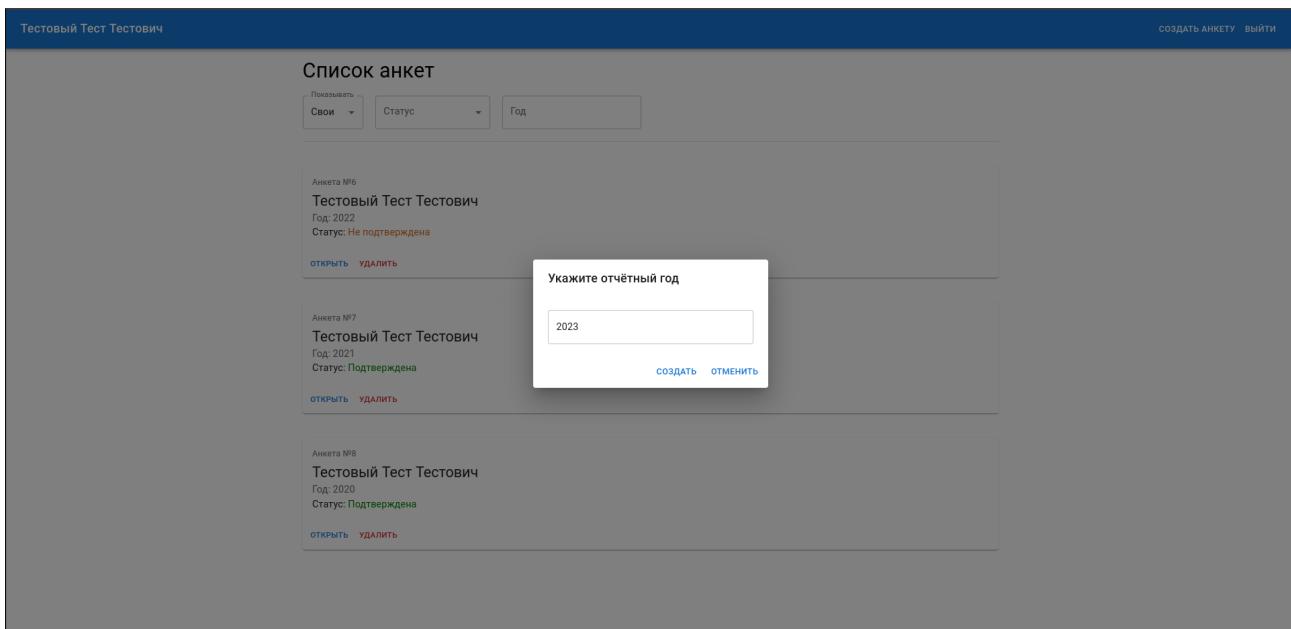


Рисунок 4.23 – Модальное окно создания анкеты

A screenshot of a survey detail page. The header includes "Тестовый Тест Тестович" on the left and "УТВЕРДИТЬ АНКЕТУ ВЕРНУТЬСЯ К АНКЕТАМ ВЫЙТИ" on the right. The main content area is titled "Тестовый Тест Тестович, 2022 год". It contains a list of achievements with checkboxes and numerical counts. The items and their counts are: Ученая степень кандидата наук (0), Ученая степень доктора наук (0), Ученое звание доцент (0), Ученое звание профессор (0), Академик РАН (0), Член-корр. РАН (0), Членство в других государственных академиях (0), Членство в национальных академиях других стран (0), Членство в иных (общественных) академиях (0), Членство в творческих союзах РФ (Союз архитекторов, Союз дизайнеров, Союз художников) (0), Государственные награды и премии (0), Почетные звания и знаки, награды РАН (0), and Министерские награды и премии (0).

Рисунок 4.24 – Страница просмотра и редактирования анкеты

ЗАКЛЮЧЕНИЕ

В ходе работы была рассмотрена существующая методика проведения рейтинговой оценки деятельности преподавателей. Был проведен анализ методики и выявлены её недостатки. Для установленных проблем были приняты соответствующие решения по их минимизации.

В результате поиска аналогов было выяснено, что на текущий момент количество систем, связанных с рейтинговой оценкой преподавателей не велико и они имеют узкоспециализированный характер.

Анализ аналогов в совокупности с текущими недостатками методики подчеркнул важность и актуальность разработки данной системы, которая сможет принести пользу университету и способствовать повышению качества обучения и росту навыков у преподавательского состава.

В связи с этим было принято решение проектирования и реализации новой системы, которая бы помимо соблюдения требований уже имеющихся положений, также учитывала установленные по результатам анализа недостатки текущей методики.

В результате была реализована система состоящая из двух основных модулей – серверная часть, представляющая собой многофункциональное API и клиентская часть, представляющая собой веб-приложение для преподавательского состава.

Помимо этого была проведена работа по интеграции с системой учёта сотрудников университета 1С. Это позволит синхронизировать данные о преподавателях и их показателях, исключая необходимость вручную вносить информацию в обеих системах.

В итоге разработанная система представляет собой комплексное решение для рейтинговой оценки преподавателей, которое устраняет недостатки существующей методики, обеспечивая задействованных сторон удобным инструментом и упрощая их процесс взаимодействия.

Направлением дальнейшего развития проекта может являться разработка внешних модулей и приложений, интегрирующихся с системой. Примером может являться модуль аналитики рейтинга, мобильное приложение для преподавателя, модуль интеграции с базами научных публикаций и прочее.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Василенко, О. Ю. Преподаватель вуза: мотивация и стимулирование трудовой деятельности / О.Ю. Василенко, Е.В. Вельц // Вестник Омского университета. – 1999. – №4. – С. 134-136.
2. Гуцу, Е. Г. Исследование мотивации трудовой деятельности преподавателя вуза / Е. Г. Гуцу, М. Д. Няголова, Т. А. Рунова // Вестник Мининского университета – 2018. – Т. 6, №3 – URL: <https://www.minin-vestnik.ru/jour/article/view/874/686> (дата обращения 02.04.2023).
3. Моэт, Э. Использование Docker / Э. Моэт; пер. с англ. А. В. Снастина; под ред. А. А. Маркелова. – Москва : ДМК Пресс, 2017. – 354 с. – ISBN 978-5-97060-426-7.
4. Docker Docs // Docker : офиц. сайт. – URL: <https://docs.docker.com/> (дата обращения 03.04.23).
5. Django Cors Headers Configuration // Django Cors Headers : офиц. сайт. – URL: <https://pypi.org/project/django-cors-headers/> (дата обращения 2.05.23).
6. Django documentation // Django : офиц. сайт. – URL: <https://docs.djangoproject.com/en/4.2/> (дата обращения 5.04.23).
7. Django Command Extensions // Django Extensions: офиц. сайт. – URL: https://django-extensions.readthedocs.io/en/latest/command_extensions.html (дата обращения 17.05.23).
8. Django REST framework // Django REST framework : офиц. сайт. – URL: <https://www.django-rest-framework.org/> (дата обращения 9.04.23).
9. Djoser Settings // Djoser: офиц. сайт. – URL: <https://djoser.readthedocs.io/en/latest/settings.html> (дата обращения 20.04.23).
10. Gdown Usage Examples // Gdown: офиц. сайт. – URL: <https://pypi.org/project/gdown/> (дата обращения 25.04.23).
11. Jazzmin Configuration // Jazzmin: офиц. сайт. – URL: <https://django-jazzmin.readthedocs.io/configuration/> (дата обращения 17.04.23).

12. NPM CLI Commands // NPM: офиц. сайт. – URL: <https://docs.npmjs.com/cli/v9/commands> (дата обращения 10.05.23).
13. Open API Specification // Swagger: офиц. сайт. – URL: <https://swagger.io/specification/v2/> (дата обращения 6.04.23).
14. Index of Python Enhancement Proposals // Python : офиц. сайт. – URL: <https://peps.python.org/pep-0000/> (дата обращения: 13.04.23).
15. Pika Usage Examples // Pika: офиц. сайт. – URL: <https://pika.readthedocs.io/en/stable/examples.html> (дата обращения 12.05.23).
16. Poetry documentation // Poetry : офиц. сайт. – URL: <https://python-poetry.org/docs/> (дата обращения 04.04.23).
17. RabbitMQ Documentation // RabbitMQ : офиц. сайт. – URL: <https://www.rabbitmq.com/documentation.html> (дата обращения 7.05.23).
18. React reference // React : офиц. сайт. – URL: <https://react.dev/reference/react> (дата обращения 10.05.23).
19. Redoc Documentation // Redocly : офиц. сайт. – URL: <https://redocly.com/docs/> (дата обращения 16.04.23).
20. Redux Toolkit Api Reference // Redux Toolkit : офиц. сайт. – URL: <https://redux-toolkit.js.org/> (дата обращения 12.05.23).