

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Программные средства обеспечения
механизма единой аутентификации**

Студент

(подпись, дата) Лакеева Софья Романовна
(ФИО полностью)

Руководитель

(подпись, дата) Иванчуков А.Г.
(фамилия, инициалы)

Нормоконтролёр

(подпись, дата) Такташкин Г.В.
(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 09.06.2025 № 12)

Заведующий кафедрой

(подпись) Козлов А.Ю.
(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

(подпись) Попова Н.А.
(фамилия, инициалы)

Пенза, 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«20» декабря 2024 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Лакеева Софья Романовна гр. 21ВП2 факультета ФВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Программные средства обеспечения механизма единой аутентификации

Тема утверждена приказом ПГУ № _____ от « _____ » _____ 20 _____

3. Руководитель работы Иванчуков А.Г.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

1) Разработать программные компоненты подсистемы, обеспечивающие единую аутентификацию внешних систем, запрашивающих доступ к защищенным ресурсам

2) Среда выполнения приложения: ОС Windows 10 и выше

Языковые средства разработки: C#, SQLite, UML, React.js

Инструментальные средства разработки: Visual Studio, DB Browser for SQLite, Visual Paradigm 16.3, Visual Studio Code

Технологии разработки: RUP, ООП, Entity Framework Core, SPA

3) Основные функции программного обеспечения:

а) регистрация внешних систем;

б) аутентификация и авторизация внешних систем, поддерживающих разные протоколы проверки подлинности;

в) предоставление авторизованным системам доступа к защищенным

<i>ресурсам;</i>
<i>г) администрирование внешних систем и настройка прав доступа;</i>
<i>д) протоколирование действий внешних систем.</i>

5. Перечень подлежащих разработке вопросов

<i>1) Анализ предметной области и постановка задачи</i>
<i>2) Проектирование</i>
<i>3) Реализация</i>
<i>4) Контроль качества программного обеспечения</i>
<i>5) Оформление пояснительной записки</i>

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025 – 14.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	20%	13.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 121 лист, 26 рисунков, 22 таблицы, 26 использованных источников, 5 приложений.

АУТЕНТИФИКАЦИЯ, АВТОРИЗАЦИЯ, ВЗАИМОДЕЙСТВИЕ СИСТЕМ, ИНТЕГРАЦИЯ МЕХАНИЗМОВ АУТЕНТИФИКАЦИИ, БЕЗОПАСНЫЙ ОБМЕН ДАННЫМИ, OAUTH 2.0, JWT, C#, ASP.NET CORE, ENTITY FRAMEWORK CORE, REACT, SQLITE, UML.

Объектом разработки являются функциональные компоненты информационной системы электронного документооборота.

Предметом разработки является подсистема обеспечения механизма единой аутентификации внешних систем.

Цель работы – разработка подсистемы обеспечения единой аутентификации внешних систем, запрашивающих доступ к функциональным компонентам информационной системы электронного документооборота.

Средства разработки: язык программирования C#, фреймворк Entity Framework Core, библиотека React.js, среды разработки Visual Studio, Visual Studio Code, СУБД SQLite.

Среда исполнения – Windows 10, web-браузер.

Результаты работы: приведен обзор и анализ существующих протоколов аутентификации, определены функциональные и нефункциональные требования к подсистеме обеспечения единой аутентификации, спроектирована архитектура на основе сервисов, разработано и протестировано программное обеспечение механизма единой аутентификации внешних систем.

					ПГУ 09.03.04 – 08BP212.13 ПЗ			
Изм.	Лист	№ докум.	Подп.	Дата	Программные средства обеспечения механизма единой аутентификации. Пояснительная записка.	Лит.	Лист	Листов
Разраб.		Лакеева С.Р.					4	121
Пров.		Иванчуков А.Г.						
Реценз.								
Н. контр.		Такташкин Г.В.						
Утверд.		Макарычев П.П.				Группа 21ВП2		

Оглавление

Введение.....	7
1 Анализ предметной области и постановка задачи на разработку	8
1.1 Анализ предметной области	8
1.1.1 Понятие аутентификации в информационных системах	8
1.1.2 Проблемы аутентификации при взаимодействии систем	10
1.2 Анализ существующих протоколов аутентификации.....	10
1.2.1 Аутентификация конечных пользователей.....	10
1.2.1.1 Протокол Kerberos	11
1.2.1.2 Протокол SSL/TLS.....	14
1.2.1.3 Протокол SAML.....	15
1.2.1.4 Протокол OAuth 2.0.....	17
1.2.1.5 Протокол OpenID Connect	19
1.2.2 Аутентификация сервисов и компонентов систем.....	21
1.2.2.1 Протокол mTLS.....	22
1.2.2.2 Протокол OAuth 2.0 Client Credentials Grant.....	22
1.2.2.3 Стандарт JWT.....	23
1.2.2.4 Механизм API-ключей	24
1.3 Постановка задачи на разработку программных средств	26
1.4 Функционально-стоимостной анализ	28
2 Анализ требований к разрабатываемым средствам	33
2.1 Анализ функциональных требований.....	33
2.2 Анализ нефункциональных требований	36
3 Проектирование программных средств.....	38
3.1 Общая архитектура программных средств.....	38
3.2 Проектирование интерфейса администрирования	41
3.3 Проектирование подсистемы единой аутентификации	42
4 Реализация программных средств.....	56
4.1 Реализация серверной части программных средств.....	56
4.2 Реализация клиентской части программных средств.....	60

	6
4.3 Развертывание компонентов	64
5 Контроль качества программных средств	66
5.1 Функциональное тестирование серверной части приложения	66
5.1.1 Классы эквивалентности	67
5.1.2 Диаграмма состояний и переходов	70
5.2 Функциональное тестирование интерфейса администрирования	72
5.3 Оценка качества программного кода	74
Заключение	79
Список использованных источников	81
Приложение А (справочное) Глоссарий	84
Приложение Б (обязательное) Исходный код серверной части	87
Приложение В (обязательное) Исходный код клиентской части	101
Приложение Г (рекомендуемое) Результаты тестирования программных средств	109
Приложение Д (рекомендуемое) Акт о внедрении	120

Введение

С развитием цифровых технологий информационные системы стали неотъемлемой частью практически всех аспектов современной жизни. Эти системы активно взаимодействуют между собой, обмениваются данными и образуют распределенные архитектуры. В такой среде обеспечение целостности и конфиденциальности передаваемой информации является одной из приоритетных задач. Одним из ключевых элементов защиты при взаимодействии систем выступает аутентификация – процесс проверки подлинности стороны, инициирующей соединение.

Актуальность построения доверенной среды для интеграции разнородных компонентов и информационных систем обусловлена рядом факторов. Во-первых, формирование цифровых экосистем (ВКонтакте, СберБанк Онлайн, Яндекс) требует безопасного взаимодействия множества внутренних и сторонних сервисов в рамках единой платформы. Во-вторых, развертывание IoT-сетей (умные города, промышленный интернет вещей) подразумевает взаимодействие тысяч устройств от разных производителей, что приводит к необходимости унифицированной идентификации устройств. В-третьих, современные программные продукты (заказные решения, SaaS-платформы) часто требуют взаимодействия со сторонними программными компонентами через API. Это создает потребность в унификации протоколов аутентификации, гибкой системе управления доступом и безопасном обмене данными.

Целью выпускной квалификационной работы является разработка подсистемы обеспечения единой аутентификации внешних систем, поддерживающих разные протоколы аутентификации, для доступа к функциональным компонентам информационной системы электронного документооборота. Работа включает в себя анализ предметной области, обзор существующих аналогов, анализ функциональных и нефункциональных требований к программным средствам, проектирование, реализацию и контроль качества программного обеспечения.

1 Анализ предметной области и постановка задачи на разработку

1.1 Анализ предметной области

1.1.1 Понятие аутентификации в информационных системах

Термин «информационная безопасность» в последние годы приобрёл особую актуальность в связи с активным развитием информационных технологий. Он всё чаще встречается в различных сферах жизни – от государственной политики до повседневной деятельности организаций и частных пользователей. Информационная безопасность – это состояние защищённости информации, при котором обеспечены её [1]:

- конфиденциальность (доступ только у тех, кто имеет на это право);
- целостность (информация не может быть изменена без разрешения);
- доступность (информация доступна тогда, когда она нужна уполномоченным пользователям).

Для поддержания информационной безопасности необходимо защитить данные от несанкционированного доступа, утечки, потери, подделки и других угроз. Это важная задача любой информационной системы. Для того, чтобы обеспечить безопасный доступ к данным системы, нужно успешно пройти три последовательных этапа: идентификация, аутентификация и авторизация. Рассмотрим подробнее эти понятия.

Идентификация – это процесс определения субъекта в системе (пользователя, устройства или сервиса). Она отвечает на вопрос: «Кто вы?». В качестве уникального идентификатора могут выступать логин, номер телефона, почта, идентификатор устройства и другие.

Аутентификация – это процесс подтверждения личности субъекта на основе предъявленных им данных. Она отвечает на вопрос: «Докажите, что вы тот, за кого себя выдаете». Например, ввод пароля, сканирование отпечатка пальца, подтверждение через одноразовый код.

Авторизация – это процесс проверки прав субъекта на доступ к определенным ресурсам после успешной аутентификации. Она отвечает на

вопрос: «Какие у вас права в системе?». Например, одни пользователи могут только читать данные, а другие – изменять их.

Взаимодействие между компонентами распределённых систем, интеграция внешних сервисов и автоматизированные процессы требуют аутентификации различных классов субъектов. Условно их можно разделить на три основные категории:

- пользователи (физические лица) – конечные пользователи, которые входят в систему с помощью логина и пароля, биометрических данных, аппаратных токенов или многофакторной аутентификации;
- приложения и программные агенты – компоненты распределенной системы (микросервисы, API, фоновые процессы) требуют подтверждения подлинности при взаимодействии с другими компонентами системы;
- внешние системы и интеграционные компоненты – взаимодействие с внешними API, облачными платформами или партнёрскими системами требует безопасного обмена данными.

Несмотря на общую цель – подтвердить подлинность субъекта, – способы реализации аутентификации для пользователей и систем существенно различаются. При аутентификации конечных пользователей система проверяет, действительно ли обращающийся к ней человек является тем, за кого себя выдает. Это может происходить при входе в аккаунт через логин и пароль. При такой аутентификации важную роль играет удобство для пользователей, защита личных данных и устойчивость к взлому со стороны злоумышленников. Аутентификация системы или сервиса осуществляется между программными компонентами. В отличие от конкретного пользователя, который осознанно выполняет шаги для получения доступа к системе, аутентификация программных компонентов происходит автоматически, без участия человека. В этом случае критично важны такие аспекты, как безопасность канала связи, доверие между взаимодействующими сторонами, защита от подмены и повторного воспроизведения запросов.

1.1.2 Проблемы аутентификации при взаимодействии систем

Современные информационные системы редко работают изолированно – они обмениваются данными с внешними сервисами, API и другими компонентами распределённой архитектуры. Однако обеспечение безопасной аутентификации в таких сценариях сопряжено с рядом проблем, связанных с различиями в протоколах, требованиях безопасности и механизмах доверия.

Одной из ключевых проблем является разнородность механизмов и протоколов аутентификации у взаимодействующих систем. Например, одна система использует протокол OAuth 2.0 для делегированного доступа, другая использует взаимную аутентификацию по TLS (подробное описание протоколов аутентификации приведено в разделе 1.2). Это приводит к необходимости реализации дополнительного программного обеспечения (ПО), которое унифицирует протоколы и обеспечит их совместимость. Также при взаимодействии систем необходимо решить, как передавать аутентификационные данные (токены, сертификаты, ключи), избегая перехвата, и где хранить секреты (пароли сервисных аккаунтов, API-ключи).

1.2 Анализ существующих протоколов аутентификации

Приведем обзор и анализ протоколов аутентификации для каждого класса субъектов: конечные пользователи, программные компоненты системы, внешние системы.

1.2.1 Аутентификация конечных пользователей

Аутентификация пользователя – это процесс проверки подлинности личности пользователя, пытающегося получить доступ к системе. В проверке подлинности пользователей могут участвовать факторы знания, обладания и свойства. Факторы знания представляют собой данные, которые пользователь системы точно знает, например, пароль или ПИН-код. Фактор обладания – материальный объект, который пользователь может предъявить, чтобы подтвердить свою принадлежность к учетной записи, например, аппаратный

токен, карта доступа, цифровой сертификат. Фактором свойства являются физиологические характеристики пользователя, например, отпечаток пальца, вектор характеристик лица. Следовательно, механизмы аутентификации можно классифицировать по перечисленным выше факторам. Для повышения безопасности применяются многофакторные методы аутентификации, при которых объединяются два или более факторов из разных групп.

1.2.1.1 Протокол Kerberos

Одним из широко распространённых протоколов аутентификации пользователей в распределённых информационных системах является Kerberos – служба распределения ключей и проверки подлинности, разработанная в Массачусетском технологическом институте (MIT). Данный протокол использует концепцию симметричного шифрования. Проблема, которую решает Kerberos, заключается в том, что в открытой распределенной среде неавторизованный пользователь со своей рабочей станции может получить доступ к данным сервера, поскольку существуют следующие угрозы [2]:

- пользователь может получить физический доступ к рабочей станции и выдать себя за другого пользователя, работающего на той же машине;
- пользователь может изменить сетевой адрес своей рабочей станции, чтобы отправляемые с неё запросы выглядели как поступившие от доверенной машины;
- пользователь может подслушивать сетевой трафик и с помощью атаки повторного воспроизведения получить доступ к серверу.

Вместо того чтобы разрабатывать отдельные механизмы аутентификации на каждом сервере, Kerberos предлагает централизованную службу аутентификации, задача которой – удостоверять пользователей для серверов и серверы для пользователей. Современным стандартом считается Kerberos v5, описанный в RFC 4120 [3].

Kerberos функционирует по модели «трёхстороннего доверия», где все участники полагаются на центральную службу аутентификации. Перечислим основные компоненты протокола:

- клиент – инициатор запроса аутентификации;
- конечный сервер (S), предоставляющий нужный клиенту сервис;
- сервер аутентификации (AS), проверяющий личность клиента;
- сервер выдачи билетов (TGS) для доступа к конкретным сервисам;
- центр распределения ключей (KDC) – логически объединяет AS и TGS, как единый доверенный центр, хранит закрытые ключи пользователей и сервисов, которые доверяют KDC.

Работа протокола Kerberos v5 состоит из трех основных этапов: аутентификация клиента перед сервером аутентификации; запрос и получение билета у службы выдачи билетов; доступ к прикладному сервису с использованием полученного билета [2].

На первом шаге клиент отправляет запрос AS и передает свое имя и имя TGS, к которому хочет получить доступ. AS, получив запрос, ищет информацию о клиенте в базе данных. Если клиент зарегистрирован, сервер формирует временный сеансовый ключ для дальнейшего обмена между клиентом и TGS. Сеансовый ключ отправляется клиенту в зашифрованном виде – с использованием ключа, полученного из пароля клиента. Таким образом, если клиент знает свой пароль, он сможет расшифровать сообщение и получить сеансовый ключ. Вместе с этим сервер отправляет клиенту так называемый «билет TGS» – криптографически защищённый блок данных, предназначенный для сервера выдачи билетов. Этот билет содержит тот же сеансовый ключ и сведения о пользователе, но зашифрован с использованием ключа, известного только TGS.

На втором этапе клиент отправляет запрос TGS, в котором передается ранее полученный билет TGS и специальный временный блок данных, называемый аутентификатором, который создаётся на стороне клиента и шифруется тем сеансовым ключом, что был получен от AS. TGS сначала

расшифровывает билет своим закрытым ключом и извлекает из него сеансовый ключ, которым затем расшифровывает аутентификатор клиента. TGS проверяет: действительно ли билет и аутентификатор относятся к одному клиенту, не истёк ли срок действия билета, не является ли запрос повторным (по временной метке). Если все проверки успешно пройдены, TGS создаёт новый сеансовый ключ, предназначенный для общения клиента с нужным сервисом. Клиенту этот ключ передаётся в зашифрованном виде – с использованием старого сеансового ключа. Также TGS формирует новый билет для целевого сервиса, в который также включается тот же сеансовый ключ и сведения о пользователе. Этот билет зашифровывается ключом, известным только нужному сервису.

На последнем этапе клиент отправляет целевому сервису билет, полученный от TGS, и аутентификатор, зашифрованный с использованием нового сеансового ключа. Конечный сервис, получив запрос, сначала расшифровывает билет своим закрытым ключом и извлекает из него сеансовый ключ. С помощью этого ключа он затем расшифровывает аутентификатор и проверяет данные пользователя и временную метку. При успешном результате проверки сервис убеждается, что перед ним доверенный клиент. Для подтверждения своей подлинности сервис может отправить ответное сообщение, зашифрованное тем же сеансовым ключом.

Приведем обзор протоколов аутентификации пользователей, использующих ассиметричное шифрование, при котором у каждой из взаимодействующих сторон имеется два вида ключей: закрытый и открытый. Для подтверждения подлинности открытого ключа применяются цифровые сертификаты по стандарту X.509, которые выдаются удостоверяющим центром (CA, Certificate Authority) и подтверждают, что данный открытый ключ действительно принадлежит конкретному пользователю. Инфраструктура безопасности для распространения открытых ключей, управления электронными сертификатами и ключами пользователей получила название инфраструктуры открытых ключей – Public Key Infrastructure (PKI) [4].

Аутентификацию пользователей на основе сертификатов и асимметричного шифрования рассмотрим на примере протоколов Secure Socket Layer (SSL), Transport Layer Security (TLS) и Secure Shell (SSH).

1.2.1.2 Протокол SSL/TLS

Протоколы SSL и TLS [5] обеспечивают аутентификацию сторон при установлении соединения между двумя приложениями. Их основная задача – гарантировать, что обе стороны действительно являются теми, за кого себя выдают, предотвращая подделку и несанкционированный доступ. Эти протоколы работают на транспортном уровне поверх TCP и создают защищённый канал для дальнейшего взаимодействия. В то время как SSL устарел и содержит множество уязвимостей, современные версии TLS значительно улучшили механизмы проверки подлинности, что делает их основным средством обеспечения доверия в сети. Рассмотрим подробнее, как именно происходит аутентификация в TLS.

Когда клиент (например, браузер пользователя) подключается к веб-сайту по HTTPS, между клиентом и сервером запускается TLS Handshake – процедура установления защищённого соединения. Она позволяет аутентифицировать сервер, сгенерировать общий секретный ключ, определиться с параметрами шифрования и начать защищенный обмен данными. Опишем этапы данной процедуры.

Клиент инициирует соединение (сообщение ClientHello). Клиент отправляет список поддерживаемых версий протокола TLS, список поддерживаемых алгоритмов шифрования и хэширования, сгенерированное случайное число и дополнительную информацию. Сервер отвечает на запрос клиента (сообщение ServerHello). Сервер отправляет выбранную версию TLS и алгоритмы шифрования из предложенного списка клиента; сгенерированное случайное число; свой сертификат X.509, подписанный СА; если необходимо, параметры общего секретного ключа, используемого для генерации сессионных ключей. Клиент проверяет подлинность сервера: проверка полей сертификата,

цифровой подписи с помощью открытого ключа CA, цепочки сертификатов. Если проверки пройдены успешно переходит к следующему этапу.

На следующем этапе создается общий секрет (pre-master secret). В TLS 1.2 чаще всего используются алгоритмы RSA или ECDHE. При RSA клиент генерирует случайное значение – общий секрет – и зашифровывает его открытым ключом сервера; сервер на своей стороне своим закрытым ключом расшифровывает Pre-Master Secret. Здесь применяется асимметричное шифрование. Клиент и сервер по выбранному на первых этапах алгоритму рассчитывают мастер-ключ (master secret), используя общий pre-master secret и оба случайных числа (client random и server random). На основе мастер-ключа затем генерируются на каждой стороне симметричные ключи шифрования сервера и клиента, MAC-ключи клиента и сервера и при необходимости вектор инициализации.

Каждая сторона отправляет сообщение, подтверждающее, что процедура Handshake прошла успешно. Данное сообщение включает хеш от всех предыдущих сообщений, зашифрованный с помощью сгенерированного ключа шифрования.

После успешного выполнения всех этапов построен защищенный канал передачи, и стороны могут безопасно обмениваться сообщениями. Отправитель сначала формирует содержимое, затем рассчитывает MAC от сообщения и специального счётчика, добавляет его к сообщению, и только после этого производит симметричное шифрование. Получатель, в свою очередь, расшифровывает полученное сообщение и проверяет корректность MAC. Несовпадение означает попытку модификации или сбой в передаче. Таким образом, обеспечиваются конфиденциальность, целостность и подлинность данных.

1.2.1.3 Протокол SAML

Основным стандартом, лежащим в основе федеративной идентификации, является SAML (Security Assertion Markup Language) [7].

Наиболее распространённый сценарий использования SAML – это единый вход (Single Sign-On, SSO) между доменами. Пользователю не нужно создавать отдельные учетные записи для каждого приложения, достаточно пройти аутентификацию один раз.

Протокол определяет следующие термины [8]:

- субъект (subject) – сущность, в отношении которой обмениваются информацией о безопасности, обычно это пользователь приложения;
- утверждение SAML (SAML Assertion) – сообщение в формате XML, содержащее информацию о безопасности субъекта;
- профиль SAML (SAML Profile) – спецификация, определяющая, как использовать сообщения SAML для конкретного бизнес-сценария;
- поставщик идентификаций (Identity Provider) – сервер, выпускающий утверждения SAML об аутентифицированном субъекте в контексте кросс-доменной аутентификации;
- поставщик услуг (Service Provider) – сервер, предоставляющий данные и услуги для субъекта, делегирует аутентификацию Identity Provider и полагается на информацию о субъекте, полученную в утверждении SAML;
- отношения доверия (Trust Relationship) – соглашение между поставщиком услуг и поставщиком идентификаций, в рамках которого первый доверяет утверждениям, выданным вторым.

Рассмотрим принцип работы стандарта SAML. Пользователь пытается получить доступ к защищённому ресурсу, размещённому у поставщика услуг (например, SaaS-приложению). Поскольку пользователь ещё не аутентифицирован, Service Provider инициирует запрос на аутентификацию и перенаправляет браузер пользователя к поставщику идентификаций. При успешной аутентификации пользователя Identity Provider формирует SAML-утверждение, содержащее информацию о субъекте, времени аутентификации и других атрибутах. Данное утверждение передаётся обратно Service Provider через браузер пользователя. Поставщик услуг проверяет подлинность SAML-утверждения, используя установленное доверие к поставщику идентификаций.

После успешной проверки пользователь получает доступ к ресурсу. При повторном обращении к сервису не потребуется аутентификация на стороне Identity Provider, так как он запомнит идентификатор сессии пользователя и автоматически вернет утверждение об аутентификации. SAML – браузерно-ориентированный протокол, который полагается на редиректы и веб-формы, что не подходит для мобильных или серверных API-сценариев.

1.2.1.4 Протокол OAuth 2.0

Протокол OAuth 2.0 (Open Authorization) [9] был разработан для того, чтобы приложения могли получать разрешение на вызов сторонних API. Например, получив согласие пользователя, приложение сможет вызвать API от его имени для получения определенных данных. OAuth 2.0 позволяет авторизовать приложение на доступ к API от имени пользователя; не передавать при этом личные данные пользователя (пароль, логин); ограничить доступ к данным пользователя и выполняемые с ними действия.

Протокол OAuth 2.0 является протоколом авторизации. Токен доступа предназначен только для доступа к API и не содержит информации о событии аутентификации или о самом пользователе. OAuth 2.0 определяет четыре роли, участвующие в процедуре авторизации [8]:

- сервер ресурсов – сервис, в котором хранятся защищённые ресурсы, к которым обращается приложение;
- владелец ресурса – пользователь или другой субъект, который владеет защищёнными ресурсами на сервере ресурсов;
- клиент – приложение, которому требуется доступ к ресурсам на сервере ресурсов от имени владельца ресурса;
- сервер авторизации – сервис, которому сервер ресурсов доверяет авторизовывать приложения для доступа к себе.

Спецификация OAuth 2.0 определяет четыре способа, с помощью которых приложение получает доступ к API [8]: использование гранта авторизационного входа, неявного гранта, гранта с использованием логина и пароля, гранта

клиентских учетных данных. Выбор конкретного типа учетных данных (гранта) для авторизации определяется особенностями сценария применения и типом приложения. Для авторизации данного класса субъектов подходят первые три, описание их приведено ниже.

Грант авторизационного кода (authorization code grant) применяется в случаях, когда приложение хочет получить доступ к защищённым ресурсам от имени пользователя, не обрабатывая его учетные данные самостоятельно. На первом этапе пользователь перенаправляется браузером на сервер авторизации, где проходит аутентификацию и предоставляет согласие на доступ приложения к определённым данным. После успешного подтверждения сервер возвращает авторизационный код – временный маркер, не содержащий информации о пользователе или правах доступа. На втором этапе приложение серверной частью обменивает авторизационный код на токен доступа, обращаясь к точке выдачи токенов сервера авторизации. Владея токеном доступа, приложение сможет выполнять запросы к API от имени пользователя в пределах согласованного набора прав.

Неявный грант (implicit grant) позволяет получить токен доступа в одном запросе и используется публичными клиентами (например, одностраничными приложениями, SPA). Изначально он создавался из-за ограничений браузеров, которые не поддерживали стандарт CORS. В случае механизма неявного гранта токен доступа возвращается приложению через редирект с фрагментом URL, что является небезопасным из-за рисков утечки. Поскольку сейчас большинство браузеров поддерживают политики CORS, для SPA предпочтительнее использовать грант авторизационного кода с PKCE (Proof Key for Code Exchange).

Грант с использованием логина и пароля (resource owner password credentials) позволяет приложению самостоятельно управлять личными учетными данными пользователя, отправлять их на сервер авторизации для получения токена доступа. Использование данного типа гранта не рекомендуется, так как он раскрывает учетные данные пользователя

приложению и не включает шаг согласия. Обычно используется в случаях миграции пользователей между системами, а также раньше применялся в мобильных приложениях.

1.2.1.5 Протокол OpenID Connect

Протокол OpenID Connect (OIDC) – протокол аутентификации, позволяет приложению делегировать аутентификацию пользователя серверу авторизации OAuth 2.0 и получить от него утверждения о пользователе и о самом событии аутентификации в стандартном формате [8]. Приложение перенаправляет браузер пользователя на сервер авторизации (OpenID Provider), реализующий OIDC. После аутентификации пользователя OpenID Provider может вернуть приложению данные пользователя в ID токене – специальном токене безопасности, а также токен доступа по стандарту OAuth 2.0. Полученный токен доступа приложение может использовать для доступа к ресурсам пользователя, а также при запросах к конечной точке UserInfo для получения полной информации о пользователе.

OIDC определяет три различных потока, с помощью которых приложение может взаимодействовать с сервером авторизации: поток авторизационного кода (authorization code flow), имплицитный поток (implicit flow), гибридный поток (hybrid flow). Назначение и принцип действия потоков аналогичен грантам в стандарте OAuth 2.0, ключевым отличием является назначение получаемых токенов. Помимо access token, предназначенного для вызова защищённых API, в рамках OIDC клиент также получает ID токен, содержащий утверждения о пользователе и событии аутентификации. Таким образом, основной целью OIDC является подтверждение личности пользователя и передача его идентификационных данных приложению, тогда как OAuth 2.0 используется исключительно для предоставления ограниченного доступа к ресурсам от имени пользователя.

В таблице 1 приведен сравнительный анализ перечисленных в данном разделе протоколов аутентификации конечных пользователей. Рассматриваются

ключевые параметры протоколов, отражающие их особенности и области применения.

Таблица 1 – Сравнительный анализ протоколов аутентификации конечных пользователей

Протокол	Тип шифрования	Класс	Архитектура	Протокол транспорта	Особенности	Применение
Kerberos	Симметричное	Аутентификация	Централизованная	TCP/UDP	Использование билетов	Корпоративные сети
TLS	Ассиметричное при установлении и соединении, после симметричное	Шифрование + аутентификация	Точка-точка	TCP	Использование сертификатов X.509, PKI	HTTPS, VPN
SAML	Не определено (XML-формат)	Аутентификация	Через посредника	HTTP (POST/Redirect)	Браузерно-ориентирован, XML, SSO	Корпоративные веб-приложения
OAuth	Не определено (работает поверх TLS)	Аутентификация	Через посредника	HTTP/HTTPS	Взаимодействие с API, Access Token	Используется только для доступа к защищённым ресурсам, без раскрытия личной информации пользователя.
OIDC	Не определено (работает поверх OAuth)	Аутентификация	Через посредника	HTTP/HTTPS	Взаимодействие с API, ID Token	Надстройка над OAuth 2.0, используется, когда нужно определить личность пользователя и затем выдать доступ к ресурсам.

Выполненный анализ показывает, что перечисленные протоколы решают разные задачи в зависимости от сценария участия пользователя. Kerberos, TLS и SSH обеспечивают безопасный обмен данными между узлами, но не всегда

напрямую вовлекают пользователя. В то же время, SAML, OAuth 2.0 и OpenID Connect ориентированы на взаимодействие с пользователем: SAML чаще используется для входа в веб-приложения внутри организаций, OAuth 2.0 – для предоставления доступа к пользовательским данным в сторонних сервисах, а OIDC – для обеспечения единой аутентификации пользователя в разных компонентах системы.

1.2.2 Аутентификация сервисов и компонентов систем

Для аутентификации сервисов и компонентов как внутри одной системы, так и внешних систем применяются схожие протоколы, поэтому протоколы аутентификации этих классов субъектов будут рассмотрены вместе. Основное различие заключается в контексте их применения, уровне доверия и требованиях безопасности. В обоих случаях речь идет о машинной аутентификации, где взаимодействие для проверки подлинности и авторизации происходит между сервисами.

У монолитных приложений обычно немного точек входа и большинство компонентов недоступны извне. Поэтому при монолитной архитектуре безопасность обеспечивается централизованно, и отдельные компоненты не нуждаются в аутентификации, если в этом нет особой необходимости.

В микросервисной архитектуре каждый микросервис является отдельной точкой входа, что существенно расширяет потенциальную поверхность атаки и требует реализации механизмов защиты на уровне каждого сервиса. В таких условиях особенно важно внедрение аутентификации между компонентами – она позволяет удостовериться, что каждый микросервис действительно является тем, за кого себя выдаёт, и имеет право инициировать взаимодействие [10]. Иначе система становится уязвимой к подмене запросов, вмешательству со стороны третьих лиц и неконтролируемому доступу к данным.

Для машинной аутентификации рассмотрим следующие протоколы: mTLS (Mutual TLS), OAuth 2.0 с использованием Client Credentials Grant, подписанные JWT, API Keys.

1.2.2.1 **Протокол mTLS**

Протокол mutual TLS (mTLS) представляет собой расширение стандартного TLS, обеспечивающее не только проверку подлинности сервера клиентом, но и аутентификацию клиента сервером. Это гарантирует, что оба участника – легитимные субъекты и могут доверять друг другу.

Сервер дополнительно запрашивает у клиента его сертификат. Клиент, в ответ, предоставляет X.509-сертификат и формирует специальное сообщение CertificateVerify, в котором подписывает предыдущие handshake-сообщения своим закрытым ключом. Сервер проверяет сертификат, цифровую подпись сообщения и убеждается, что клиент действительно обладает приватным ключом, соответствующим предоставленному сертификату.

Также необходимо учитывать, что полноценная работа протокола mTLS возможна только при наличии инфраструктуры открытых ключей (PKI). Каждый участник взаимодействия должен иметь действительный X.509-сертификат, выданный общим доверенным центром сертификации. В условиях распределённой архитектуры, в которой много компонентов обмениваются зашифрованными запросами, централизованная PKI и доверие к общему СА становятся необходимыми элементами системы.

1.2.2.2 **Протокол OAuth 2.0 Client Credentials Grant**

Протокол OAuth 2.0 с использованием других грантов подробно описан в разделе 1.1.3.1. В контексте машинной аутентификации будет описан механизм работы гранта Client Credentials Grant. В данном случае приложение действует не от имени пользователя, а от своего имени, поэтому не нужно требовать аутентификацию пользователя и разрешение на доступ к его данным [11]. Такой подход используется, если запрашивается доступ к данным конкретного приложения.

Сервис, которому нужно обратиться к другому защищённому API, предварительно регистрируется в системе авторизации (в OAuth-провайдере) и получает уникальные идентификатор и пароль: client ID и client secret. Провайдер

авторизации выступает единым доверенным посредником между всеми участниками федерации. Кроме учетных данных клиентов, OAuth-провайдер хранит для каждого клиента области доступа (scopes) – перечень API и операций, к которым клиент может запрашивать доступ. Данные области конфигурируются администратором системы или автоматически в процессе настройки клиента.

Для обращения к сервису ресурсов, клиенту необходимо получить токен доступа у провайдера авторизации. Сервис авторизации, получив запрос, проверяет корректность учетных данных клиента и наличие в конфигурации запрошенных областей доступа. Если всё верно, он выдает клиенту токен доступа, подтверждающий, что клиент имеет право обращаться к соответствующему ресурсу. Сервер ресурсов затем проверяет подпись и содержание токена, чтобы удостовериться, что вызов разрешён.

1.2.2.3 Стандарт JWT

Стандарт JWT (JSON Web Token) [12] предназначен для безопасного обмена информацией в формате JSON. Он включает в себя утверждения (claims), содержащие сведения о субъекте, и другие сопутствующие данные. Для обеспечения подлинности и целостности токен подписывается с помощью секретного ключа или криптографической пары ключей, используя алгоритмы HMAC или RSA. JWT состоит из трех частей, разделенных точками:

- header (заголовок) содержит информацию о типе токена и алгоритме подписи;
- payload (полезная нагрузка) содержит claims – данные, которые необходимо передать;
- signature (подпись) обеспечивает целостность токена, позволяя проверять, что он не был изменен.

В основном JWT решает две ключевые задачи в архитектуре безопасности микросервисов: защита взаимодействия между сервисами и передача информации о конечном пользователе между микросервисами [10]. В контексте взаимодействия микросервисов рассмотрим первый вариант. Для

идентификации микросервисов используется механизм самоподписанных токенов. Подобно mTLS, каждый микросервис должен обладать своей собственной парой ключей: публичным и приватным. Микросервис создаёт JWT, подписывает его своим приватным ключом и передаёт в HTTP-заголовке вместе с запросом получателю по защищенному каналу. Принимающий микросервис проверяет подпись JWT, используя соответствующий публичный ключ, и таким образом может установить, кто отправил запрос. В отличие от mTLS, JWT не создаёт защищённого канала передачи данных, он применяется для идентификации сервисов и передачи контекстной информации. Поэтому при использовании самоподписанных JWT необходимо дополнительно применять TLS для обеспечения конфиденциальности и целостности передаваемых данных.

Кроме самоподписанных токенов, JWT могут выдаваться централизованным доверенным сервером аутентификации. В этом сценарии микросервисы не подписывают токены самостоятельно, а запрашивают их у доверенного сервера, который сначала проверяет их подлинность (например, через mTLS или client credentials). Такой подход напоминает OAuth 2.0 Client Credentials Flow с JWT-токенами, где сервис аутентифицируется с помощью client_id и client_secret, а в ответ получает JWT с необходимыми данными. Ключевое отличие от классического OAuth 2.0 – акцент на криптографической подписи токена, а не на протоколе обмена. Сервер аутентификации выступает единым центром доверия, а микросервисы проверяют токены через его публичный ключ, что исключает необходимость обращений к /introspect-конечной точке API, как в случае с opaque-токенами.

1.2.2.4 Механизм API-ключей

API-ключ – это уникальный идентификатор, представляющий собой строку символов, который используется для аутентификации и авторизации клиентов при обращении к API. Он служит своего рода «пропуском», подтверждающим право доступа к определённым функциям или данным API [10]. Клиент регистрируется на платформе поставщика API и получает

секретный ключ. При каждом обращении к API клиент включает API-ключ в запрос, обычно в заголовке или параметрах URL. Сервер API проверяет валидность ключа и определяет, имеет ли он право на выполнение запрашиваемой операции.

В микросервисной архитектуре API-ключи часто выдаются централизованным провайдером авторизации, которому доверяют все сервисы внутри системы. Ключи используются для идентификации вызывающей стороны при взаимодействии между сервисами и позволяют ограничить доступ к определённым API. API-ключи удобны в простых сценариях, где между участниками уже установлено доверие (например, внутри одного контура микросервисов), или при доступе к открытым данным с низким уровнем риска. Они легко реализуются, но для защиты конфиденциальной информации должны использоваться вместе с другими средствами безопасности, такими как TLS, проверка IP-адресов или ограничение прав доступа.

В таблице 2 приведен сравнительный анализ рассмотренных протоколов машинной аутентификации. Данный сравнительный анализ позволяет лучше понять применимость каждого из них в различных сценариях защиты данных и взаимодействия сервисов. Особое внимание уделено таким аспектам, как цель протокола, механизмы аутентификации, архитектурные особенности и требования к инфраструктуре для каждого протокола, что позволяет оценить возможности и ограничения каждого протокола в контексте машинной аутентификации.

Таблица 2 – Сравнительный анализ протоколов машинной аутентификации

Критерий	mTLS	OAuth 2.0	Подписанные JWT	API Keys
Цель	Построение защищенного канала связи с взаимной аутентификацией	Аутентификация и авторизация машинных клиентов для получения доступа к защищённым ресурсам от имени самого клиента	Передача информации об идентичности и/или правах субъекта между участниками системы в самодостаточном формате	Идентификация вызывающей стороны и контроль доступа к API в простых сценариях

Продолжение таблицы 2

Критерий	mTLS	OAuth 2.0	Подписанные JWT	API Keys
Механизм аутентификации	Сертификаты X.509	Access Token	JWT токен	Уникальный ключ
Кто выдает/проверяет	PKI/CA	Провайдер авторизации	Локальная проверка подписи	Самодостаточный API-сервис или централизованный компонент авторизации
Архитектура	Точка-точка	Централизованная	Централизованная выдача, локальная проверка	Централизованная или на уровне отдельного API-сервиса
Особенности	Необходима инфраструктура PKI, определение идентичности клиента без контекста данных	Необходимы защита канала (TLS), безопасное хранение client credentials	Необходима защита канала (TLS), целостная передача контекста	Простота реализации, нет встроенной поддержки авторизации

Проведенный анализ показывает, что выбор протоколов машинной аутентификации зависит от требований к безопасности, сложности инфраструктуры и сценариев использования. Ключевым компромиссом остается баланс между уровнем защиты и простотой внедрения.

1.3 Постановка задачи на разработку программных средств

В ходе выполнения выпускной квалификационной работы необходимо разработать подсистему обеспечения единой аутентификации внешних систем в рамках информационной системы электронного документооборота. Информационная система электронного документооборота (ИС ЭДО) – это программно-аппаратный комплекс, предназначенный для юридически значимого подписания электронных документов между участниками. Одно из требований к подсистеме – высокая степень переносимости, а также легкость и дешевизна при необходимости поддержания аутентификации новой внешней

системы с известным, либо неизвестным алгоритмом аутентификации. Решение должно поддерживать стандартные протоколы аутентификации (OAuth 2.0, JWT, TLS и др.) и предоставлять гибкие настройки политик доступа.

На основе проведенного анализа предметной области и существующих решений в рамках разработки необходимо выполнить следующие задачи:

- определить требования к подсистеме единой аутентификации с учётом анализа предметной области;
- выбрать и обосновать основные протоколы аутентификации, подходящие для заданного сценария;
- спроектировать архитектуру системы, включая модули аутентификации и авторизации, систему управления правами доступа;
- реализовать ключевые компоненты подсистемы;
- протестировать работоспособность решения в различных сценариях взаимодействия.

Разрабатываемая подсистема должна обеспечивать:

- возможность подключения новых внешних систем с известными алгоритмами аутентификации/авторизации;
- возможность быстрого и легкого встраивания нового алгоритма аутентификации в случае, если при развитии ИС ЭДО потребуется поддержать новую внешнюю систему с ранее неизвестным алгоритмом аутентификации/авторизации;
- ведение журнала протоколирования событий взаимодействия с внешними системами для обеспечения требования сопровождаемости ИС ЭДО;
- единый интерфейс аутентификации для ИС ЭДО.

В результате разработки должна быть создана подсистема обеспечения механизма единой аутентификации, позволяющая внешним системам, поддерживающим разные стандарты и протоколы аутентификации, проходить аутентификацию и получать доступ к данным ИС ЭДО. Таким образом, создаваемая подсистема станет универсальным шлюзом аутентификации,

обеспечивающим гибкость, расширяемость и безопасность взаимодействия между ИС ЭДО и разнообразными внешними информационными системами.

1.4 Функционально-стоимостной анализ

Функционально-стоимостной анализ (ФСА) – метод системного исследования, направленный на снижение затрат путём анализа функций объекта и их стоимости. Согласно [13], основная идея ФСА заключается в том, что потребитель приобретает не просто продукт, а функционал, который этот продукт предоставляет. ФСА помогает выявить важность каждой отдельной функции, оценить её практическую значимость для пользователя и сопоставить с затратами на её реализацию в конечном продукте. Повышение ценности продукта достигается двумя способами: либо через сокращение расходов на производство, следовательно снижение цены, либо через расширение полезных характеристик продукта.

Проведем функционально-стоимостной анализ для определения стоимости и целесообразности разработки программного продукта. Определим продолжительность реализации продукта, а также стоимость каждого из этапов разработки. Для определения стоимости разработки системы обеспечения единой аутентификации проанализируем уровень заработной платы программиста с января по апрель 2025 года в городе Пенза. Диаграмма средней заработной платы программиста за указанный период приведена на рисунке 1 [14].

На основании приведенной диаграммы средняя заработная плата программиста составляет 73 475 рублей при 40-часовой рабочей недели. Следовательно, программист получает за каждый час работы приблизительно 459 рублей.

Для представления выполнения выпускной квалификационной работы была построена контекстная диаграмма в нотации IDEF0 в среде моделирования Ramus (рисунок 2). Контекстная диаграмма – это самая верхнеуровневая модель

в IDEF0 (уровень A-0), которая определяет границы системы и её взаимодействие с внешней средой.

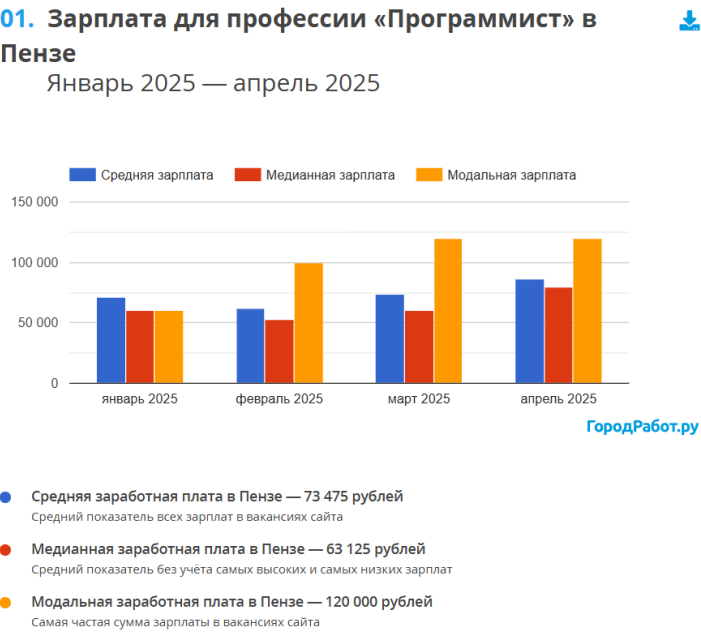


Рисунок 1 – Диаграмма средней заработной платы программиста в 2025 году в городе Пенза

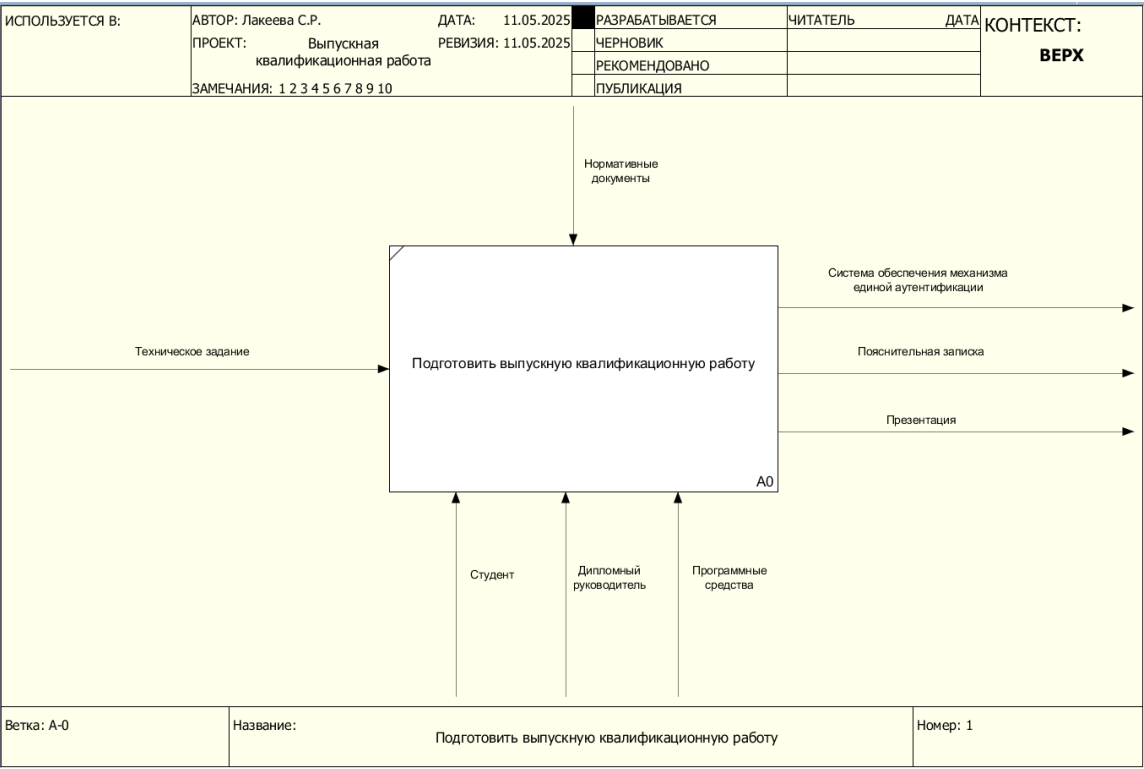


Рисунок 2 – Контекстная диаграмма процесса выполнения выпускной квалификационной работы

Контекстная диаграмма состоит из единственного функционального блока и четырех видов стрелок: вход, выход, механизм, управление, отражающих связи объекта с окружающей средой [15]. Функциональные блоки потребляют входные данные для формирования результатов своей деятельности. Управления определяют условия или правила, необходимые функции для получения результата. Выходы представляют итоговый результат выполнения функции. Механизмы – ресурсы, используемые в ходе работы.

На уровне А-0 функция «Подготовить выпускную квалификационную работу» преобразует входные данные (техническое задание) с использованием знаний студента, консультаций научного руководителя и программных средств (ПС), под управлением нормативных документов, в итоговый результат – пояснительную записку, презентацию и программный продукт.

Для более детального анализа процесса подготовки выпускной квалификационной работы (ВКР) проведём декомпозицию верхнеуровневой функции А-0, выделив основные подпроцессы, отражающие ключевые этапы выполнения ВКР. Диаграмма уровня А0 изображена на рисунке 3.

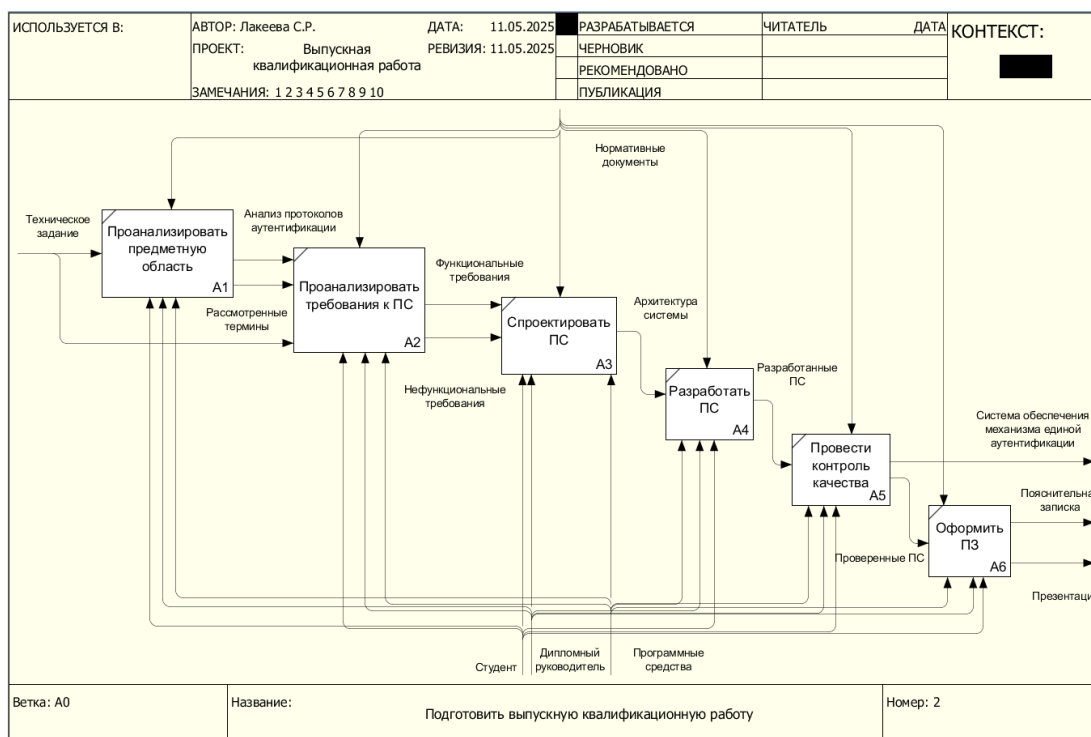


Рисунок 3 – Декомпозиция функции «Подготовить выпускную квалификационную работу»

В результате декомпозиции основного бизнес-процесса «Подготовить выпускную квалификационную работу» было получено шесть подпроцессов: «Проанализировать предметную область», «Проанализировать требования к ПС», «Спроектировать ПС», «Разработать ПС», «Провести контроль качества» и «Оформить ПЗ».

Для планирования, учета бюджета и визуального контроля выполнения проекта во времени были составлены план разработки проекта (рисунок 4) и диаграмма Ганта (рисунок 5).

Задача		Начало	Длительность	Завершение
		07.04.2025	400ч	13.06.2025
1	☐ Подготовить ВКР	07.04.2025	400ч	13.06.2025
1.1	Проанализировать предметную область	07.04.2025	56ч	15.04.2025
1.2	Проанализировать требования к ПС	16.04.2025	32ч	21.04.2025
1.3	Спроектировать ПС	22.04.2025	80ч	05.05.2025
1.4	Разработать ПС	06.05.2025	168ч	03.06.2025
1.5	Провести контроль качества	04.06.2025	40ч	10.06.2025
1.6	Оформить ПЗ	11.06.2025	24ч	13.06.2025

Рисунок 4 – План разработки проекта

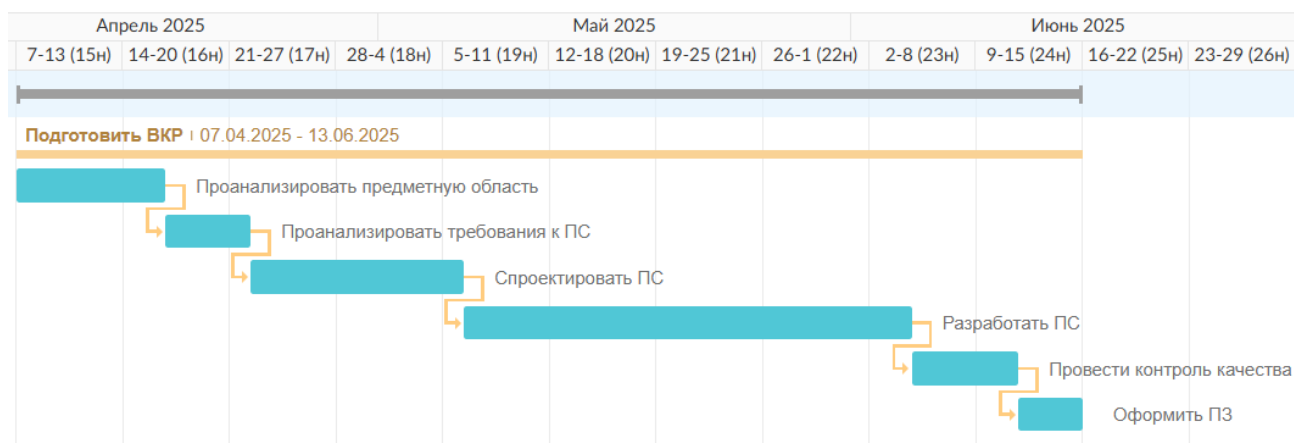


Рисунок 5 – Диаграмма Ганта

Для оценки стоимости проекта был выполнен анализ трудозатрат, отражающий объём времени (в часах), необходимый для реализации каждого этапа разработки проекта, а также соответствующую стоимость выполнения работ. Сводные данные представлены в таблице 3.

Таблица 3 – Затраты на разработку проекта

Этап	Продолжительность	Зарплата	Стоимость
Проанализировать предметную область	56 ч	459 руб/ч	25 704 руб
Проанализировать требования к ПС	32 ч	459 руб/ч	14 688 руб
Спроектировать ПС	80 ч	459 руб/ч	36 720 руб
Разработать ПС	168 ч	459 руб/ч	77 112 руб
Провести контроль качества	40 ч	459 руб/ч	18 360 руб
Оформить ПЗ	24 ч	459 руб/ч	11 016 руб
Итого:	400 ч	459 руб/ч	183 600 руб

Из результатов проведенного функционально-стоимостного анализа получили, что суммарные временные затраты на разработку системы обеспечения механизма единой аутентификации и оформление пояснительной записки будут составлять 400 часов, а стоимость выполнения равняться 183 600 рублей.

2 Анализ требований к разрабатываемым средствам

2.1 Анализ функциональных требований

Функциональные требования – это описание того, что должна делать система, какие функции и услуги она должна предоставлять пользователям или другим системам. Они формализуют основную бизнес-логику системы и задают перечень операций, которые система должна выполнять в рамках своей области применения. Функциональные требования:

- отражают назначение системы и её основные функции;
- включают сценарии обработки входных данных, реакции на события, доступ к данным и выполнение бизнес-операций;
- определяют интерфейсы, потоки данных, а также допустимые действия для различных ролей или внешних субъектов.

В рамках данной работы необходимо разработать подсистему, обеспечивающую механизм единой аутентификации и авторизации внешних систем для доступа к функциональным компонентам ИС ЭДО. Разрабатываемый механизм должен выполнять роль шлюза, проверяющего легитимность внешних систем перед предоставлением доступа к конфиденциальным данным ИС ЭДО. Цель решения – минимизировать риски утечки данных за счет централизованного управления аутентификацией всех подключаемых внешних систем.

Перечислим функциональные требования к разрабатываемой подсистеме. Решение должно:

- поддерживать механизм аутентификации компонентов ИС ЭДО;
- предоставить внешним системам механизм проверки подлинности с поддержкой стандартных протоколов аутентификации;
- предоставить настройку прав доступа внешних систем;
- обеспечить безопасное хранение идентификационной информации;
- обеспечить протоколирование всех попыток аутентификации и операций с данными.

– для моделирования контекста системы – используется для определения внешних участников (акторов) и их взаимодействия с системой, а также для очерчивания границ самой системы;

[illegible]

Рисунок 6 – Диаграмма вариантов использования

Сценарии выполнения некоторых вариантов использования приведены на рисунках 7 - 11.

Наименование: настраивать права доступа для клиентов
ID: 1
Краткое описание: администратор настраивает допустимые области видимости для каждой подключенной внешней системы
Главные акторы: администратор
Второстепенные акторы: отсутствуют
Предусловия: администратор прошел аутентификацию в системе
Основной поток:
1. Администратор просматривает список подключенных клиентов
2. Администратор настраивает разрешенные области видимости для клиентов
Постусловия: настроены области видимости

Рисунок 7 – Описание сценария варианта использования «Настраивать права доступа для систем»

Наименование: управлять подключенными клиентами
ID: 2
Краткое описание: администратор управляет подключенными клиентами
Главные акторы: администратор
Второстепенные акторы: отсутствуют
Предусловия: администратор прошел аутентификацию в системе
Альтернативный поток:
1. Администратор может заблокировать клиента
2. Администратор может разблокировать клиента
Постусловия: клиент заблокирован/разблокирован

Рисунок 8 – Описание сценария варианта использования «Управлять подключенными системами»

Наименование: зарегистрироваться в системе
ID: 3
Краткое описание: внешняя система проходит регистрацию в системе единой аутентификации
Главные акторы: внешняя система
Второстепенные акторы: отсутствуют
Основной поток: данные клиента сохраняются в базе данных
Второстепенный поток: генерируются аутентификационные данные и возвращаются внешней системе
Постусловия: внешняя система зарегистрирована

Рисунок 9 – Описание сценария варианта использования «Зарегистрироваться в системе»

Наименование: аутентифицироваться в системе
ID: 4
Краткое описание: внешняя система проходит аутентификацию в системе единой аутентификации
Главные акторы: внешняя система
Второстепенные акторы: отсутствуют
Основной поток: проверяются аутентификационные данные внешней системы
Второстепенный поток: при успешной аутентификации предоставляется средство доступа
Постусловия: внешняя система аутентифицирована/ошибка аутентификации

Рисунок 10 – Описание сценария варианта использования «Пройти аутентификацию»

Наименование: отправить запрос на получение данных
ID: 5
Краткое описание: внешняя система отправляет запрос на получение данных ИС ЭДО
Главные акторы: внешняя система
Второстепенные акторы: отсутствуют
Основной поток: <ol style="list-style-type: none"> 1. Проверяется валидность средства доступа. 2. Проверяются права внешней системы. 3. Подсистема проходит аутентификацию в ИС ЭДО. 4. Подсистема получает данные ИС ЭДО.
Постусловия: получены данные ИС ЭДО/ошибка авторизации

Рисунок 11 – Описание сценария варианта использования «Отправить запрос на получение данных»

Построенная диаграмма вариантов использования точно отражает функциональные требования, предъявляемые к системе, а также показывает, как подсистема единой аутентификации реагирует на действия внешних акторов.

2.2 Анализ нефункциональных требований

Нефункциональные требования – это характеристики и ограничения, определяющие качество работы системы, а не её поведение. В отличие от функциональных требований, описывающих, что должна делать система, нефункциональные указывают, как система должна это делать с точки зрения производительности, безопасности, надёжности, удобства сопровождения и других аспектов.

Учитывая, что данные характеристики напрямую влияют на проектные решения подсистемы, перечислим нефункциональные требования в

соответствии с классификацией, основанной на структурных и сквозных свойствах архитектуры [17] программного обеспечения:

- подсистема должна предоставлять возможности для гибкой настройки прав доступа внешних систем, включая определение допустимых областей видимости и срока действия средств аутентификации (конфигурируемость);

- механизм аутентификации должен быть доступен для различных внешних систем, использующих разные технологии и протоколы (совместимость);

- подсистема должна реализовать механизм единой аутентификации внешних систем, поддерживающих разные стандарты и протоколы (аутентифицируемость);

- доступ к ресурсам ИС ЭДО должен предоставляться на основе сконфигурированных для каждой системы прав, определяющих допустимые действия и области данных, к которым разрешён доступ (авторизуемость);

- подсистема должна обеспечивать простую интеграцию новых компонентов для авторизации внешних систем, поддерживающих другие стандарты аутентификации, без необходимости существенной переработки существующих модулей (сопровождаемость);

- подсистема должна вести протоколирование ключевых событий для повышения прозрачности работы системы.

Перечисленные нефункциональные требования послужат основой для обоснованного выбора архитектурных решений и дальнейшего проектирования системы.

3 Проектирование программных средств

3.1 Общая архитектура программных средств

В рамках проектируемого механизма единой аутентификации задействованы следующие компоненты:

- интерфейс администрирования – интерфейс для настроек прав доступа внешних систем, управления внешними системами;
- подсистема единой аутентификации (ЕА) – программный компонент, обеспечивающий аутентификацию внешних систем и взаимодействие с ИС ЭДО;
- функциональные компоненты ИС ЭДО, содержащие запрашиваемые внешними системами ресурсы, обрабатывают запросы от подсистемы ЕА;
- внешние системы – сторонние сервисы/системы, инициирующие запросы к ИС ЭДО через проектируемую подсистему ЕА.

Для отображения структуры взаимодействий между объектами подсистемы единой аутентификации была построена диаграмма коммуникации [16] (рисунок 12).

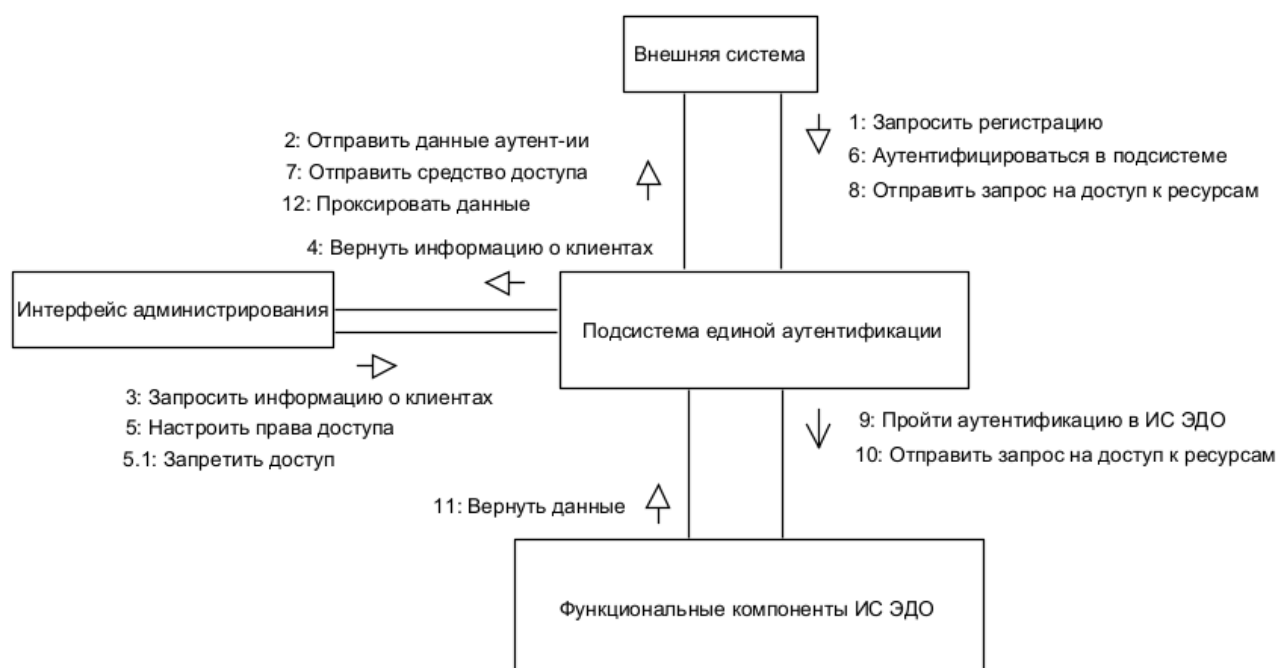


Рисунок 12 – Диаграмма коммуникации

Подсистема единой аутентификации реализует механизм проверки подлинности внешних систем, запрашивающих доступ к ресурсам ИС ЭДО. С учетом требований к сопровождаемости, гибкости настройки прав доступа, поддержке различных протоколов аутентификации внешних систем при проектировании подсистемы выбрана архитектура на основе сервисов с независимыми базами данных в каждом сервисе [17]. Основное отличие от микросервисной архитектуры заключается в уровне детализации и размере сервисов: сервисы проектируются как крупноблочные, каждый из которых охватывает широкую функциональную область.

Для отражения архитектуры подсистемы с точки зрения ее компонентов и взаимосвязей построена диаграмма компонентов [16] (рисунок 13). Диаграмма компонентов позволяет наглядно представить составные части подсистемы единой аутентификации, определить границы их ответственности и отследить взаимодействие между ними.

Описание компонентов приведено в таблице 4.

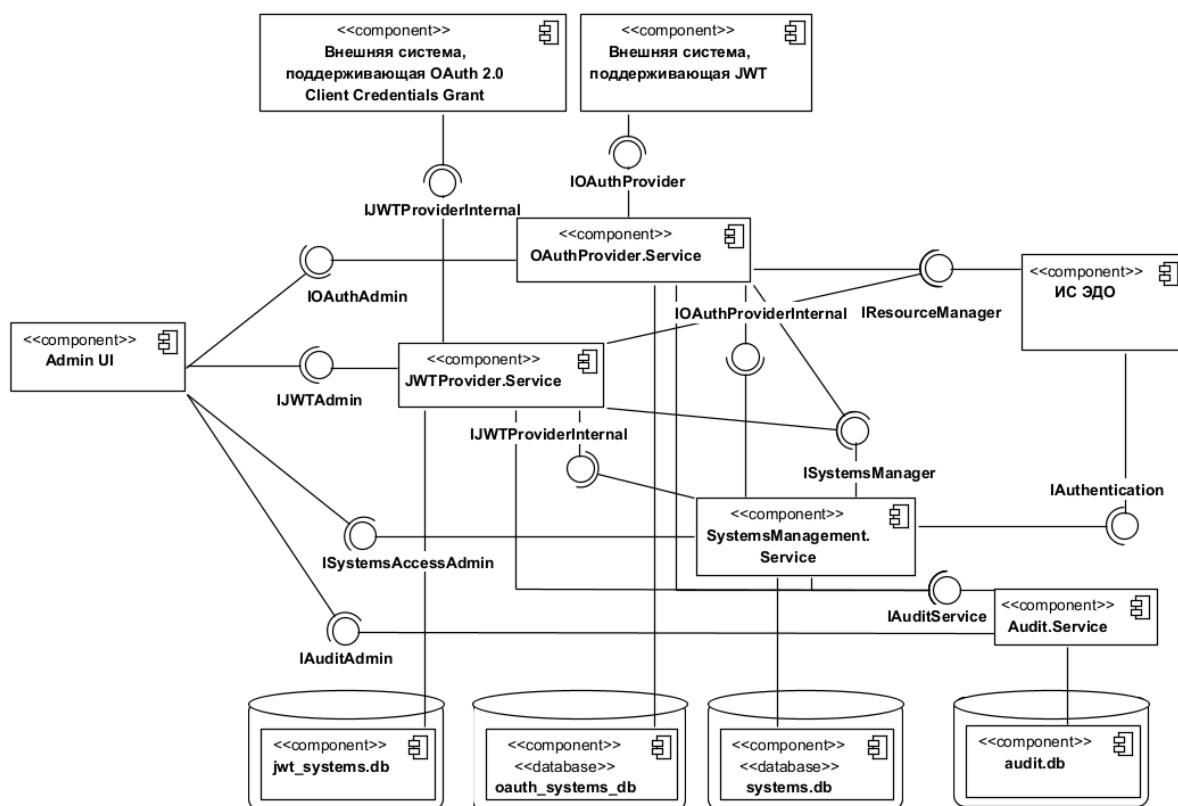


Рисунок 13 – Диаграмма компонентов

Таблица 4 – Описание компонентов подсистемы единой аутентификации

Название компонента	Описание
Admin UI	Интерфейс для администрирования внешних систем.
OAuthProvider.Service	Сервис, поддерживающий протокол аутентификации OAuth 2.0 Client Credentials внешних систем. Предоставляет интерфейсы: IOAuthProvider для внешних систем, IOAuthAdmin для веб-интерфейса администрирования, IOAuthProviderInternal для внутреннего взаимодействия с сервисом управления системами.
JWTProvider.Service	Сервис, поддерживающий протокол аутентификации внешних систем по JWT токенам. Предоставляет интерфейсы: IJWTProvider для внешних систем, IOAuthAdmin для веб-интерфейса администрирования, IJWTProviderInternal для внутреннего взаимодействия с сервисом управления системами.
SystemsManagement.Service	Сервис, агрегирующий данные зарегистрированных внешних систем. Предоставляет функционал для настроек прав доступа систем и поддержки протокола аутентификации компонентов ИС ЭДО. Реализует интерфейс ISystemsManager, предназначенный для сервисов аутентификации с целью получения JWT токенов авторизации в ИС ЭДО.
Audit.Service	Сервис протоколирования событий, предоставляет интерфейсы IAuditAdmin для веб-интерфейса администрирования и IAuditService для сбора протоколируемой информации от остальных сервисов.
oauth_systems.db	База данных сервиса поддержки протокола OAuth 2.0 Client Credentials.
systems.db	База данных сервиса управления внешними системами.
jwt_systems.db	База данных сервиса поддержки протокола JWT.
audit.db	База данных сервиса протоколирования событий.

В рамках реализации выпускной квалификационной работы предусмотрена поддержка двух наиболее распространённых протоколов аутентификации: JWT и OAuth 2.0 Client Credentials Grant. JWTProvider.Service обеспечивает регистрацию внешних систем, выпуск и верификацию JWT-токенов в соответствии с заданным протоколом. OAuthProvider.Service реализует регистрацию внешних систем, выпуск и валидацию access-токенов в рамках

протокола OAuth 2.0, обеспечивает безопасную авторизацию и управление доступом для доверенных систем без участия пользователя.

Архитектура подсистемы построена по принципу модульности и расширяемости, что позволяет в дальнейшем легко интегрировать поддержку дополнительных протоколов аутентификации и авторизации без необходимости внесения значительных изменений в основной функционал системы. Добавление поддержки нового протокола сводится к разработке соответствующего модуля, реализующего стандартизированный интерфейс взаимодействия с внешней системой и внутренний интерфейс взаимодействия с компонентами подсистемы.

3.2 Проектирование интерфейса администрирования

Интерфейс администрирования предназначен для управления доступом внешних систем к данным ИС ЭДО через подсистему единой аутентификации. Он должен предоставить администратору удобный способ настройки параметров взаимодействия, включая управление подключенными внешними системами, определение областей доступа, работу с журналом событий. Интерфейс реализуется в виде веб-приложения.

Приведем основной функционал интерфейса:

- а) управление внешними системами:
 - 1) просмотр общей информации о зарегистрированных внешних системах;
 - 2) блокировка/разблокировка внешней системы;
 - 3) настройка прав доступа внешних систем;
 - 4) просмотр журнала событий;
- б) управление конфигурацией OAuth 2.0:
 - 1) просмотр информации по зарегистрированным внешним системам, использующим протокол OAuth 2.0 Client Credentials;
 - 2) просмотр информации по выданным токенам доступа;
 - 3) установка срока действия токена доступа;

- 4) просмотр журнала событий в рамках функционирования механизма аутентификации по протоколу OAuth 2.0;
- в) управление конфигурацией JWT:
 - 1) просмотр информации по зарегистрированным внешним системам, использующим стандарт JWT;
 - 2) установка срока действия токена JWT;
 - 3) просмотр журнала событий в рамках функционирования механизма аутентификации по протоколу JWT.

Для удобства взаимодействия администратора с подсистемой единой аутентификации каждый функциональный блок вынесен в отдельный раздел интерфейса. В рамках интерфейса предусмотрены следующие основные разделы: «Главная» – содержит общую информацию об использовании механизма единой аутентификации внешними системами, «Панель управления OAuth 2.0» и «Панель управления JWT» – позволяют настраивать основную конфигурацию и просматривать данные в рамках взаимодействия с внешними системами по указанным протоколам.

3.3 Проектирование подсистемы единой аутентификации

В ходе проектирования подсистемы единой аутентификации выделено четыре самостоятельных сервиса:

- сервис аутентификации на основе OAuth 2.0 Client Credentials;
- сервис аутентификации на основе JWT;
- сервис управления внешними системами;
- сервис протоколирования событий.

Сервис аутентификации на основе OAuth 2.0 Client Credentials реализует механизм аутентификации внешних систем, поддерживающих данный протокол, для обращения к ресурсам ИС ЭДО. Сервис предоставляет функционал для регистрации внешней системы, генерации аутентификационных данных, выдачи токена доступа в формате JWT на основе разрешенных областей доступа (scopes) и проверки корректности его подписи. В базе данных сервиса будут содержаться

данные внешних систем, включая идентификаторы, секреты, разрешенные области доступа и другое. В таблице 5 описано REST API сервиса аутентификации на основе OAuth 2.0 Client Credentials.

Таблица 5 – Описание REST API сервиса аутентификации на основе OAuth 2.0 Client Credentials

HTTP Метод	Описание	Входная структура	Выходная структура	Статусы
POST api/oauth/ clients/new	Регистрация клиента	{ «client_name»: «CRM-система отдела X», «origins»: [«https://example.com/api», «https://example.com/api2»] }	{ «client_id»: «client_c9b1d8f2f62c4b2ebf5 61333a5cc9c4d», «client_secret»: «V7G3X7dCmgvxCVZ0VRH Jfg==», «allowed_scopes»: [«read:scopes»:] }	Status200OK – Успешная регистрация клиента. Status422Unprocessable Entity – При регистрации клиента произошла ошибка. Status423Locked – Клиент уже зарегистрирован.
GET api/oauth/ clients/{c lient_id}/ scopes	Получение разрешенных областей доступа клиента	{ «client_id»: «JH97chis» }	[«read:data», «write:data»]	Status200OK – Области доступа успешно получены. Status401Unauthorized – Ошибка авторизации. Status422Unprocessable Entity – При получении областей доступа клиента произошла ошибка.
GET .well- known/op enid- configura tion	Получение конфигурации	–	{ «registration_ specification_url»: «https://auth.example.com/clie nts/register/openapi.json», «client_scopes_ specification_url»: «https://auth.example.com/clie nts/allowed- scopes/openapi.json», «token_endpoint»: https://auth.example.com/conn ect/token , «resource_api_specification_u rl»: https://auth.example.com/reso urce- proxy/protected/openapi.json, «scopes_allowed»: ["scopes.read", "datetime.read", "datetime.write", "data.write" }] }	Status200OK – Метаданные сервиса успешно получены.

Продолжение таблицы 5

HTTP Метод	Описание	Входная структура	Выходная структура	Статусы
GET api/oauth/ managem ent/clients	Получение информац ии о клиентах	–	[{ «client_id»: «system-x», «clientName»: «CRM- система отдела X», «enabled»: true, «created_at»: «2025 - 04- 24T10:15:30 Z», «allowed_scopes»: [read:data», «write:data»] }]	Status200OK – Данные клиентов успешно получены. Status401Unauthorized – Ошибка авторизации. Status422Unprocessable Entity – При получении данных клиентов произошла ошибка.
POST /connect/to ken	Получение токена доступа	client_id=client1 client_secret=secret1 grant_type=client_cre dentials scope=api1	{ «access_token»: «2YotnFZFEjr1zCsicM WpAA», «token_type»: «bearer», «expires_in»: 3600, }	200OK – токен успешно выдан, тело ответа содержит JSON с токеном; 400BadRequest – неверные или отсутствующие параметры в запросе; 401Unauthorized – неверная аутентификация клиента; 403Forbidden – клиент не имеет прав запрашивать указанные области доступа.
GET api/oauth/ managem ent/tokens	Получение информац ии по всем токенам	–	[{ «value»: «eyJhbGciOi...K9x3j18Q »», «creation_time»: «», «lifetime»: 3600, «client_id»: «», «scopes»: [«read:api1», «write:api2»] }]	Status200OK – Данные токенов успешно получены. Status401Unauthorized – Ошибка авторизации. Status422Unprocessable Entity – При получении данных токенов произошла ошибка.
PUT api/oauth/ managem ent/tokens/e xpiration?e xp_time=3 600	Настройка срока действия токена	exp_time – время действия токена	–	Status200OK – Срок действия токена успешно изменен. Status401Unauthorized – Ошибка авторизации. Status422Unprocessable Entity – При изменении срока действия токена произошла ошибка.

Окончание таблицы 5

HTTP Метод	Описание	Входная структура	Выходная структура	Статусы
GET api/oauth/resources/datetime	Получение даты и времени системы		«2025-05-11T14:30:45.1234567»	Status200OK – Данные успешно получены. Status401Unauthorized – Ошибка авторизации. Status422UnprocessableEntity – При получении даты и времени произошла ошибка.
PUT api/oauth/resources/datetime?datetime=2025-05-11T14:30:45.1234567	Установка даты и времени системы	datetime – дата и время	–	Status200OK – Данные успешно установлены. Status401Unauthorized – Ошибка авторизации. Status422UnprocessableEntity – При установке даты и времени произошла ошибка.
POST api/oauth/resources/data	Отправка данных	{ «message»: «Сообщение» }	–	Status200OK – Данные успешно отправлены. Status401Unauthorized – Ошибка авторизации. Status422UnprocessableEntity – При отправке данных произошла ошибка.

Сервис аутентификации на основе JWT аутентифицирует внешние системы, поддерживающие данный стандарт. Сервис предоставляет функционал для регистрации внешней системы, генерации, выдачи токена и проверки корректности его подписи. При регистрации внешняя система передает имя, идентификатор и данные открытого ключа. Для получения токена доступа к ресурсам система должна аутентифицироваться, передав свой токен, подписанный закрытым ключом. Сервис аутентификации проверяет подпись полученного токена сохраненным открытым ключом внешней системы, и, если проверки пройдены, генерирует свой JWT токен для доступа к ресурсам.

В базе данных сервиса будет содержаться аутентификационная информация внешних систем. В таблице 6 описано REST API сервиса аутентификации на основе JWT.

Таблица 6 – Описание REST API сервиса аутентификации на основе JWT

HTTP Метод	Описание	Входная структура	Выходная структура	Статусы
POST api/jwt/ systems/new	Регистрация внешней системы	{ «system_name»: «CRM-система отдела X», «system_id»: «system_hdYuk56o3», «public_key»: key.pem }	[{ "path": "api/jwt/systems/scopes", "scope": "scopes.read" }]	Status200OK – Успешная регистрация внешней системы. Status422UnprocessableEntity – При регистрации внешней системы произошла ошибка. Status423Locked – Внешняя система уже зарегистрирована.
POST api/jwt/sy stems/auth	Получение JWT токена	Authorization: Bearer <JWT> claims: { "system_id": "system_c9b1cc9c4d", "sub": "system_c9b1cc9c4d", "aud": "api/jwt/systems/auth", "exp": 3600, "scope": "scopes.read" }	JWT токен "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."	Status200OK – Токен успешно создан. Status401Unauthorized – Ошибка аутентификации. Status403Forbidden – Запрещен доступ к запрошенным ресурсам. Status422UnprocessableEntity – При получении токена произошла ошибка.
GET api/jwt/sy stems/scopes	Получение информации о разрешенных областях доступа	JWT токен "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."	[{ "path": "api/jwt/systems/scopes", "scope": "scopes.read" }]	Status200OK – Информации о разрешенных областях доступа успешно получена. Status401Unauthorized – Ошибка авторизации. Status422UnprocessableEntity – При получении информации о разрешенных областях доступа произошла ошибка.
GET api/jwt/m anagement/systems	Получение информации о зарегистрированных внешних системах	–	[{ «system_name»: «CRM-система отдела X», «created_at»: «2025 - 04- 24T10: 15:30Z», «active_token»: «eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...» }]	Status200OK – Данные клиентов успешно получены. Status401Unauthorized – Ошибка авторизации. Status422UnprocessableEntity – При получении данных клиентов произошла ошибка.
GET api/jwt/m anagement/tokens	Получение информации по всем токенам JWT	–	[{ «value»: «eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...K9x3j18Q», «creation_time»: «», «exp_time»: «», «lifetime»: 3600, «claims»: [{ «name»: «system_name», «value»: «CRM-система отдела X», }] }]	Status200OK – Данные токенов успешно получены. Status401Unauthorized – Ошибка авторизации. Status422UnprocessableEntity – При получении данных токенов произошла ошибка.

Продолжение таблицы 6

HTTP Метод	Описание	Входная структура	Выходная структура	Статусы
PUT api/jwt/management/tokens/expiration?exp_time=3600	Настройка срока действия токена	exp_time – время действия токена	–	Status200OK – Срок действия токена успешно изменен. Status401Unauthorized – Ошибка авторизации. Status422UnprocessableEntity – При изменении срока действия токена произошла ошибка.
GET api/jwt/resources/datetime	Получение даты и времени системы	JWT токен "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."	«2025-05-11T14:30:45.1234567»	Status200OK – Данные успешно получены. Status401Unauthorized – Ошибка авторизации. Status422UnprocessableEntity – При получении даты и времени произошла ошибка.
PUT api/jwt/resources/datetime?datetime=2025-05-11T14:30:45.1234567	Установка даты и времени системы	JWT токен "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..." datetime – дата и время	–	Status200OK – Данные успешно установлены. Status401Unauthorized – Ошибка авторизации. Status422UnprocessableEntity – При установке даты и времени произошла ошибка.
POST api/jwt/resources/data	Отправка данных	JWT токен "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..." { «message»: «Сообщение» }	–	Status200OK – Данные успешно отправлены. Status401Unauthorized – Ошибка авторизации. Status422UnprocessableEntity – При отправке данных произошла ошибка.

Сервис управления внешними системами агрегирует общую информацию по зарегистрированным внешним системам, включая названия систем, идентификаторы, время регистрации, поддерживаемый протокол. Позволяет блокировать/разблокировать внешнюю систему для быстрого запрета доступа к ресурсам. Также предоставляет функционал для настройки прав внешних систем. Сервис управления внешними системами реализует общий протокол взаимодействия с ИС ЭДО и запрашивает JWT токены для сервисов аутентификации. Сервисы аутентификации внешних систем на основе OAuth 2.0

и JWT используют JWT токены для авторизации в ИС ЭДО, поскольку функциональные компоненты ИС ЭДО поддерживают данный стандарт.

В базе данных сервиса управления внешними системами будут содержаться общие данные внешних систем. В таблице 7 описано REST API сервиса управления внешними системами.

Таблица 7 – Описание REST API сервиса управления внешними системами

HTTP Метод	Описание	Входная структура	Выходная структура	Статусы
GET api/management/systems	Получение общей информации о внешних системах	–	[{ «system_id»: «system-x», «system_name»: «CRM-система отдела X», «enabled»: true, «created_at»: «2025 - 04- 24T10: 15:30Z», «allowed_scopes»: [«datetime.read», «datetime.write»], «protocol»: «OAuth 2.0» }]	Status200OK – Данные внешних систем успешно получены. Status401Unauthorized – Ошибка авторизации. Status422UnprocessableEntity – При получении данных внешних систем произошла ошибка.
POST api/management/systems/{system_id}/scopes	Изменение областей доступа внешней системы	[«data.write»]	–	Status200OK – Области доступа системы успешно изменены. Status401Unauthorized – Ошибка авторизации. Status422UnprocessableEntity – При изменении областей доступа системы произошла ошибка.
PUT api/management/client/{client_id}/state?enabled=false	Блокировка внешней системы	enabled – признак активности внешней системы	–	Status200OK – Состояние внешней системы успешно изменено. Status401Unauthorized – Ошибка авторизации. Status422UnprocessableEntity – При изменении состояния внешней системы произошла ошибка.
PUT api/management/client/{client_id}/state?enabled=true	Разблокировка внешней системы	enabled – признак активности внешней системы	–	Status200OK – Состояние внешней системы успешно изменено. Status401Unauthorized – Ошибка авторизации. Status422UnprocessableEntity – При изменении состояния внешней системы произошла ошибка.

Сервис протоколирования событий предназначен для хранения наиболее важных событий аутентификации и авторизации в базе данных для последующего мониторинга и отслеживания администратором. Также каждый сервис подсистемы будет вести протоколирование в файлы. В базе данных будет храниться следующая информация о событиях: сообщение, время, тип события, идентификатор внешней системы и поддерживаемый протокол. В таблице 8 описано REST API сервиса протоколирования событий.

Таблица 8 – Описание REST API сервиса протоколирования событий

URL	HTTP Метод	Описание	Входная структура	Выходная структура	Статусы
api/ management /audit?start_i ndex=0&cou nt=50	GET	Получение записей в указанном диапазоне	start_index – начальный индекс; count – количество записей	[{ «message»: «Зарегистрирован клиент client_id1», «timestamp»: «2025 - 04 - 25T14:32: 10Z», «type»: «Registration», «client_id»: «client_id1», «level»: «Information» }]	Status200OK – Список записей событий успешно получен. Status401Unauthori zed – Ошибка авторизации. Status422Unprocess ableEntity – При получении списка записей произошла ошибка.
api/ management /audit/count	GET	Получение количества записей о событиях	–	{ «count»: 10 }	Status200OK – Количество записей успешно получено. Status401Unauthori zed – Ошибка авторизации. Status422Unprocess ableEntity – При получении количества записей произошла ошибка.

Процесс взаимодействия внешних систем с сервисом аутентификации на основе протокола OAuth 2.0 изображен на диаграмме последовательности [16] (рисунок 14). Диаграмма последовательности показывает, как объекты и

компоненты обмениваются сообщениями во времени в рамках конкретного сценария. На рисунке 14 изображены процессы регистрации внешней системы, получения аутентификационных данных, запроса токена доступа и обращения к ресурсам ИС ЭДО в рамках протокола OAuth 2.0 Client Credentials.

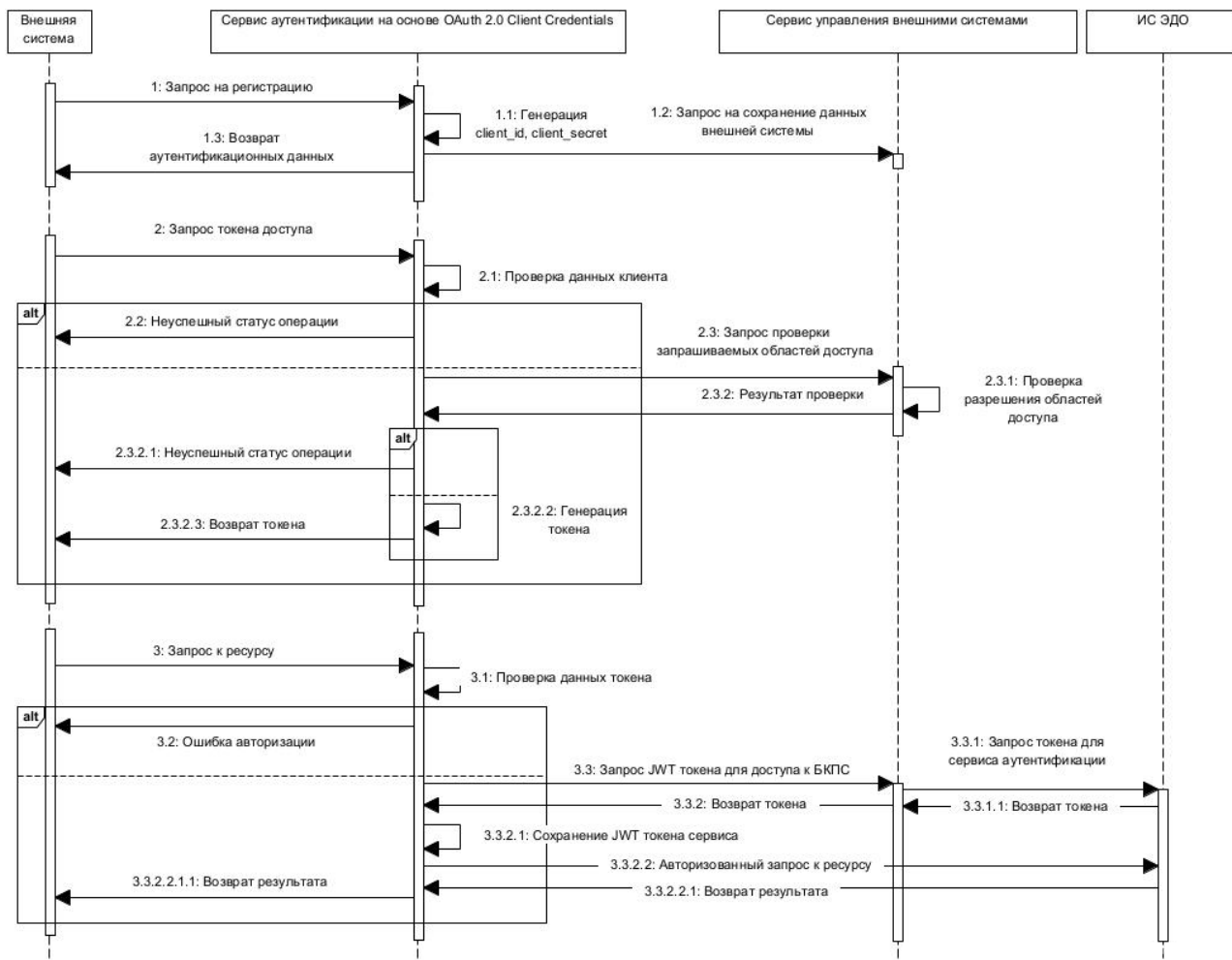


Рисунок 14 – Диаграмма последовательности

Для представления структуры сервиса с точки зрения классов, атрибутов и взаимосвязей между ними была построена диаграмма классов [16] сервиса аутентификации на основе протокола OAuth 2.0 Client Credentials (рисунок 15).

Данная диаграмма отражает основные сущности, участвующие в работе сервиса, включая контроллеры, сервисы, хранилища и интерфейсы для взаимодействия с другими компонентами. Взаимодействие между классами построено по принципу инверсии зависимостей и внедрения интерфейсов, что является преимуществом ASP.NET.

В таблице 9 приведено описание основных классов и их назначение.

Класс	Назначение
ClientsController	Контроллер, предоставляющий функционал регистрации и получения прав доступа для внешних систем.
ClientsManagementController	Контроллер, предоставляющий функционал веб-приложению администрирования для получения данных систем.
TokensManagementController	Контроллер, предоставляющий функционал веб-приложению администрирования для получения данных токенов и настройки параметров токенов.
ResourcesController	Контроллер, предоставляющий доступ к ресурсам ИС ЭДО авторизованным внешним системам.
ServiceTokenRefresher	Фоновый сервис, отслеживающий обновление JWT токена для авторизации сервиса в ИС ЭДО.

Продолжение таблицы 10

Класс	Назначение
SystemsManagementController	Контроллер, предоставляющий функционал веб-приложению администрирования для получения данных систем.
TokensManagementController	Контроллер, предоставляющий функционал веб-приложению администрирования для получения данных токенов JWT и настройки параметров токенов.
ResourcesController	Контроллер, предоставляющий доступ к ресурсам ИС ЭДО авторизованным внешним системам.
ServiceTokenRefresher	Фоновый сервис, отслеживающий обновление JWT токена для авторизации сервиса в ИС ЭДО.
SystemsManagementClient	Клиент для взаимодействия с сервисом управления внешними системами.
AuditServiceClient	Клиент для взаимодействия с сервисом протоколирования событий.
ServiceStorage	Класс для доступа к хранилищу данных.
OAuthThriftServer	Трифт-сервер сервиса, обрабатывающий запросы от ClientsManagement.Service на изменение состояния и разрешенных областей доступа внешних систем.

Диаграмма классов сервиса протоколирования событий приведена на рисунке 17.

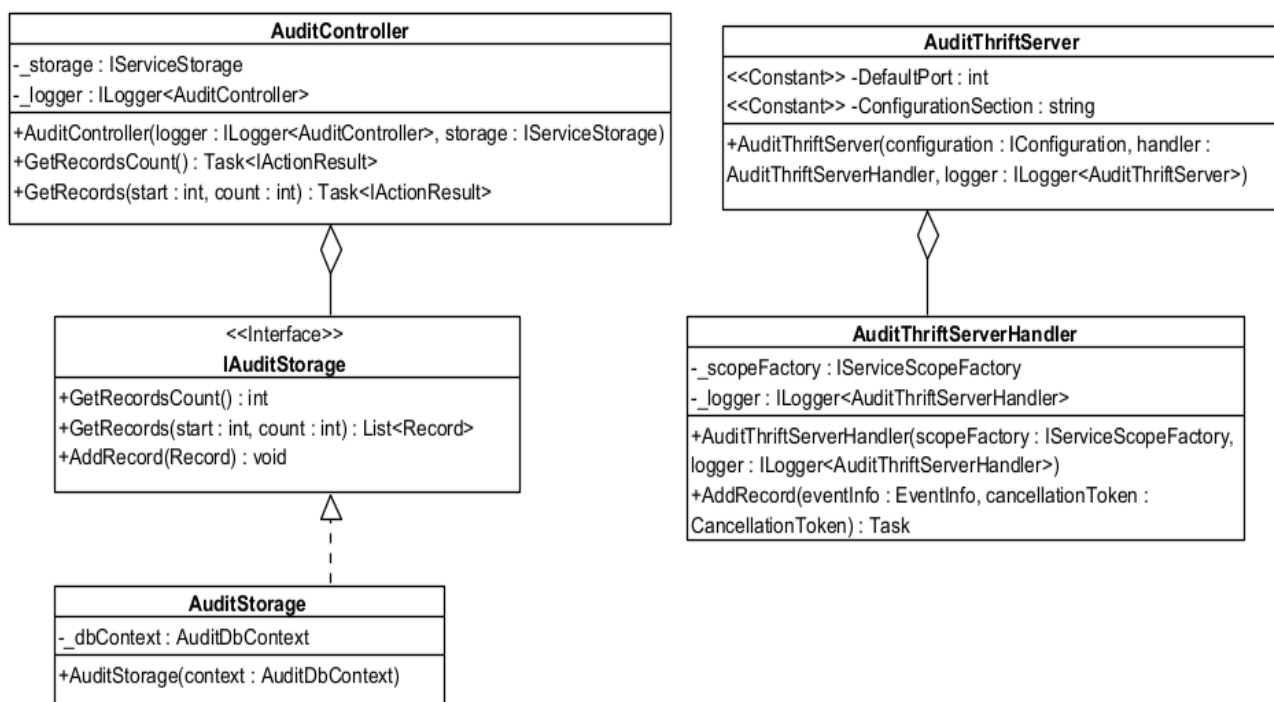


Рисунок 17 – Диаграмма классов сервиса Audit.Service

В таблице 11 приведено описание основных классов сервиса протоколирования событий и их назначение.

Таблица 11 – Описание классов сервиса Audit.Service

Класс	Назначение
AuditController	Контроллер, предоставляющий функционал веб-приложению администрирования для получения информации и событиях.
AuditStorage	Класс для работы с хранилищем данных.
AuditThriftServer	Трифт-сервер сервиса, обрабатывающий запросы на сохранение основных событий, связанных с действиями внешних систем, в хранилище данных.

Диаграмма классов сервиса управления внешними системами приведена на рисунке 18.

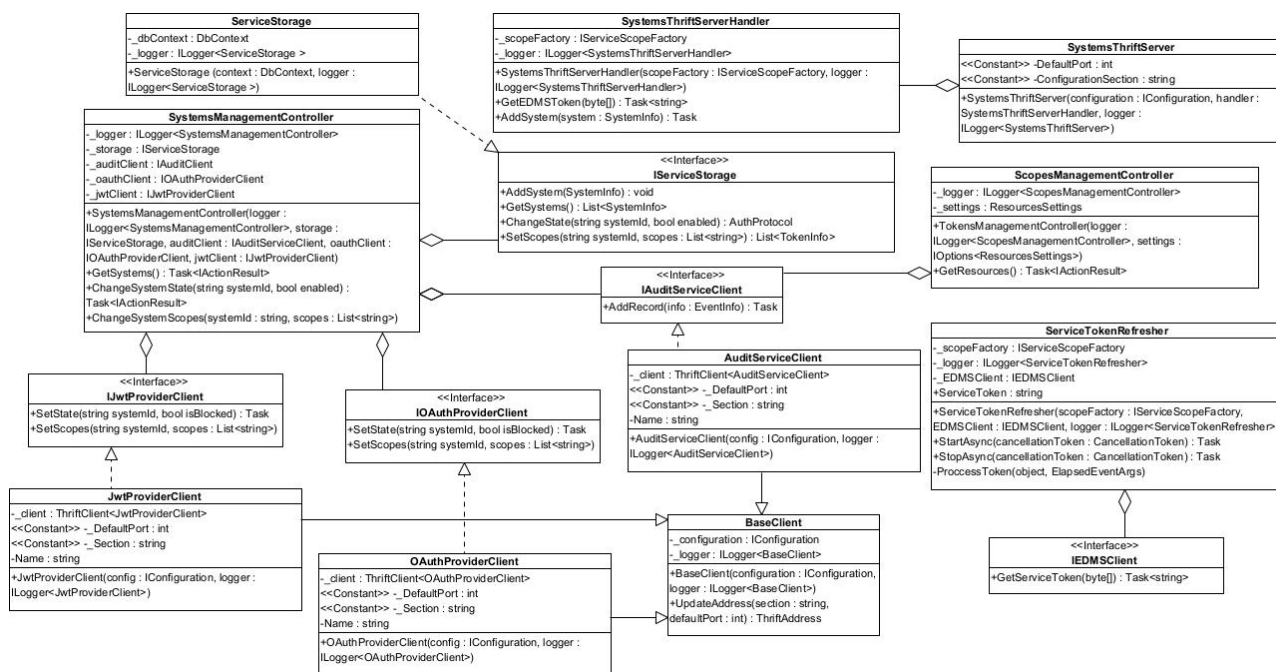


Рисунок 18 – Диаграмма классов сервиса SystemsManagement.Service

В таблице 12 приведено описание основных классов сервиса управления внешними системами и их назначение.

Таблица 12 – Описание классов сервиса SystemsManagement.Service

Класс	Назначение
SystemsManagementController	Контроллер, предоставляющий функционал веб-приложению администрирования для получения данных систем и управления системами.

Продолжение таблицы 12

Класс	Назначение
ScopesManagementController	Контроллер, предоставляющий функционал веб-приложению администрирования для получения информации о ресурсах, предоставляемых системам.
ServiceTokenRefresher	Фоновый сервис, отслеживающий обновление JWT токена для авторизации сервиса в ИС ЭДО.
AuditServiceClient	Клиент для взаимодействия с сервисом протоколирования событий.
ServiceStorage	Класс для доступа к хранилищу данных.
SystemsThriftServer	Трифт-сервер сервиса, обрабатывающий запросы от сервисов аутентификации на добавление внешней системы и получение токенов авторизации.
SystemsThriftServerHandler	Обработчик запросов, поступающих на SystemsThriftServer
JwtProviderClient	Клиент для взаимодействия с сервисом аутентификации на основе JWT для настройки состояния систем и прав доступа.
OAuthProviderClient	Клиент для взаимодействия с сервисом аутентификации на основе OAuth 2.0 для настройки состояния систем и прав доступа.

На основе проведённого проектирования и представленных взаимодействий будет осуществляться реализация компонентов подсистемы обеспечения механизма единой аутентификации.

4 Реализация программных средств

4.1 Реализация серверной части программных средств

На основе разработанной архитектуры механизма единой аутентификации была реализована серверная часть приложения. Код основного функционала приведен в приложении Б.

Разработка серверной части подсистемы единой аутентификации осуществлялась на языке программирования C# и платформы .NET 8.0 [18]. Каждый сервис представляет собой веб-приложение на основе ASP.NET Core. ASP.NET Core – это кроссплатформенный фреймворк для создания веб-приложений и REST API, разработанный Microsoft [19]. Он обеспечивает высокую производительность, встроенную поддержку маршрутизации, аутентификации, авторизации и работу с HTTP-запросами. Благодаря middleware-архитектуре и системе внедрения зависимостей ASP.NET Core позволяет быстро разрабатывать и гибко настраивать серверные приложения.

Для хранения данных сервисы используют встроенные реляционные базы данных SQLite. Для работы с базой данных используется NuGet-пакет Microsoft.EntityFrameworkCore.Sqlite. База данных SQLite компактна и проста в конфигурировании, представляет собой файл с расширением .db, что удобно на этапах разработки, поскольку нет необходимости в развертывании отдельного сервера.

При реализации аутентификации внешних систем на основе протокола OAuth 2.0 Client Credentials Flow в сервисе OAuthProvider.Service использовался NuGet-пакет Duende.IdentityServer [20]. Данный инструмент предоставляет функционал для организации централизованной аутентификации и авторизации в распределённых системах. Готовая реализация стандартов NuGet-пакета была расширена и изменена в соответствии с требованиями к системе и разработанной архитектурой. Была реализована собственная логика регистрации внешних систем, настройки областей доступа, выпуска и проверки токенов. Данные систем, их разрешенные области доступа, конфигурационные данные

сохраняются и настраиваются в базе данных с использованием NuGet-пакета Duende.IdentityServer.EntityFramework. Он предоставляет сущности протокола OAuth 2.0 с готовым набором атрибутов.

Для протоколирования событий в файлы использовался NuGet-пакет NLog.Web.AspNetCore. Также основная информация о действиях внешних систем сохраняется в базу данных сервиса протоколирования событий.

Для внутреннего межсервисного взаимодействия использовался фреймворк Apache Thrift. Он позволяет описывать интерфейсы взаимодействий и структуры в формате, независимом от языков программирования, а затем генерировать код для нужного языка. Благодаря бинарному формату сериализации данных передача происходит достаточно быстро. Thrift-интерфейсы сервисов указаны в приложении Б.

Все перечисленные технологии, используемые при разработке, приведены в таблице 13.

Таблица 13 – Описание используемых технологий серверной части

Компонент	Описание
Microsoft.AspNetCore.Authentication.JwtBearer	Предоставляет функционал для работы с JWT-токенами, используется при аутентификации и авторизации, создании и проверке токенов.
Microsoft.EntityFrameworkCore.Sqlite	Предоставляет функционал для создания, управления базой данных SQLite.
NLog.Web.AspNetCore	Предоставляет функционал для протоколирования событий.
ApacheThrift	Предоставляет функционал для межсервисного взаимодействия и передачи данных.
Duende.IdentityServer	Предоставляет функционал для реализации протокола OAuth 2.0 Client Credentials Flow.
Duende.IdentityServer.EntityFramework	Предоставляет функционал для взаимодействия с базой данных в рамках реализации OAuth 2.0 Client Credentials Flow.

Для предоставления метаданных сервиса аутентификации OAuthProvider.Service используется конечная точка /.well-known/openid-configuration, являющийся стандартом спецификации OpenID Connect [9].

Полученные данные позволят системам узнать, как правильно взаимодействовать с сервером авторизации. Пример содержимого конечной точки `/.well-known/openid-configuration` в рамках разработанного сервиса приведен на рисунке 19.

```

1  {
2    "registration_specification_uri": "register/openapi.json",
3    "client_scopes_specification_url": "allowed-scopes/openapi.json",
4    "token_endpoint": "connect/token",
5    "scopes_allowed": [
6      "scopes.read",
7      "datetime.read",
8      "datetime.write",
9      "data.write"
10   ],
11   "resource_api_specification_url": "protected-resources/openapi.json"
12 }

```

Рисунок 19 – Содержимое конечной точки `/.well-known/openid-configuration`

Для моделирования переходов потока управления в рамках процесса получения доступа к ресурсам ИС ЭДО внешней системы, поддерживающей протокол аутентификации на основе JWT, построена диаграмма деятельности [16] (рисунок 20). Диаграмма деятельности отражает алгоритмы выполнения процесса и последовательность действий, выполняемых системой. Она позволяет визуальнo проследить поток управления, включая условия переходов, ветвления и параллельные действия, обеспечивая наглядное понимание логики получения доступа к запрашиваемым ресурсам.

В рассматриваемом процессе участвуют: сервис аутентификации на основе JWT, сервис управления внешними системами и функциональные компоненты ИС ЭДО. Сначала сервис `JWTProvider.Service` проверяет полученный JWT токен внешней системы – проверяются издатель, аудитория, срок действия, подпись и области доступа токена. Затем проверяется состояние блокировки внешней системы, если система заблокирована дальнейшие проверки прерываются и возвращается неуспешный статус операции. Далее сервис `JWTProvider.Service` проверяет срок действия своего токена авторизации в ИС ЭДО, если он истек, запрашивает новый у сервиса управления системами. Получив действующий

токен, отправляет авторизованный запрос в ИС ЭДО на получение запрашиваемых внешней системой ресурсов.

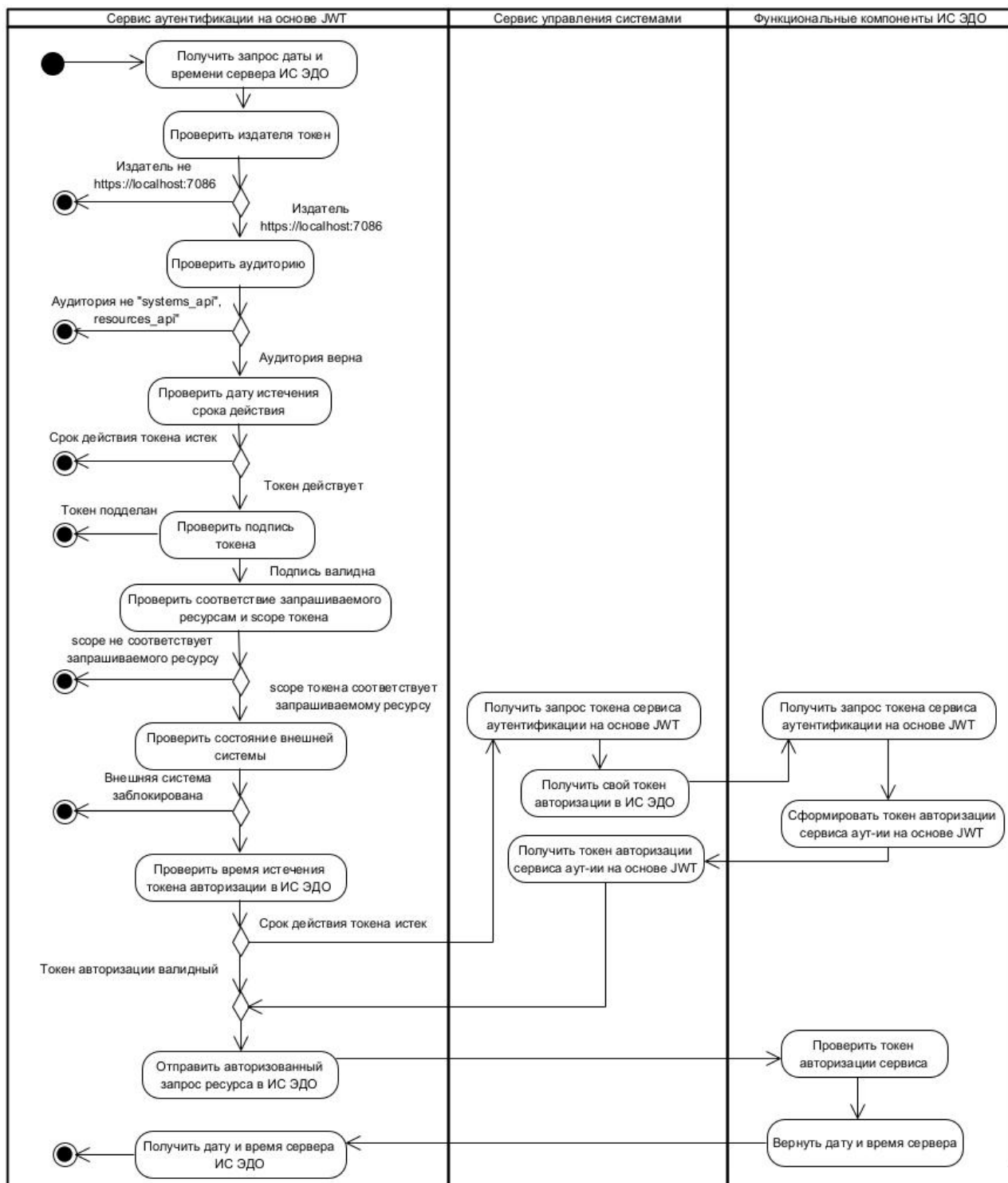


Рисунок 20 – Диаграмма деятельности процесса получения внешней системой, поддерживающей аутентификацию с помощью JWT, даты и времени сервера

В процессе реализации серверной части подсистемы был разработан механизм единой аутентификации внешних систем, обеспечивающий доступ к

ресурсам авторизованным системам, поддерживающим протокол OAuth 2.0 Client Credential Grant и стандарт JWT. Реализация данного механизма позволила стандартизировать подход к проверке подлинности внешних систем и упростить взаимодействие с компонентами ИС ЭДО.

4.2 Реализация клиентской части программных средств

Клиентская часть подсистемы единой аутентификации представляет собой веб-приложение для администрирования внешних систем, запрашивающих доступ к ресурсам защищенной системы. Исходный код интерфейса администрирования приведен в приложении В.

Для реализации веб-интерфейса административной панели была выбрана архитектура Single Page Application (SPA), которая позволяет обеспечить быстрый отклик системы и удобную навигацию между разделами без перезагрузки страницы. При разработке интерфейса использовался React [21] – JavaScript-библиотека для построения пользовательских интерфейсов, разработанная и поддерживаемая компанией Meta. Основными особенностями React является управление виртуальным DOM, компонентный подход к разработке интерфейсов, управление состояниями компонентов с помощью хуков (hooks).

Для создания отзывчивого пользовательского интерфейса была использована библиотека Material-UI (MUI), основанная на CSS-in-JS подходе. Она предоставляет готовые, стилизованные UI-элементы, которые можно использовать для быстрого проектирования интерфейсов. Подход CSS-in-JS позволяет динамически менять стили компонента в зависимости от его состояния, применять единые темы, использовать готовые хуки.

Маршрутизация в приложении реализована с помощью библиотеки React Router [21], позволяющей создавать многостраничные интерфейсы в одностраничных приложениях без перезагрузки страницы. В зависимости от пути в адресной строке загружается тот или иной компонент.

Взаимодействие с серверной частью реализовано с помощью библиотеки Axios, которая предоставляет удобные методы для отправки GET, POST, PUT, DELETE и других типов запросов, и встроенную обработку ошибок. Также реализовано автоматическое обновление JWT-токенов по истечении срока действия и их хранение в localStorage – встроенный в браузер механизм для хранения данных на стороне клиента.

Приведем описание компонентов веб-интерфейса. Основным компонентом, определяющим структуру всего приложения, является Layout.js. Он отвечает за общую компоновку интерфейса, включая верхнюю панель и боковое меню навигации. Главная страница (Home.jsx) (рисунок 21) содержит функционал для общей настройки внешних систем: блокировка/разблокировка, установка разрешенных областей доступа. На этой странице отображаются данные ресурсов, предоставляемых системам, информация о внешних системах и записи журнала событий.

The screenshot displays the Admin UI interface. On the left is a dark sidebar with the 'Admin UI' header and navigation links: 'Главная' (selected), 'JWT', and 'OAuth'. The main content area is divided into three sections:

- API ресурсы (API Resources):** Contains two panels. The 'Systems API' panel shows a 'Scope' of 'scopes.read' and a 'Path' of 'api/jwt/systems/{systemid}/scopes'. The 'Resources API' panel shows three entries: 'datetime.read' (Path: 'api/jwt/resources/datetime'), 'datetime.write' (Path: 'api/jwt/resources/datetime'), and 'data.write' (Path: 'api/jwt/resources/data').
- Системы (Systems):** A table listing systems. One system is shown: 'system11' with ID 'system_1lhgce6788', status 'on', and access scopes 'scopes.read' and 'datetime.read'. It uses the 'JWT' protocol and was created on '13.06.2025, 14:56:26'.
- Журнал событий (Event Log):** A table with a search bar and columns for 'ID системы', 'Сообщение', 'Тип', 'Уровень', 'Протокол', and 'Время'. It contains five entries related to the system 'system_1lhgce6788', including successful unlocking, access scope changes, and successful locking.

Рисунок 21 – Представление компонента Home.jsx

Форма входа (AuthDialog.jsx) (рисунок 22) предназначена для аутентификации, где администратор вводит отпечаток сертификата. В рамках данной выпускной квалификационной работы реализован упрощенный демонстрационный вариант аутентификации администратора в системе, поскольку основная цель работы в реализации подсистемы обеспечения механизма единой аутентификации внешних систем. Форма включает поле ввода отпечатка сертификата и кнопку входа. При успешной аутентификации загружается главная страница приложения.

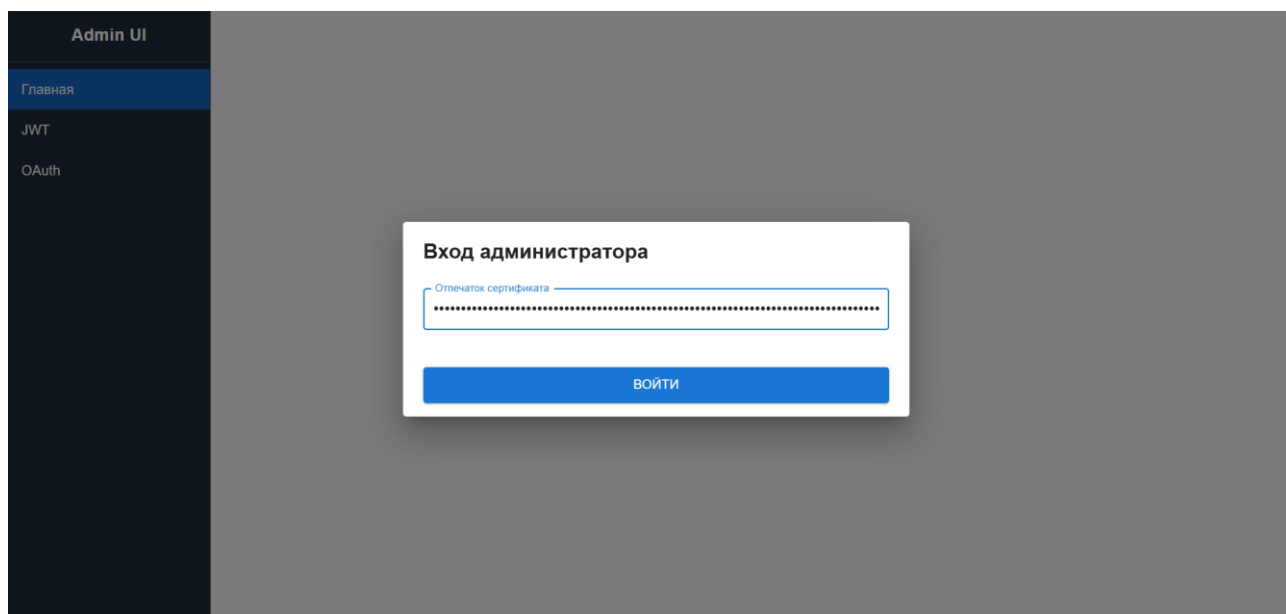


Рисунок 22 – Представление компонента «AuthDialog.js»

На вкладках «JWT» (JwtSettings.jsx) и «OAuth» (OAuthSettings.jsx) представлена информация о внешних системах, поддерживающих аутентификацию с помощью данных протоколов (рисунки 23-24). На страницах представлены данные внешних систем, информация по выданным токенам для доступа к ресурсам ИС ЭДО и записи журнала событий. Также предоставляется возможность смены срока действия JWT токена и выполнения операций с таблицами для более удобного представления информации.

В рамках работы клиентского приложения реализован механизм централизованного хранения и отслеживания токенов доступа к backend-сервисам. После запуска приложения и при отсутствии действующих токенов

администратору отображается форма авторизации. Через неё он вводит отпечаток сертификата (X-Certificate-Fingerprint), на основании которого производится аутентификация. Успешные ответы от сервисов содержат набор токенов, которые сохраняются в состоянии приложения так, что любой компонент может получить к ним доступ. При отправке запросов соответствующий токен (например, systemsToken) добавляется в заголовок авторизации.

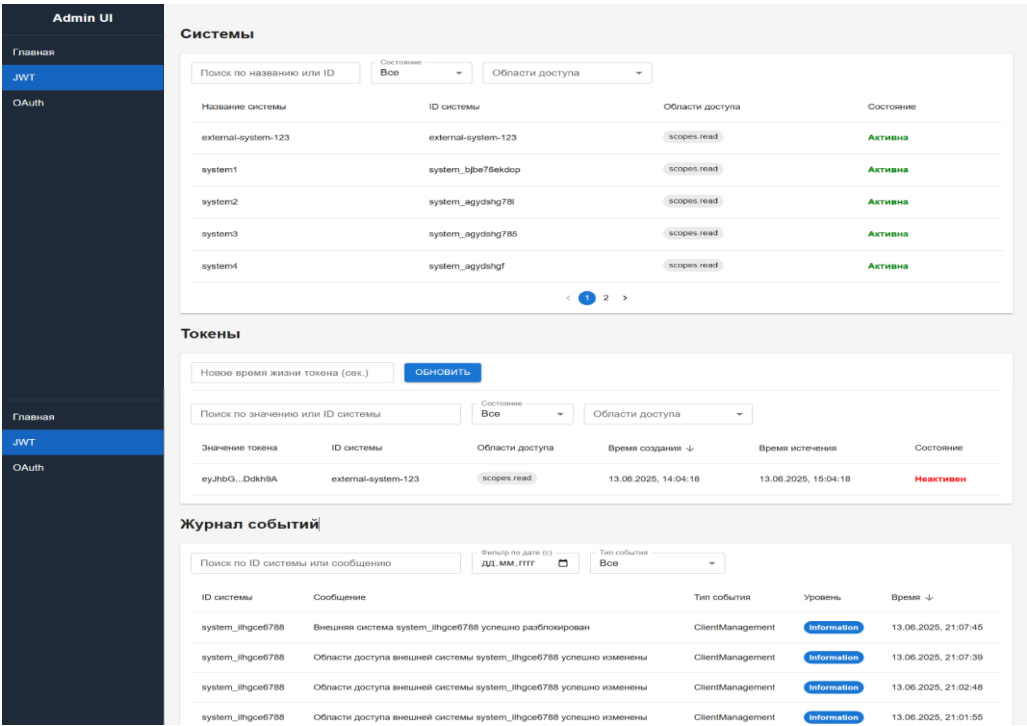


Рисунок 23 – Представление компонента «JwtSettings.js»

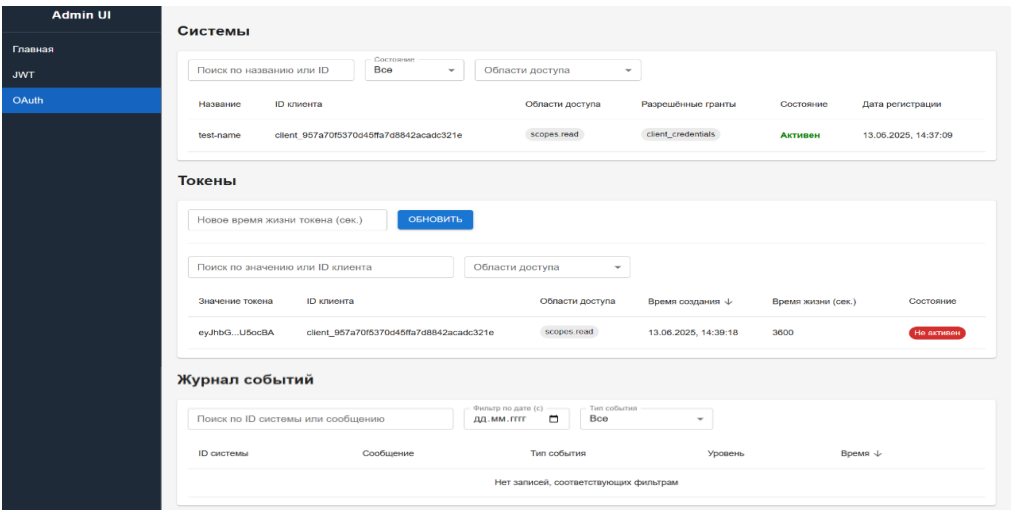


Рисунок 24 – Представление компонента «OAuthSettings.js»

Каждый токен имеет ограниченное время жизни, поэтому для обеспечения безопасности реализован таймер, отсчитывающий срок действия токенов. По его истечении токены автоматически удаляются из состояния, и приложение возвращает администратора к форме авторизации.

В результате была реализована клиентская часть административного интерфейса в виде одностраничного веб-приложения, обеспечивающая удобное и безопасное управление внешними системами, их областями доступа, токенами и событиями аудита. Использование современных веб-технологий React, React Router и Material-UI [21] позволило создать отзывчивое и визуально целостное приложение, соответствующее требованиям по производительности и удобству использования.

4.3 Развертывание компонентов

В ходе реализации серверной части механизма единой аутентификации было разработано четыре сервиса:

- `JWTProvider.Service` – сервис для аутентификации внешних систем, поддерживающих стандарт JWT;
- `OAuthProvider.Service` – сервис для аутентификации внешних систем, поддерживающих протокол аутентификации OAuth 2.0 Client Credentials Grant;
- `SystemsManagement.Service` – сервис, управляющий настройками зарегистрированных внешних систем, реализует интерфейс взаимодействия с компонентами ИС ЭДО по части аутентификации сервисов подсистемы единой аутентификации;
- `Audit.Service` – сервис, отвечающий за сбор и хранение наиболее важных событий, связанных с деятельностью внешних систем.

Каждый сервис представляет собой веб-приложение, которое может быть запущено как самостоятельный процесс на сервере.

Клиентская часть Admin UI предоставляет интерфейс для администрирования, также запускается как отдельный процесс на АРМ

администратора. Взаимодействует с серверной частью по защищенному каналу передачи данных в соответствии с определенным REST API через веб-браузер.

Для хранения данных используются локальные файловые базы данных SQLite, располагающиеся на сервере.

Для отображения физического размещения компонентов системы на узлах и связей между ними построена диаграмма развертывания [16] (рисунок 25).

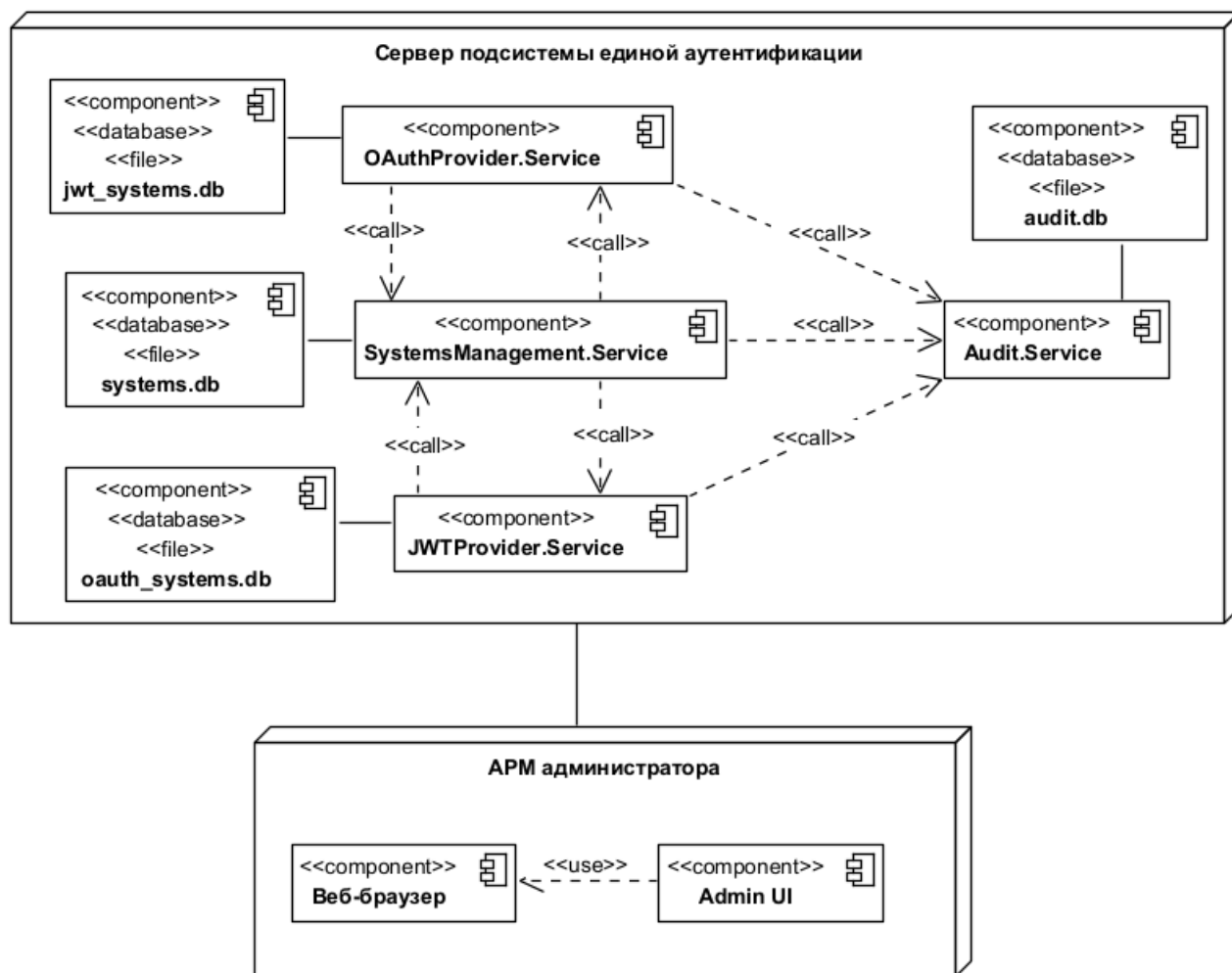


Рисунок 25 – Диаграмма развертывания подсистемы единой аутентификации

Диаграмма развертывания позволяет наглядно представить физическое распределение компонентов системы по узлам инфраструктуры. Она отображает связи между компонентами, протоколы взаимодействия, а также показывает, какие части системы размещаются на стороне клиента, а какие – на стороне сервера.

5 Контроль качества программных средств

5.1 Функциональное тестирование серверной части приложения

Функциональное тестирование программных средств предназначено для проверки работоспособности функций приложения, выполнения всех функциональных требований к системе. Основная цель функционального тестирования – убедиться, что программный продукт работает так, как ожидается, без ошибок в логике и взаимодействии с пользователем [22]. В отличие от нефункционального тестирования, которое проверяет параметры качества системы (производительность, стабильность, безопасность и другие), функциональное тестирование фокусируется на действиях системы и ожидаемом результате.

Приведем основные виды функционального тестирования:

- модульное тестирование – проверка отдельных компонентов системы изолированно друг от друга;
- интеграционное тестирование – проверка корректного взаимодействия между различными компонентами;
- системное тестирование – проверка работоспособности всей системы в целом;
- приемочное тестирование – финальная проверка работоспособности системы перед итоговым релизом, часто выполняется с участием заказчика, чтобы определить соответствует ли разработанный функционал его требованиям;
- регрессионное тестирование – проверка того, что новые внесенные изменения не сломали существующий функционал.

Существует три основных метода функционального тестирования: метод «черного ящика», метод «белого ящика» и метод «серого ящика» [23]. Рассмотрим каждый из них подробнее.

Метод «чёрного ящика» предполагает, что тестировщик рассматривает тестируемую систему как «чёрный ящик», не зная, как она устроена внутри. Он

не видит исходный код, алгоритмы и структуру приложения, а ориентируется только на входы и ожидаемые выходы. Основная цель данного метода – найти ошибки в функциональности, логике, валидации данных, интерфейсе и других внешне проявляемых аспектах. Отсутствие доступа к коду позволяет тестировщику взглянуть на систему глазами пользователя, не ограничиваясь ожиданиями разработчика – это помогает выявлять нестандартные сценарии и ошибки. Основными техниками данного метода тестирования являются: эквивалентное разбиение, анализ граничных значений, таблицы принятия решений, тестирование на основе состояний.

Тестирование по методу «белого ящика» основывается на внутренней структуре приложения. Основное внимание уделяется проверке логики, ветвлений, циклов и условий внутри программы. Тестирование методом «белого ящика» обеспечивает максимальное покрытие кода тестами, нахождение логических ошибок, которые не видны при тестировании методом «черного ящика».

При тестировании методом «серого ящика» тестировщик имеет частичное представление о внутреннем устройстве системы, но при этом ориентируется на поведение программы снаружи, как конечный пользователь. Это гибридный подход к тестированию, сочетающий в себе элементы методов «чёрного ящика» и «белого ящика». Целью метода «серого ящика» является проверка системы с разных сторон, при этом не углубляясь в полную реализацию кода.

В данной выпускной квалификационной работе проведено функциональное тестирование методом «черного ящика» процессов получения внешней системой, поддерживающей протокол аутентификации OAuth 2.0 Client Credentials Grant, токена доступа и получения доступа к защищенным ресурсам.

5.1.1 Классы эквивалентности

Классы эквивалентности – это техника тестирования черного ящика, при которой входные данные разбиваются на группы (классы), где все значения внутри одного класса считаются эквивалентными с точки зрения поведения

программы. Это позволяет сократить количество тестовых случаев, проверяя только по одному представителю из каждого класса [24].

Для получения токена доступа (access token) внешняя система должна отправить запрос токена на конечную точку /connect/token. В таблице 14 приведена корректная структура запроса получения токена доступа.

Таблица 14 – Корректная структура запроса «/connect/token»

Название параметра	Пример значения	Описание
client_id	client_df63a612a3c24c12a87a147de5825721	Идентификатор клиента
client_secret	qk6vEwxHdcOFCnobyGNLPhgRpAtBjbzsXtuTKyWgi1cmBXYiHD/IMTWsP7FVrqiKk/f/OqZ7lLdT/FnP/Xb8rg==	Секрет клиента
grant_type	client_credentials	Тип гранта
scope	scopes.read	Запрашиваемые области доступа

Тело запроса токена доступа – это входная структура, следовательно, определим классы эквивалентности для входных параметров. В таблице 15 приведены правильные и неправильные классы эквивалентности [24] параметров запроса.

Таблица 15 – Правильные и неправильные классы эквивалентности (КЭ)

	Правильные КЭ	Неправильные КЭ
grant_type	«client_credentials» (1)	Остальное (2)
client_id	Зарегистрированный client_id (3)	Незарегистрированный/пустой (4)
client_secret	client_secret, соответствующий client_id (5)	Неправильный/пустой (6)
scope	Области доступа, разрешенные клиенту client_id (7)	Остальное (8)

На основе полученных классов эквивалентности составим тест-кейсы (таблица 16). Результаты тестирования приведены в приложении Г. На рисунке Г.1 изображен результат успешной регистрации внешней системы и допустимые значения для запроса токена доступа. Функциональное тестирование осуществлено с помощью инструмента Postman.

Таблица 16 – Тест-кейсы по классам эквивалентности

№	КЭ	Входные данные	Выходные данные	Результат прохождения теста
1	1, 3, 5, 7	grant_type=client_credentials client_id=client_df63a612a3c24c12a87a147de5825721 client_secret=qk6vEwxHdcOFCnobyGNLPhgRpAtBjbzsXtuTKyWgi1cmBXYiHD/IMTWsP7FVrqiKk/f/OqZ7lLdT/FnP/Xb8rg== scope=scopes.read	{ «access_token»: «eyJhbGciOiJSUzI1NiIsImtpZCI6IjVvZzFzXz1ld2t5R2VrVVliOUtpdXRTc0UtT3hNQzds...», «expires_in»: 3600, «token_type»: «Bearer», «scope»: «scopes.read» }	Рисунок Г.2
2	2, 3, 5, 7	grant_type=authorization_code client_id=client_df63a612a3c24c12a87a147de5825721 client_secret=qk6vEwxHdcOFCnobyGNLPhgRpAtBjbzsXtuTKyWgi1cmBXYiHD/IMTWsP7FVrqiKk/f/OqZ7lLdT/FnP/Xb8rg== scope=scopes.read	{ «error»: «unauthorized_client» }	Рисунок Г.3
3	1, 4, 5, 7	grant_type=client_credentials client_id= « » client_secret=qk6vEwxHdcOFCnobyGNLPhgRpAtBjbzsXtuTKyWgi1cmBXYiHD/IMTWsP7FVrqiKk/f/OqZ7lLdT/FnP/Xb8rg== scope=scopes.read	{ «error»: «invalid_request» }	Рисунок Г.4
4	1, 3, 6, 7	grant_type=client_credentials client_id= client_df63a612a3c24c12a87a147de5825721 client_secret=1qk6vEwxHdcOFCnobyGNLPhgRpAtBjbzsXtuTKyWgi1cmBXYiHD/IMTWsP7FVrqiKk/f/OqZ7lLdT/FnP/Xb8rg== scope=scopes.read	{ «error»: «invalid_client» }	Рисунок Г.5
5	1, 3, 5, 8	grant_type=client_credentials client_id= client_df63a612a3c24c12a87a147de5825721 client_secret=qk6vEwxHdcOFCnobyGNLPhgRpAtBjbzsXtuTKyWgi1cmBXYiHD/IMTWsP7FVrqiKk/f/OqZ7lLdT/FnP/Xb8rg== scope=data.read	{ «error»: «invalid_scope» }	Рисунок Г.6

Проведённые тесты подтвердили корректность обработки запроса на получение токена доступа. Все сценарии, включая валидные и невалидные данные, отработали в соответствии с ожидаемыми результатами.

5.1.2 Диаграмма состояний и переходов

Рассмотрим тестирование методом «черного ящика» на основе диаграммы состояний и переходов [24]. На рисунке 26 изображена диаграмма состояний [16] для реактивного объекта «Токен доступа». Диаграмма состояний отражает, как объект изменяет свои состояния на протяжении жизненного цикла, реагируя на различные события.

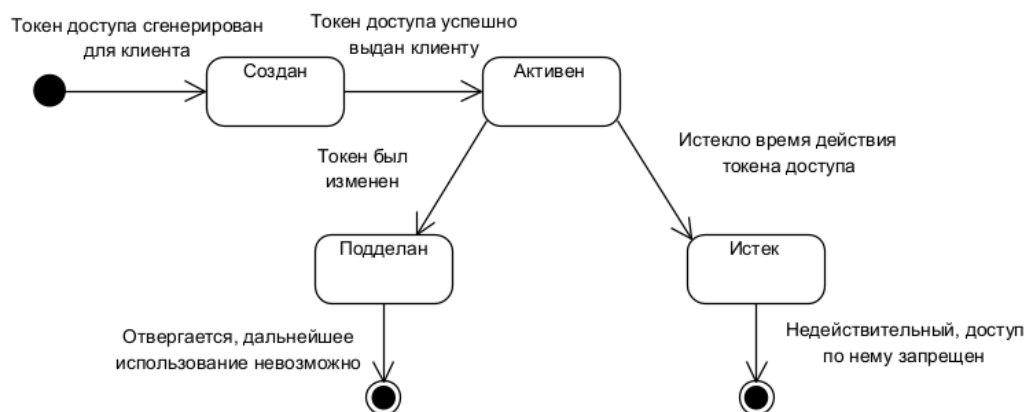


Рисунок 26 – Диаграмма состояний токена доступа

Тест-кейсы разрабатываются на основе каждого состояния и перехода в другое состояние, также учитываются граничные условия. Для получения доступа к ресурсам ИС ЭДО внешняя система, поддерживающая OAuth 2.0, должна предоставить валидный JWT-токен доступа, выданный разработанным механизмом единой аутентификации. В зависимости от сценария токен может находиться в одном из состояний, изображенных на рисунке 26.

Составим тест-кейсы (таблица 17) для проверки получения внешней системой доступа к методу просмотра списка разрешенных scopes на основе диаграммы состояний. Перед этим добавим в список разрешенных областей доступа зарегистрированной ранее системы scopes «datetime.read» и «datetime.write».

Таблица 17 – Тест-кейсы по диаграмме состояний и переходов

№	Переход/Граничное условие	Входные данные	Выходные данные	Результат прохождения теста
1	Создан → Активен	Валидный токен: eyJhbGciOiJSUzI1NiIsIm... GET https://localhost:7080/unified-auth/api/clients/client_df63a612a3c24c12a87a147de5825721/scopes	[«datetime.read», «datetime.write», «scopes.read»]	Рисунок Г.7
2	Активен → Изменен	Изменен payload токена: 1eyJhbGciOiJSUzI1NiIsIm... GET https://localhost:7080/unified-auth/api/clients/client_df63a612a3c24c12a87a147de5825721/scopes	401 Unathorized, соответствующее сообщение в файле с логами	Рисунок Г.8
3	Активен → Истек	Дождаться истечения срока действия токена: eyJhbGciOiJSUzI1NiIsIm... GET https://localhost:7080/unified-auth/api/clients/client_df63a612a3c24c12a87a147de5825721/scopes	401 Unathorized, соответствующее сообщение в файле с логами	Рисунок Г.9
4	Scope токена не соответствует scope метода	Валидный токен с другим scope: eyJhbGciOiJSUzI1NiIsImtpZCI6IjVvZzFzXz1ld2t5R2VrVVliOUtpdXRTc0UtT... GET https://localhost:7080/unified-auth/api/clients/client_df63a612a3c24c12a87a147de5825721/scopes	401 Unathorized, соответствующее сообщение в файле с логами	Рисунок Г.10
5	Пустой токен	Отсутствует токен в заголовке аутентификации запроса: GET https://localhost:7080/unified-auth/api/clients/client_df63a612a3c24c12a87a147de5825721/scopes	401 Unathorized, соответствующее сообщение в файле с логами	Рисунок Г.11

Тестирование на основе диаграммы состояний и переходов показало работоспособность механизма проверки токенов доступа. Дополнительно были проверены пограничные и недопустимые состояния, что подтвердило устойчивость системы к ошибочным входным данным. Результаты тестирования подтвердили соответствие поведения системы заданной спецификации, представленной в виде диаграммы состояний.

5.2 Функциональное тестирование интерфейса администрирования

Тестирование Web API – это проверка взаимодействия клиентской части с серверной частью через HTTP-запросы к API. Основными целями данного вида тестирования являются проверка корректной отправки запросов и обработки ответов, проверка визуальной и интерактивной части приложения, проверка поведения системы при негативных сценариях поведения пользователя [25].

Интерфейс администрирования состоит из трех основных вкладок: «Главная», «JWT» и «OAuth». Для каждой вкладки составим свои тест-кейсы, покрывающие ключевые пользовательские сценарии, включая как проверки API, так и корректность работы и отображения пользовательского интерфейса.

В таблице 18 приведены тест-кейсы проверки входа администратора в систему. После корректного ввода отпечатка сертификата должна быть отображена главная страница.

Таблица 18 – Тест-кейсы для формы авторизации в системе

Название теста	Шаги выполнения	Ожидаемый результат	Наблюдаемый результат
Успешная аутентификация администратора	Ввести корректный отпечаток сертификата, заранее сохраненный на сервере	Успешная аутентификация, переход на главную страницу панели администрирования	Рисунок Г.12
Ошибка при аутентификации администратора	Ввести невалидный отпечаток сертификата	Ошибка аутентификации	Рисунок Г.13

В таблице 19 приведены тест-кейсы проверки главной страницы управления внешними системами. Тест-кейсы покрывают сценарии сортировки и фильтрации данных, блокировку и разблокировку систем, изменение разрешенных областей доступа систем.

Таблица 19 – Тест-кейсы главной страницы

Название теста	Шаги выполнения	Ожидаемый результат	Наблюдаемый результат
Корректное отображение всех данных	Перейти на страницу «Главная»	Данные систем, ресурсы и журнал событий успешно отображены	Рисунок Г.14
Заблокировать внешнюю систему	Заблокировать внешнюю системы с названием jwt-system1	Появилась запись в журнале событий, на вкладке «JWT» данная система заблокирована, запрещено получение токенов доступа.	Рисунок Г.15
Разблокировать внешнюю систему	Разблокировать внешнюю системы с названием jwt-system1	Появилась запись в журнале событий, на вкладке «JWT» данная система активна, доступно получение токенов для системы.	Рисунок Г.16
Установка областей доступа системе	Добавить системе с названием oauth-system3 разрешенные scopes «datetime.read», «data.write»	Появилась запись в журнале событий, на вкладке «OAuth» данные системы изменились, доступно получение токена с соответствующими scopes.	Рисунок Г.17

В таблице 20 приведены тест-кейсы проверки вкладки «JWT». Тест-кейсы покрывают сценарии поиска, фильтрации данных в таблице внешних систем, а также изменения срока действия JWT токена.

Таблица 20 – Тест-кейсы вкладки «JWT»

Название теста	Шаги выполнения	Ожидаемый результат	Наблюдаемый результат
Корректное отображение всех данных	Открыть вкладку «JWT»	Данные внешних систем, токенов и журнал событий корректно отображены.	Рисунок Г.18

Продолжение таблицы 20

Название теста	Шаги выполнения	Ожидаемый результат	Наблюдаемый результат
Осуществить поиск и фильтрацию по данным таблицы внешних систем	Поиск по идентификатору и фильтрация по score «data.write»	Корректное отображение данных	Рисунок Г.19
Поменять срок действия токена	Поменять срок действия токена, запросить от имени внешней системы новый токен.	Срок действия токена изменился.	Рисунок Г.20

В таблице 21 приведены тест-кейсы проверки вкладки «OAuth». Тест-кейсы покрывают сценарии сортировки, фильтрации данных в таблице журнала событий, а также установки нового срока действия токена доступа.

Таблица 21 – Тест-кейсы вкладки «OAuth»

Название теста	Шаги выполнения	Ожидаемый результат	Наблюдаемый результат
Корректное отображение всех данных	Открыть вкладку «OAuth»	Данные внешних систем, токенов и журнал событий корректно отображены.	Рисунок Г.21
Осуществить поиск и фильтрацию по данным таблицы журнала событий	Поиск по сообщению и фильтрация по дате	Корректное отображение данных	Рисунок Г.22
Поменять срок действия токена доступа	Поменять срок действия токена, запросить от имени внешней системы новый токен.	Срок действия токена изменился.	Рисунок Г.23

Результаты тестирования интерфейса администрирования показали полное соответствие ожидаемому поведению. Несовпадений между ожидаемыми и наблюдаемыми результатами не выявлено.

5.3 Оценка качества программного кода

Одним из ключевых этапов обеспечения надёжности и сопровождаемости программного обеспечения является количественная оценка качества программного кода. Для этого применяются специальные метрики,

позволяющие анализировать такие характеристики, как сложность, объём, читаемость и структуру кода. Метрики широко используются в процессе разработки для выявления потенциальных проблем, оценки трудоёмкости и качества программных решений.

Метрики сложности программного обеспечения традиционно классифицируют на три основные группы [26]:

- метрики размера оценивают объём программного кода, они опираются на количественные характеристики, такие как количество операторов, строк кода, уникальных элементов, например, метрики Холстеда;
- метрики сложности потока управления анализируют логическую структуру выполнения программы, наиболее известный представитель этой группы – цикломатическая сложность Маккейба, отражающая количество независимых путей выполнения;
- метрики сложности потока данных оценивают способы использования данных в программе, позволяют выявить избыточные или неэффективные взаимодействия с данными, одним из распространенных подходов является метрика Чепина.

В данной выпускной квалификационной работе для количественной оценки сложности программного кода будут применяться метрики Холстеда – набор показателей, разработанных Морисом Холстедом в 1977 году. Этот подход основывается на анализе исходного текста программы путём подсчёта количества операторов и операндов для вычисления сложности и объема программы. Перечислим основные метрики Холстеда [26]:

- словарь программы (n) находится как сумма уникальных операторов и операндов:

$$n = n1 + n2,$$

где $n1$ – число уникальных операторов, $n2$ – число уникальных операндов;

- длина программы (N):

$$N = N1 + N2,$$

где $N1$ – общее количество операторов, $N2$ – общее количество операндов;

– объем программы (V) определяет информационный объем программы:

$$V = N * \log_2(n);$$

– сложность (D) показывает, насколько трудно понимать и изменять код:

$$D = \left(\frac{n1}{2}\right) * \left(\frac{N2}{n2}\right);$$

– трудоемкость (E) оценивает количество ментальных усилий, необходимых для написания кода:

$$E = D * V.$$

Рассчитаем данные метрики для функции проверки данных внешней системы в рамках протокола OAuth 2.0 (листинг 1). В таблице 22 приведен подсчет элементов кода.

Листинг 1 – Функция проверки данных клиента

```
public (bool, ClientInfo?) VerifyClient(string clientId, string
secretHash, string url)
{
    var client = _dbContext.Clients.Include(c =>
c.AllowedGrantTypes).Include(c => c.AllowedScopes).
Include(c => c.ClientSecrets).Include(c =>
c.AllowedCorsOrigins).AsNoTracking()
.FirstOrDefault(cl => cl.ClientId == clientId);

    if (client == null)
    {
        string error = $"Клиент с clientId: {clientId} не найден»;

        _auditClient.AddRecord(DataFactory.BuildEventInfo(clientId, error,
EventType.TokenReceiving,
SingleAuthMechanism.AuditApi.LogLevel.Warning));

        _logger.LogWarning(error);
        return (false, null);
    }

    if (!client.AllowedCorsOrigins.Any(origin => origin.Origin ==
url))
    {
        string error = $"Недопустимый адрес запроса {url} клиента
{clientId}»;
    }
}
```

```

_auditClient.AddRecord(DataFactory.BuildEventInfo(clientId, error,
    EventType.TokenReceiving,
    SingleAuthMechanism.AuditApi.LogLevel.Warning));

_logger.LogWarning(error);
return (false, null);
}

var isValidsecret = client.ClientSecrets.Any(secret =>
secret.Value == secretHash);

return (isValidsecret, isValidsecret ? new ClientInfo()
{
    ClientId = client.ClientId,
    ClientName = client.ClientName,
    CreatedAt = client.Created,
    Enabled = client.Enabled,
    AllowedScopes = client.AllowedScopes.Select(scope =>
scope.Scope).ToList(),
    AllowedGrants = client.AllowedGrantTypes.Select(grant =>
grant.GrantType).ToList()
} : null);
}

```

Таблица 22 – Элементы кода функции проверки данных клиента

Операторы			Операнды		
Оператор	Номер	Число операторов (N1)	Операнд	Номер	Число операндов (N2)
=	1	10	client	1	16
.	2	22	error	2	6
()	3	24	\$»Клиент с clientId: {clientId} не найден»	3	1
=>	4	9	_auditClient	4	2
==	5	4	clientId	5	5
if	6	2	EventType.TokenReceiving	6	2
\$»«	7	2	LogLevel.Warning	7	2
return	8	3	false	8	2
!	9	1	null	9	3
? :	10	1	origin	10	1
new	11	1	url	11	2
			\$»Недопустимый адрес запроса {url} клиента {clientId}»	12	1
			isValidsecret	13	3
			secret	14	1
			secretHash	15	1
			new ClientInfo() {...}	16	1
			ClientId	17	1

Продолжение таблицы 22

Операторы			Операнды		
Оператор	Номер	Число операторов (N1)	Операнд	Номер	Число операндов (N2)
			ClientName	18	1
			CreatedAt	19	1
			Enabled	20	1
			AllowedScopes	21	1
			AllowedGrants	22	1
			scope	23	1
			grant	24	1
			dbContext	25	1
Всего:	11	79	Всего:	25	58

На основании данных, приведённых в таблице 22, были рассчитаны метрики Холстеда. Словарь программы составляет 36, длина программы – 137, объём программы равен 708,3, сложность – 12,76, а усилия программиста оцениваются в 9037,9.

На основе расчётов метрик Холстеда можно сделать вывод, что программа обладает средней степенью алгоритмической сложности. Объём и усилия программиста ($V = 708.3$, $H = 9037.9$) указывают на умеренную трудоёмкость реализации и сопровождения данного фрагмента кода.

Заключение

В результате выполнения выпускной квалификационной работы была разработана подсистема единой аутентификации внешних систем, поддерживающих разные протоколы аутентификации/авторизации, для доступа к ресурсам ИС ЭДО. В программных средствах реализована аутентификация на основе токенов JWT и протокола OAuth 2.0 Client Credentials Grant, однако разработанная подсистема обеспечивает легкую встраиваемость новых компонентов, поддерживающих другие протоколы аутентификации, и позволяет расширять функционал без необходимости внесения больших изменений в существующие программные компоненты. Помимо поддержки разных протоколов аутентификации, разработанное решение предоставляет веб-интерфейс для администрирования систем, позволяющий:

- управлять внешними системами;
- блокировать доступ;
- настраивать разрешённые права доступа внешних систем;
- вести учет полученных токенов;
- просматривать журнал безопасности и отслеживать действия внешних систем в рамках процесса аутентификации и доступа к ресурсам.

Это расширяет возможности контроля и адаптации механизма под конкретные задачи, повышает удобство его эксплуатации по сравнению с рядом существующих решений.

На этапе анализа предметной области был проведен обзор основных протоколов аутентификации и авторизации пользователей и систем. Для каждого из них были рассмотрены особенности реализации, преимущества и ограничения, а также определены типовые сценарии применения. Также на основе полученных данных был проведен их сравнительный анализ. На данном этапе была поставлена задача на разработку программных средств, проведен функционально-стоимостной анализ для оценки ресурсов, требуемых для реализации функционала.

На этапе анализа требований к подсистеме обеспечения единой аутентификации были определены функциональные и нефункциональные требования, описывающие, что именно должна делать система и как она должна это выполнять.

На этапе проектирования программных средств были спроектированы интерфейс администрирования и подсистема единой аутентификации внешних систем.

В ходе реализации была разработана серверная и клиентская части механизма единой аутентификации. Серверная часть имеет архитектуру на основе сервисов, каждый сервис выполняет четко определенную задачу, является самостоятельным компонентом, мало зависит от других сервисов. Веб-интерфейс администрирования обеспечивает удобное управление данными единой подсистемы аутентификации.

На этапе контроля качества программных средств были проведены функциональное тестирование наиболее значимых функций с помощью классов эквивалентности и диаграммы состояний и переходов, тестирование клиентского веб-интерфейса и оценка качества программного кода на основе метрик Холстеда. Несовпадений ожидаемого и наблюдаемого результатов не было выявлено, тестирование показало корректность работы программных средств.

Также корректную работу разработанной подсистемы подтверждает ее внедрение в технологическую платформу в компании «ИнфоТеКС», что подтверждается актом о внедрении, приведенном в приложении Д.

Список использованных источников

1. Полянская, О. Ю. Инфраструктуры открытых ключей / О. Ю. Полянская. – М.: Горячая линия – Телеком, 2004. – 224 с.
2. Stallings, W. Network Security Essentials: Applications and Standards = Основы сетевой безопасности: приложения и стандарты / W. Stallings; [пер. с англ. автора ВКР]. – 4th ed. – Upper Saddle River: Pearson Education, 2011. – 432 p.
3. [RFC 4120] The Kerberos Network Authentication Service (V5) [Электронный ресурс] / C. Neuman, T. Yu, S. Hartman, K. Raeburn; IETF. – 2005. – Июль. – URL: <https://www.rfc-editor.org/rfc/rfc4120.txt> (дата обращения: 10.04.2025).
4. [RFC 5280] Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile [Электронный ресурс] / D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk; IETF. – 2008. – May. – URL: <https://www.rfc-editor.org/rfc/rfc5280> (дата обращения: 12.04.2025).
5. [RFC 8446] The Transport Layer Security (TLS) Protocol Version 1.3 [Электронный ресурс] / E. Rescorla; IETF. – 2018. – August. – URL: <https://www.ietf.org/rfc/rfc8446.txt> (дата обращения: 13.04.2025).
6. [RFC 4251] The Secure Shell (SSH) Protocol Architecture [Электронный ресурс] / T. Ylonen, C. Lonvick; IETF. – 2006. – January. – URL: <https://www.rfc-editor.org/rfc/rfc4251> (дата обращения: 15.04.2025).
7. [RFC 7522] Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants [Электронный ресурс] / B. Campbell, C. Mortimore, M. Jones; IETF. – 2015. – May. – URL: <https://datatracker.ietf.org/doc/html/rfc7522> (дата обращения: 16.04.2025).
8. Wilson, Y. Solving Identity Management in Modern Applications: Demystifying OAuth 2.0, OpenID Connect, and SAML 2.0 = Управление идентификацией в современных приложениях: разоблачение OAuth 2.0, OpenID Connect и SAML 2.0 / Yvonne Wilson, Abhishek Hingnikar; [пер. с англ. автора ВКР]. – 1st ed. – Berkeley, CA: Apress, 2023. – 287 p.

9. [RFC 6749] The OAuth 2.0 Authorization Framework [Электронный ресурс] / D. Hardt; IETF. – 2012. – October. – URL: <https://datatracker.ietf.org/doc/html/rfc6749> (дата обращения: 16.04.2025).
10. Siriwardena, P., Dias, N. Microservices Security in Action = Безопасность микросервисов в действии / P. Siriwardena, N. Dias; [пер. с англ. автора ВКР]. – 1st ed. – Shelter Island, NY: Manning Publications, 2020. – 550 p.
11. Стручков. Как работает OAuth2 [Электронный ресурс]. – 2023. – URL: <https://struchkov.dev/blog/ru/how-oauth2-works/> (дата обращения: 20.04.2025).
12. [RFC 7519] JSON Web Token (JWT) [Электронный ресурс] / M. Jones, J. Bradley, N. Sakimura; IETF. – 2015. – May. – URL: <https://www.rfc-editor.org/rfc/rfc7519.html> (дата обращения: 15.04.2025).
13. Князькова В. С. Развитие функционально-стоимостного анализа и особенности его применения при проектировании программных продуктов // *π-Economy*. – 2022. – Т. 15, № 5. – С. 111–121. – DOI: <https://doi.org/10.18721/JE.15507> (дата обращения: 16.04.2025).
14. Программист: зарплаты в Пензе [Электронный ресурс] // *ГородРабот.ру*. – URL: <https://penza.gorodrabot.ru/salaries/programmist> (дата обращения: 17.04.2025).
15. Business Studio. Методология IDEF0: руководство по моделированию бизнес-процессов [Электронный ресурс]. – URL: <https://www.businessstudio.ru/help/docs/current/doku.php/ru/csdesign/bpmodeling/idef0> (дата обращения: 18.04.2025).
16. Буч Г., Рамбо Г., Якобсон И. Язык UML. Руководство пользователя: пер. с англ. / Н. Мухин. – 2-е изд. – Москва: ДМК Пресс, 2006. – 496 с.
17. Ричардс М., Форд Н. Фундаментальный подход к программной архитектуре: паттерны, свойства, проверенные методы. – Санкт-Петербург: Питер, 2023. – 448 с.

18. Троелсен, Эндрю, Джепикс, Филипп. T70 Язык программирования C# 7 и платформы .NET и .NET Core, 8-е изГ. : Пер. с англ. – СПб. : ООО «Диалектика», 2018 – 1328 с.
19. Лок Э. Л73 ASP.Net Core в действии / пер. с англ. Г. А. Беликова. – М.: ДМК Пресс, 2021. – 906 с.
20. Duende Software. IdentityServer [Электронный ресурс]. – URL: <https://duendesoftware.com/products/identityserver> (дата обращения: 15.05.2025).
21. Томас Марк Тиленс T56 React в действии. – СПб.: Питер, 2019. – 368 с.
22. TestIT. Функциональное и нефункциональное тестирование: в чем разница? [Электронный ресурс]. – URL: <https://testit.software/blog/post/funkcionalnoe-i-nefunkcionalnoe-testirovanie-v-chem-raznica> (дата обращения: 15.05.2025).
23. SkillFactory. Методы черного, белого и серого ящиков: в чем разница? [Электронный ресурс]. – URL: <https://blog.skillfactory.ru/metodi-chernogo-belogo-serogo-yashikov/> (дата обращения: 15.05.2025).
24. Куликов, С. Тестирование программного обеспечения. Базовый курс / С. Куликов. – 3-е изГ. – СПб.: Питер, 2020. – 464 с.
25. Савин Р. Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах. – М.: Дело, 2007. – 312 с.
26. Новичков А. Метрики кода и их практическая реализация в Subversion и ClearCase. Часть 1: Метрики [Электронный ресурс] // RusArticles. – 2011. – URL: <https://web.archive.org/web/20110913071425/http://www.rusarticles.com/programmy-statya/metriki-koda-i-ix-prakticheskaya-realizaciya-v-subversion-i-clearcase-chast-1-metriki-603328.html> (дата обращения: 18.05.2025).

Приложение А

(справочное)

Глоссарий

SaaS (Software as a Service) – модель облачного предоставления программного обеспечения, при которой приложение размещается на серверах провайдера, доступ к нему предоставляется через интернет, пользователи не управляют инфраструктурой.

API (Application Programming Interface) – программный интерфейс приложений, набор строго определенных методов, правил взаимодействия, протоколов для обмена данными между разными программными компонентами.

Протокол – это строго формализованный набор правил и стандартов, определяющий формат передачи данных, последовательность действий для проверки подлинности, методы защиты.

Симметричное шифрование – это метод шифрования данных, при котором один и тот же секретный ключ используется как для шифрования, так и для расшифрования информации.

Асимметричное шифрование – метод шифрования данных, который предполагает использование двух математически связанных ключей: открытый (публичный) ключ для шифрования данных и закрытый (приватный) ключ для их расшифрования.

Сертификат X.505 – это стандартизированный цифровой документ, который однозначно связывает открытый криптографический ключ с идентификационными данными его владельца (физического лица, устройства или сервиса) и заверяется электронной подписью удостоверяющего центра.

PKI (Public Key Infrastructure) – комплекс технологий, стандартов и процедур, обеспечивающих безопасное управление цифровыми ключами и сертификатами для аутентификации, шифрования и цифровой подписи в распределённых средах.

MAC (Message Authentication Code) – это криптографический механизм, который позволяет гарантировать целостность данных, обеспечивает аутентификацию источника данных с использованием секретного ключа.

CORS (Cross-Origin Resource Sharing) – это механизм безопасности, реализуемый в веб-браузерах, который позволяет или ограничивает веб-приложениям обращаться к ресурсам, находящимся на другом домене.

PKCE (Proof Key for Code Exchange) – это расширение протокола OAuth 2.0, предназначенное для предотвращения атак перехвата кода авторизации за счёт использования временного, однократного «ключа подтверждения», который генерируется клиентом и преобразуется в хэш.

Конечная точка /introspect – это стандартизированный API-метод, который позволяет сервису проверить валидность и метаданные токена доступа, не умея его самостоятельно декодировать.

Opaque-токен – случайная уникальная строка, которая не может быть расшифрована и прочитана напрямую, ссылается на данные сервера, выдавшего токен.

REST API – это архитектурный стиль взаимодействия между клиентом и сервером через протокол HTTP/HTTPS, основанный на принципах REST (Representational State Transfer), который использует стандартные методы (GET, POST, PUT, DELETE и др.) для доступа и управления ресурсами, представляемыми в виде URL.

URL – это унифицированный указатель ресурса, представляющий собой текстовую строку, которая задаёт адрес и путь к конкретному ресурсу в интернете или сети.

React Hooks – это функции, которые позволяют использовать состояние и другие возможности React (например, жизненный цикл компонента, контекст) внутри функциональных компонентов без необходимости создавать классы.

Приложение Б

(рекомендуемое)

Исходный код серверной части

Листинг Б.1 – Файл «SystemsController.cs» сервиса JWTProvider.Service

```

using JWTProvider.Service.Authorization;
using JWTProvider.Service.Clients;
using JWTProvider.Service.Core;
using JWTProvider.Service.Interaction.InData;
using JWTProvider.Service.Interaction.OutData;
using JWTProvider.Service.Storage;
using JWTProvider.Service.Utills;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Options;
using System.Net;
using UnifiedAuthentication.AuditApi;
using UnifiedAuthentication.SystemsManagementApi;

namespace JWTProvider.Service.Controllers;

[ApiController]
[Route("api/jwt/systems")]
public class SystemsController : ControllerBase
{
    public SystemsController(ILogger<SystemsController> logger,
        IServiceStorage storage, IOptions<JwtSettings> settings,
        IAuditServiceClient auditClient, TokenBuilder
        tokenBuilder, ISystemsManagementClient systemsManagementClient)
    {
        _logger = logger;
        _storage = storage;
        _jwtSettings = settings.Value;
        _auditClient = auditClient;
        _tokenBuilder = tokenBuilder;
        _systemsManagementClient = systemsManagementClient;
    }

    [HttpPost("new")]
    [ProducesResponseType(StatusCodes.Status200OK, Type =
        typeof(ScopeInfo))]

    [ProducesResponseType(StatusCodes.Status422UnprocessableEntity)]
    [ProducesResponseType(StatusCodes.Status423Locked)]
    public async Task<IActionResult>
    RegisterSystem(RegistrationData data)
    {
        return await Task.Run<IActionResult>(() =>
        {
            try
            {
                _logger.LogTrace("Получен запрос на регистрацию
внешней системы {name}", data.SystemName);

                if (_storage.IsExists(data.SystemName))
                {

```

```

        string msg = $"Внешняя система с именем
{data.SystemName} уже зарегистрирована";
        _logger.LogWarning(msg);
        return Problem(msg, statusCode:
(int)HttpStatusCode.Locked);
    }

    _storage.AddSystem(data);

    _systemsManagementClient.AddClient(new
SystemInfo()
    {
        SystemId = data.SystemId,
        Name = data.SystemName,
        Protocol =
UnifiedAuthentication.SystemsManagementApi.ProtocolType.JWT,
        Scopes = _jwtSettings.AllowedScopes.Select(s
=> s.Scope).ToList()
    });

    _auditClient.AddRecord(DataFactory.BuildEventInfo(data.SystemId,
        $"Внешняя система {data.SystemId} успешно
зарегистрирована",
        EventType.Registration,
        UnifiedAuthentication.AuditApi.LogLevel.Information));

    _logger.LogInformation("Внешняя система {name}
успешно зарегистрирована", data.SystemName);

    return Ok(_jwtSettings.AllowedScopes);
}
catch (Exception ex)
{
    string msg = $"При регистрации внешней системы
{data.SystemName} произошла ошибка";
    _logger.LogError(ex, msg);
    return Problem(msg, statusCode:
(int)HttpStatusCode.UnprocessableEntity);
}
});
}

[HttpPost("auth")]
[Authorize(AuthenticationSchemes =
AuthorizationConstants.SystemOwnScheme)]
[ProducesResponseType(StatusCodes.Status200OK, Type =
typeof(ScopeInfo))]

[ProducesResponseType(StatusCodes.Status422UnprocessableEntity)]
public async Task<IActionResult> GetToken()
{
    return await Task.Run<IActionResult>(() =>

```

```

    {
        var systemId = User.FindFirst("system_id)?.Value;
        var scopeClaim = User.FindFirst("scope)?.Value;

        if (string.IsNullOrEmpty(systemId) ||
string.IsNullOrEmpty(scopeClaim))
        {
            _logger.LogWarning("Получен невалидный токен");
            return Problem("Ошибка авторизации", statusCode:
(int)HttpStatusCode.Unauthorized);
        }

        try
        {
            _logger.LogTrace("Получен запрос токена внешней
системы {id}", systemId);

            if (_storage.IsBlocked(systemId))
            {
                string msg = $"Внешняя система {systemId}
заблокирована";

                _auditClient.AddRecord(DataFactory.BuildEventInfo(systemId,
                    $"Ошибка запроса токена: внешняя система
{systemId} заблокирована",
                    EventType.TokenReceiving,
                    UnifiedAuthentication.AuditApi.LogLevel.Warning));

                _logger.LogWarning(msg);
                return Problem(msg, statusCode:
(int)HttpStatusCode.Locked);
            }

            var allowedScopes =
_storage.GetSystemScopes(systemId);
            var tokenScopes = scopeClaim.Split(' ',
StringSplitOptions.RemoveEmptyEntries);

            if (!tokenScopes.All(item =>
allowedScopes.Contains(item)))
            {
                string msg = $"Внешняя система {systemId} не
обладает запрошенными правами доступа";

                _auditClient.AddRecord(DataFactory.BuildEventInfo(systemId, msg,
                    EventType.TokenReceiving,
                    UnifiedAuthentication.AuditApi.LogLevel.Warning));

                _logger.LogWarning(msg);
                return Problem(msg, statusCode:
(int)HttpStatusCode.Forbidden);
            }
        }
    }

```

```

    }

    var jwt =
        _tokenBuilder.BuildSystemToken(tokenScopes, systemId);

    _auditClient.AddRecord(DataFactory.BuildEventInfo(systemId,
        $"Внешняя система {systemId} успешно получила
        токен доступа к ресурсам",
        EventType.TokenReceiving,
        UnifiedAuthentication.AuditApi.LogLevel.Information));

    _logger.LogInformation("Токен для внешней системы
        {id} успешно сформирован", systemId);

    return Ok(jwt);
}
catch (Exception ex)
{
    string msg = $"При получении токена внешней
        системы {systemId} произошла ошибка";

    _auditClient.AddRecord(DataFactory.BuildEventInfo(systemId, msg,
        EventType.TokenReceiving,
        UnifiedAuthentication.AuditApi.LogLevel.Warning));

    _logger.LogError(ex, msg);
    return Problem(msg, statusCode:
        (int)HttpStatusCode.UnprocessableEntity);
}
});
}

[Authorize(AuthenticationSchemes =
    AuthorizationConstants.SystemScheme)]
[RequireScopes("scopes.read")]
[HttpGet("{systemId}/scopes")]
[ProducesResponseType(StatusCodes.Status200OK, Type =
    typeof(List<ScopeInfo>))]
[ProducesResponseType(StatusCodes.Status403Forbidden)]

[ProducesResponseType(StatusCodes.Status422UnprocessableEntity)]
public async Task<IActionResult> GetScopes(string systemId)
{
    return await Task.Run<IActionResult>(() =>
    {
        try
        {
            if (_storage.IsBlocked(systemId))
            {
                string error = $"Внешняя система {systemId}
                заблокирована, доступ невозможен";

```

```

        _logger.LogWarning(error);
        return Problem(error, statusCode:
(int)HttpStatusCode.Forbidden);
    }

    _logger.LogTrace("Получен запрос областей доступа
внешней системы {id}", systemId);

    var scopes = _storage.GetSystemScopes(systemId);

    var scopeInfos = scopes.Select(scope => new
ScopeInfo()
    {
        Scope = scope,
        Path = _jwtSettings.AllScopes.Find(s =>
s.Scope == scope)!.Path
    }).ToList();

    _auditClient.AddRecord(DataFactory.BuildEventInfo(systemId,
        $"Внешняя система {systemId} успешно получила
разрешенные области доступа",
        EventType.ResourceAccess,
        UnifiedAuthentication.AuditApi.LogLevel.Information));

    _logger.LogTrace("Области доступа внешней системы
{id} получены", systemId);

    return Ok(scopeInfos);
}
catch (Exception ex)
{
    string msg = $"При получении областей доступа
внешней системы {systemId} произошла ошибка";

    _auditClient.AddRecord(DataFactory.BuildEventInfo(systemId, msg,
        EventType.ResourceAccess,
        UnifiedAuthentication.AuditApi.LogLevel.Error));

    _logger.LogError(ex, msg);
    return Problem(msg, statusCode:
(int)HttpStatusCode.UnprocessableEntity);
}
});
}

private readonly ILogger<SystemsController> _logger;

private readonly IServiceStorage _storage;

```

```

private readonly IAuditServiceClient _auditClient;

private readonly JwtSettings _jwtSettings;

private readonly TokenBuilder _tokenBuilder;

private readonly ISystemsManagementClient
_systemsManagementClient;
}

```

Листинг Б.2 – Файл «jwt_provider.thrift» сервиса JWTProvider.Service

```

namespace netstd UnifiedAuthentication.JWTProviderApi

/// thrift.exe --gen netstd:net6 -out ./Thrift/ServiceApi
./Thrift/jwt_provider.thrift
/// thrift.exe --gen netstd:net6 -out
../../SystemsManagement/SystemsManagement.Service/Clients/JWTApi
./Thrift/jwt_provider.thrift

/// @brief Тип исключения с сообщением об ошибке.
exception JWTServiceException
{
    /// @brief Сообщение об ошибке
    1: string Message
}

/// @brief Служба аутентификации на JWT.
service JWTService
{
    /// @brief Установка разрешенных областей доступа системе.
    /// @throws JWTServiceException.
    void SetScopes(
        1: required string systemId
        2: required list<string> scopes
    ) throws (
        1: JWTServiceException ex
    )

    /// @brief Установка состояния системе.
    /// @throws JWTServiceException.
    void SetState(
        1: required string systemId
        2: required bool isBlocked
    ) throws (
        1: JWTServiceException ex
    )
}

```

Листинг Б.3 – Файл «oauth_provider.thrift» сервиса OAuthProvider.Service

```

namespace netstd UnifiedAuthentication.OAuthProviderApi

```

```

/// thrift.exe --gen netstd:net6 -out ./Thrift/ServiceApi
./Thrift/oauth_provider.thrift
/// thrift.exe --gen netstd:net6 -out
../../../../SystemsManagement/SystemsManagement.Service/Clients/OAuthApi
./Thrift/oauth_provider.thrift

/// @brief Тип исключения с сообщением об ошибке.
exception OAuthServiceException
{
    /// @brief Сообщение об ошибке
    1: string Message
}

/// @brief Служба аутентификации на основе OAuth 2.0.
service OAuthService
{
    /// @brief Установка разрешенных областей доступа клиенту.
    /// @throws OAuthServiceException.
    void SetScopes(
        1: required string clientId
        2: required list<string> scopes
    ) throws (
        1: OAuthServiceException ex
    )

    /// @brief Установка состояния клиенту.
    /// @throws OAuthServiceException.
    void SetState(
        1: required string clientId
        2: required bool isBlocked
    ) throws (
        1: OAuthServiceException ex
    )
}

```

Листинг Б.4 – Файл «systems_manager.thrift» сервиса

SystemsManagement.Service

```

namespace netstd UnifiedAuthentication.SystemsManagementApi

/// thrift.exe --gen netstd:net6 -out ./Thrift/ServiceApi
./Thrift/systems_manager.thrift
/// thrift.exe --gen netstd:net6 -out
../../../../JWTProvider/JWTProvider.Service/Clients/SystemsManagementApi
./Thrift/systems_manager.thrift
/// thrift.exe --gen netstd:net6 -out
../../../../OAuthProvider/OAuthProvider.Service/Clients/SystemsManagemen
tApi ./Thrift/systems_manager.thrift

/// @brief Тип исключения с сообщением об ошибке.
exception SystemsManagementException
{

```

```

    /// @brief Сообщение об ошибке
    1: string Message
}

/// @brief Протокол.
enum ProtocolType
{
    OAuth,
    JWT
}

/// @brief Информация о системе.
struct SystemInfo
{
    /// @brief Название системы.
    1: required string name
    /// @brief Идентификатор система.
    2: required string systemId
    /// @brief Тип протокола.
    3: required ProtocolType protocol
    /// @brief Разрешенные скопы.
    4: required list<string> scopes
}

/// @brief Служба управления внешними системами.
service SystemsManagementService
{
    /// @brief Получение токена авторизации сервиса.
    /// @throws SystemsManagementException.
    string GetEDMSToken(
        1: required binary certificate
    ) throws (
        1: SystemsManagementException ex
    )

    /// @brief Добавление внешней системы.
    /// @throws SystemsManagementException.
    void AddSystem(
        1: required SystemInfo system
    ) throws (
        1: SystemsManagementException ex
    )
}

```

Листинг Б.5 – Файл «SystemsThriftServerHandler.cs» сервиса

SystemsManagement.Service

```

using SystemsManagement.Service.Clients;
using SystemsManagement.Service.Clients.EDMS;
using SystemsManagement.Service.Storage;
using SystemsManagement.Service.Utills;
using UnifiedAuthentication.JWTProviderApi;

```

```

using UnifiedAuthentication.SystemsManagementApi;

namespace SystemsManagement.Service.Thrift;

internal class SystemsThriftServerHandler :
SystemsManagementService.IAsync
{
    public SystemsThriftServerHandler(IServiceScopeFactory
scopeFactory, ILogger<SystemsThriftServerHandler> logger,
IAuditServiceClient auditClient)
    {
        _scopeFactory = scopeFactory;
        _logger = logger;
        _auditServiceClient = auditClient;
    }

    public Task<string> GetEDMSToken(byte[] certificate,
CancellationTokens cancellationTokens = default)
    {
        try
        {
            _logger.LogTrace("Получен запрос токена авторизации");

            return
Task.FromResult(EDMSSClient.GetServiceToken(certificate));
        }
        catch (Exception ex)
        {
            string msg = $"При запрос токена авторизации произошла
ошибка";
            _logger.LogError(ex, msg);
            throw new JWTServiceException() { Message = ex.Message
};
        }
    }

    public Task AddSystem(SystemInfo? system, CancellationTokens
cancellationTokens = default)
    {
        try
        {
            _logger.LogTrace("Получен запрос добавления системы
{id}", system!.SystemId);

            using var scope = _scopeFactory.CreateScope();

            var storage =
scope.ServiceProvider.GetRequiredService<IServiceStorage>();

            var data = system.Convert<SystemInfo,
Storage.Models.SystemInfo>();
            data.AllowedScopes = system.Scopes!;

```

```

        data.Protocol =
Enum.Parse<Storage.Models.AuthProtocol>(system.Protocol.ToString()
);

        storage.AddSystem(data);

        _logger.LogTrace("Система {id} успешно добавлена",
system.SystemId);

        return Task.CompletedTask;
    }
    catch (Exception ex)
    {
        string msg = $"При добавлении системы
{system!.SystemId} произошла ошибка";
        _logger.LogError(ex, msg);
        throw new JWTServiceException() { Message = ex.Message
};
    }
}

private readonly IServiceScopeFactory _scopeFactory;

private readonly ILogger<SystemsThriftServerHandler> _logger;

private readonly IAuditServiceClient _auditServiceClient;
}

```

Листинг Б.6 – Файл «AuditController.cs» сервиса Audit.Service

```

using Audit.Service.Storage;
using Audit.Service.Storage.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Net;

namespace Audit.Service.Controllers;

[ApiController]
[Route("api/management/audit")]
[Authorize]
public class AuditController : ControllerBase
{
    public AuditController(ILogger<AuditController> logger,
IAuditStorage storage)
    {
        _logger = logger;
        _storage = storage;
    }

    [HttpGet("count")]

```

```

        [ProducesResponseType(StatusCodes.Status200OK, Type =
typeof(int))]

[ProducesResponseType(StatusCodes.Status422UnprocessableEntity)]
    public async Task<IActionResult> GetRecordsCount()
    {
        return await Task.Run<IActionResult>(() =>
        {
            try
            {
                _logger.LogTrace("Получен запрос количества
записей событий в базе данных");

                var count = _storage.GetRecordsCount();

                _logger.LogTrace("Количество записей событий
успешно получено");

                return Ok(count);
            }
            catch (Exception ex)
            {
                string msg = $"При получении количества записей
произошла ошибка";
                _logger.LogError(ex, msg);
                return Problem(msg, statusCode:
(int)HttpStatusCode.UnprocessableEntity);
            }
        });
    }

[HttpGet]
[ProducesResponseType(StatusCodes.Status200OK, Type =
typeof(List<Record>))]

[ProducesResponseType(StatusCodes.Status422UnprocessableEntity)]
    public async Task<IActionResult> GetRecords(int startIndex,
int count)
    {
        return await Task.Run<IActionResult>(() =>
        {
            try
            {
                _logger.LogTrace("Получен запрос записей событий в
указанном диапазоне: [{start}, {end}]",
                startIndex, startIndex + count - 1);

                var records = _storage.GetRecords(startIndex,
count);

                _logger.LogTrace("Записи в указанном диапазоне
успешно получены");
            }
            catch (Exception ex)
            {
                string msg = $"При получении записей событий в
указанном диапазоне произошла ошибка";
                _logger.LogError(ex, msg);
                return Problem(msg, statusCode:
(int)HttpStatusCode.UnprocessableEntity);
            }
        });
    }

```

```

        return Ok(records);
    }
    catch (Exception ex)
    {
        string msg = $"При получении списка записей в
диапазоне: [{startIndex}, {startIndex + count - 1}] произошла
ошибка";
        _logger.LogError(ex, msg);
        return Problem(msg, statusCode:
(int)HttpStatusCode.UnprocessableEntity);
    }
    });
}

private readonly ILogger<AuditController> _logger;

private readonly IAuditStorage _storage;
}

```

Листинг Б.7 – Файл «audit_api.thrift» сервиса Audit.Service

```

namespace netstd UnifiedAuthentication.AuditApi

/// thrift.exe --gen netstd:net6 -out ./Thrift/AuditApi
./Thrift/audit_api.thrift
/// thrift.exe --gen netstd:net6 -out
../../OAuthProvider/OAuthProvider.Service/Clients/AuditApi
./Thrift/audit_api.thrift
/// thrift.exe --gen netstd:net6 -out
../../JWTProvider/JWTProvider.Service/Clients/AuditApi
./Thrift/audit_api.thrift
/// thrift.exe --gen netstd:net6 -out
../../SystemsManagement/SystemsManagement.Service/Clients/AuditApi
./Thrift/audit_api.thrift

/// @brief Тип исключения с сообщением об ошибке.
exception AuditServiceException
{
    /// @brief Сообщение об ошибке
    1: string Message
}

/// @brief Тип события.
enum EventType
{
    /// @brief Регистрация.
    Registration,

    /// @brief получение токена.
    TokenReceiving,
}

```

```

    /// @brief Доступ к ресурсам.
    ResourceAccess,

    /// @brief Управление клиентами.
    ClientManagement
}

/// @brief Уровень логирования.
enum LogLevel
{
    Error,
    Warning,
    Information,
    Trace
}

/// @brief Протокол.
enum ProtocolType
{
    OAuth,
    JWT
}

/// @brief Информация о событии.
struct EventInfo
{
    /// @brief Сообщение.
    1: required string message
    /// @brief Идентификатор системы.
    2: required string systemId
    /// @brief Тип события.
    3: required EventType eventType
    /// @brief Уровень логирования.
    4: required LogLevel level
    /// @brief Тип протокола.
    5: required ProtocolType protocol
}

/// @brief Служба для управления аудитом событий.
service AuditService
{
    /// @brief Добавление записи события.
    /// @throws AuditServiceException.
    void AddRecord(
        /// @brief Информация о событии.
        1: EventInfo eventInfo
    ) throws (
        1: AuditServiceException ex
    )
}

```

Приложение В

(обязательное)

Исходный код клиентской части

Листинг В.1 – Файл «App.jsx»

```
import React from "react";
import { Routes, Route } from "react-router-dom";
import Layout from "../components/Layout";
import Home from "../pages/Home";
import JwtSettings from "../pages/JwtSettings";
import OAuthSettings from "../pages/OAuthSettings";
import { AuthProvider } from "../context/AuthContext";

function App() {
  return (
    <AuthProvider>
      <Layout>
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/jwt" element={<JwtSettings />} />
          <Route path="/oauth" element={<OAuthSettings />} />
        </Routes>
      </Layout>
    </AuthProvider>
  );
}

export default App;
```

Листинг В.2 – Файл «Home.jsx»

```
import React, { useState, useEffect, useMemo } from "react";
import AuthDialog from "../components/AuthDialog";
import {
  getResources,
  getSystems,
  getLogsCount,
  getLogs,
  updateSystemState,
  updateSystemScopes,
} from "../api/management";
import SystemsTable from "../components/SystemsTable";
import LogsTable from "../components/LogsTable";
import ResourceCards from "../components/ResourceCards";
import EditScopesDialog from "../components/EditScopesDialog";
import { useAuth } from "../context/AuthContext";

const Home = () => {
  const { tokens, setTokens } = useAuth(); // { auditToken,
  systemsToken }
  const [resources, setResources] = useState([]);
  const [systems, setSystems] = useState([]);
  const [logsCount, setLogsCount] = useState(0);
  const [logs, setLogs] = useState([]);
  const [loadingData, setLoadingData] = useState(false);
  const [editingSystem, setEditingSystem] = useState(null);
```

```

// Вынесем функцию загрузки логов отдельно
const loadLogs = async () => {
  try {
    const count = await getLogsCount(tokens.auditToken);
    setLogsCount(count);
    const firstLogs = await getLogs(0, count,
tokens.auditToken);
    setLogs(firstLogs);
  } catch (e) {
    console.error("Ошибка при загрузке логов", e);
  }
};

const allScopes = useMemo(() => {
  return [...new Set(resources.flatMap((res) => res.scope ||
[]))];
}, [resources]);

const handleEditScopes = (system) => {
  setEditingSystem(system);
};

const handleSaveScopes = async (systemId, scopes) => {
  try {
    await updateSystemScopes(tokens.systemsToken, systemId,
scopes);
    setSystems((prev) =>
      prev.map((s) =>
        s.system_id === systemId ? { ...s, allowed_scopes:
scopes } : s
      )
    );
    setEditingSystem(null);
    await loadLogs();
  } catch (e) {
    console.error("Ошибка при обновлении scopes:", e);
  }
};

useEffect(() => {
  if (!tokens) return;

  const loadData = async () => {
    setLoadingData(true);
    try {
      const [resResources, resSystems, count] = await
Promise.all([
        getResources(tokens.systemsToken),
        getSystems(tokens.systemsToken),
        getLogsCount(tokens.auditToken),
      ]);
      console.log(resResources);
    }
  }
};

```

```

        setResources(resResources);
        setSystems(resSystems);

        // Загрузим логи отдельно
        await loadLogs();
    } catch (e) {
        console.error(e);
    } finally {
        setLoadingData(false);
    }
};

loadData();
}, [tokens]);

if (!tokens) {
    return <AuthDialog open={true} onSuccess={setTokens} />;
}

// Новый обработчик для переключения статуса системы
const handleToggleEnabled = async (systemId, enabled) => {
    try {
        await updateSystemState(tokens.systemsToken, systemId,
enabled);
        setSystems((prevSystems) =>
            prevSystems.map((sys) =>
                sys.system_id === systemId ? { ...sys, enabled } : sys
            )
        );
        await loadLogs();
    } catch (error) {
        console.error("Ошибка при обновлении статуса системы",
error);
    }
};

if (!tokens) {
    return <AuthDialog open={true} onSuccess={setTokens} />;
}

if (loadingData) return <div>Загрузка данных...</div>;

return (
    <div>
        <h2>API ресурсы</h2>
        <ResourceCards resources={resources} />

        <h2>Системы</h2>
        <SystemsTable
            systems={systems}
            onToggleEnabled={handleToggleEnabled}
            onEditScopes={handleEditScopes}
        />
    </div>
);

```

```

    <EditScopesDialog
      open={!editingSystem}
      onClose={() => setEditingSystem(null)}
      system={editingSystem}
      allScopes={allScopes}
      onSave={handleSaveScopes}
    />

    <h2>Журнал событий</h2>
    <LogsTable logs={logs} />
  </div>
);
};

export default Home;

```

Листинг В.3 – Файл «JwtSettings.jsx»

```

import React, { useState, useEffect, useMemo } from "react";
import {
  getJwtSystems,
  getJwtTokens,
  getLogsCount,
  getLogs,
  setJwtTokenLifetime,
} from "../api/management";
import SystemsTable from "../components/JWT/JWTSystemsTable";
import LogsTable from "../components/JWT/JWTLogsTable";
import TokensTable from "../components/JWT/JWTTokensTable";
import { useAuth } from "../context/AuthContext";
import AuthDialog from "../components/AuthDialog";

const JwtPage = () => {
  const { tokens, setTokens } = useAuth(); // { jwtToken,
  systemsToken, auditToken }
  const [systems, setSystems] = useState([]);
  const [tokensList, setTokensList] = useState([]);
  const [logs, setLogs] = useState([]);
  const [loadingData, setLoadingData] = useState(false);

  const loadLogs = async () => {
    try {
      const count = await getLogsCount(tokens.auditToken);
      const firstLogs = await getLogs(0, count,
tokens.auditToken);
      setLogs(firstLogs);
    } catch (e) {
      console.error("Ошибка при загрузке логов", e);
    }
  };

  const setTokenLifetime = async (expTime) => {

```

```

    try {
      await setJwtTokenLifetime(tokens.jwtToken, expTime);
      await loadLogs();
    } catch (error) {
      console.error("Ошибка при обновлении времени жизни токена",
error);
    }
  };

  useEffect(() => {
    if (!tokens) return;

    const loadData = async () => {
      setLoadingData(true);
      try {
        const [resSystems, resTokens] = await Promise.all([
          getJwtSystems(tokens.jwtToken),
          getJwtTokens(tokens.jwtToken),
        ]);
        setSystems(resSystems);
        setTokensList(resTokens);

        await loadLogs();
      } catch (e) {
        console.error(e);
      } finally {
        setLoadingData(false);
      }
    };

    loadData();
  }, [tokens]);

  if (!tokens) {
    return <AuthDialog open={true} onSuccess={setTokens} />;
  }

  if (loadingData) return <div>Загрузка данных...</div>;

  return (
    <div>
      <h2>Системы</h2>
      <SystemsTable systems={systems} />

      <h2>Токены</h2>
      <TokensTable tokens={tokensList}
setLifetime={setTokenLifetime} />

      <h2>Журнал событий</h2>
      <LogsTable records={logs} />
    </div>
  );
};

```

```
export default JwtPage;
```

Листинг В.4 – Файл «OAuthSettings.js»

```
import React, { useState, useEffect, useMemo } from "react";
import {
  getOAuthSystems,
  getOAuthTokens,
  getLogsCount,
  getLogs,
  setOAuthTokenLifetime,
} from "../api/management";
import ClientsTable from "../components/OAuth/OAuthSystemsTable";
import LogsTable from "../components/OAuth/OAuthLogsTable";
import TokenInfoTable from
"../components/OAuth/OAuthTokensTable.jsx";
import { useAuth } from "../context/AuthContext";
import AuthDialog from "../components/AuthDialog";

const OAuthPage = () => {
  const { tokens, setTokens } = useAuth(); // { jwtToken,
systemsToken, auditToken }
  const [systems, setSystems] = useState([]);
  const [tokensList, setTokensList] = useState([]);
  const [logs, setLogs] = useState([]);
  const [loadingData, setLoadingData] = useState(false);

  const loadLogs = async () => {
    try {
      const count = await getLogsCount(tokens.auditToken);
      const firstLogs = await getLogs(0, count,
tokens.auditToken);
      setLogs(firstLogs);
    } catch (e) {
      console.error("Ошибка при загрузке логов", e);
    }
  };

  const setTokenLifetime = async (expTime) => {
    try {
      await setOAuthTokenLifetime(tokens.oauthToken, expTime);
      await loadLogs();
    } catch (error) {
      console.error("Ошибка при обновлении времени жизни токена",
error);
    }
  };

  useEffect(() => {
    if (!tokens) return;

    const loadData = async () => {
```

```

    setLoadingData(true);
    try {
      const [resSystems, resTokens] = await Promise.all([
        getOAuthSystems(tokens.oauthToken),
        getOAuthTokens(tokens.oauthToken),
      ]);
      setSystems(resSystems);
      setTokensList(resTokens);

      await loadLogs();
    } catch (e) {
      console.error(e);
    } finally {
      setLoadingData(false);
    }
  };

  loadData();
}, [tokens]);

if (!tokens) {
  return <AuthDialog open={true} onSuccess={setTokens} />;
}

if (loadingData) return <div>Загрузка данных...</div>;
console.log(systems);
return (
  <div>
    <h2>Системы</h2>
    <ClientsTable clients={systems} />

    <h2>Токены</h2>
    <TokenInfoTable tokens={tokensList}
setLifetime={setTokenLifetime} />

    <h2>Журнал событий</h2>
    <LogsTable records={logs} />
  </div>
);
};

export default OAuthPage;

```

Приложение Г

(рекомендуемое)

Результаты тестирования программных средств

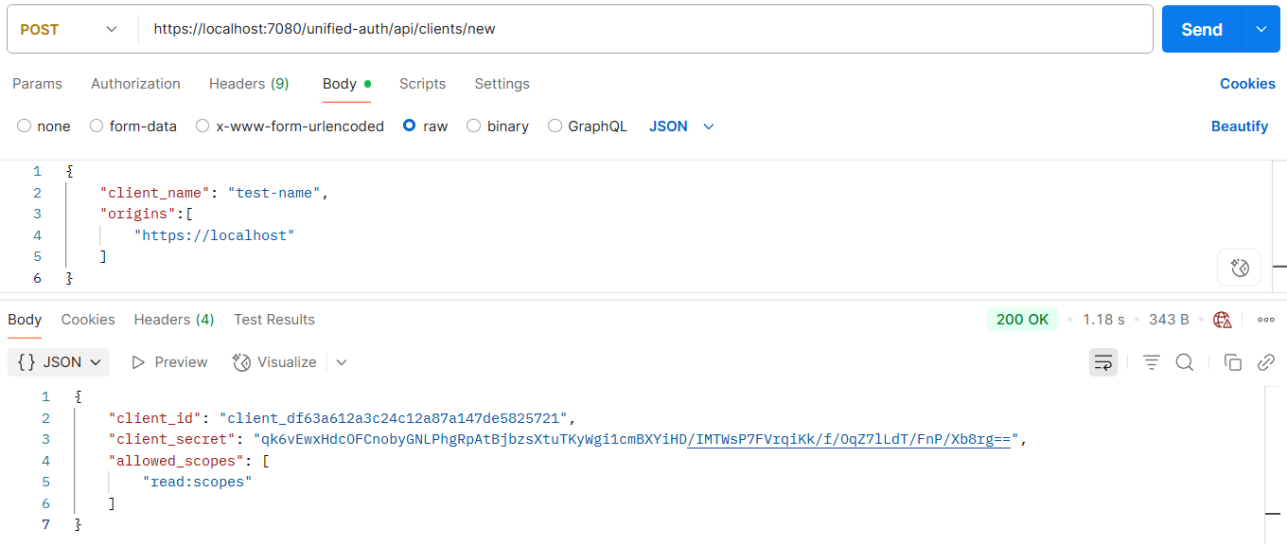


Рисунок Г.1 – Успешная регистрация клиента

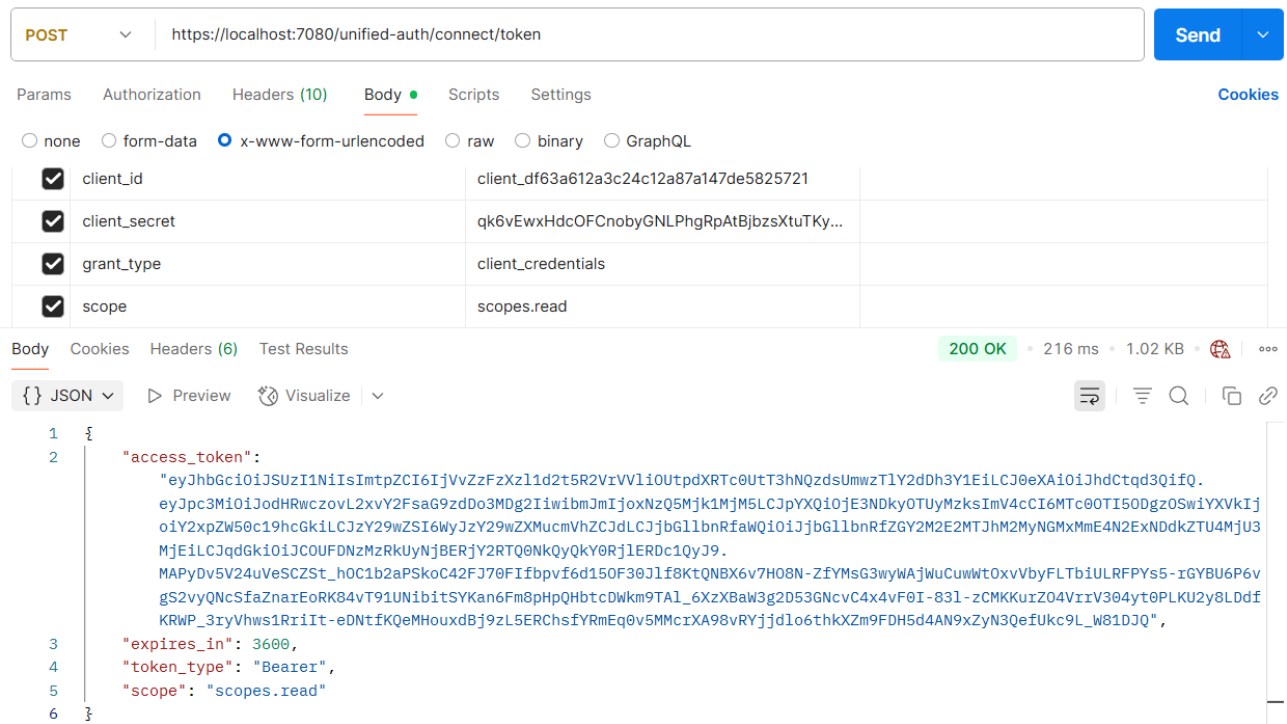


Рисунок Г.2 – Успешное получение токена доступа

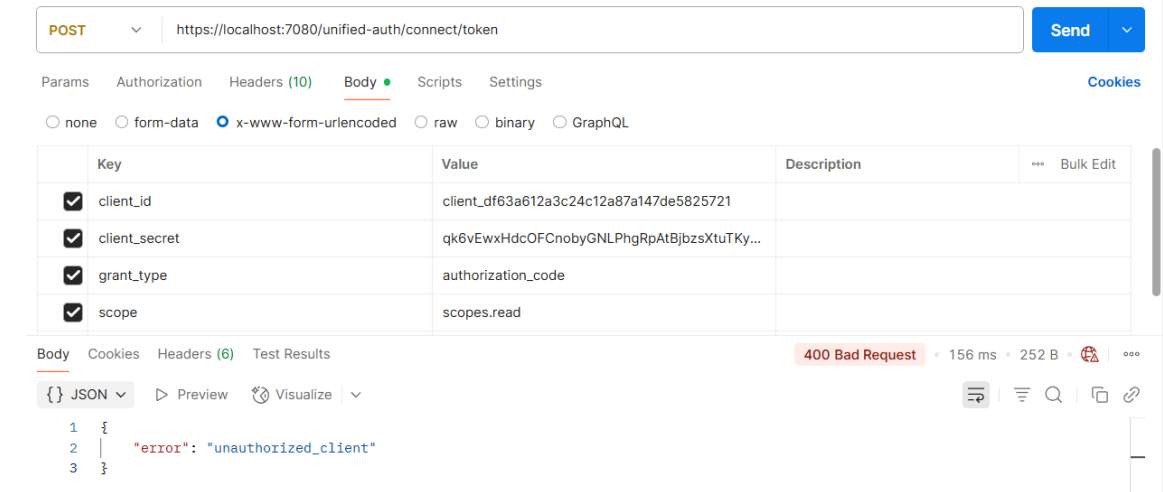


Рисунок Г.3 – Неверный тип гранта

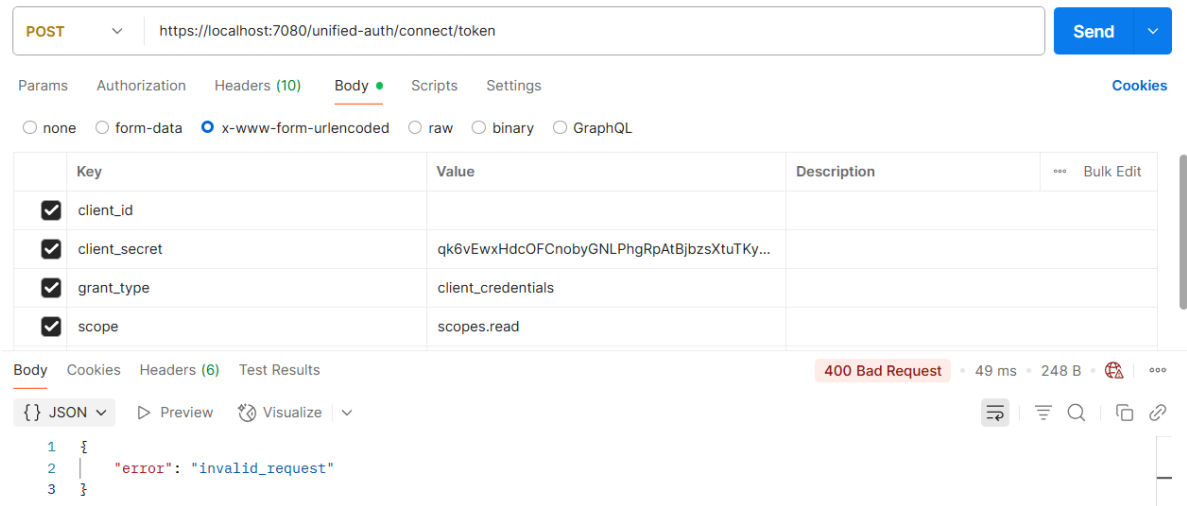


Рисунок Г.4 – Пустой идентификатор клиента

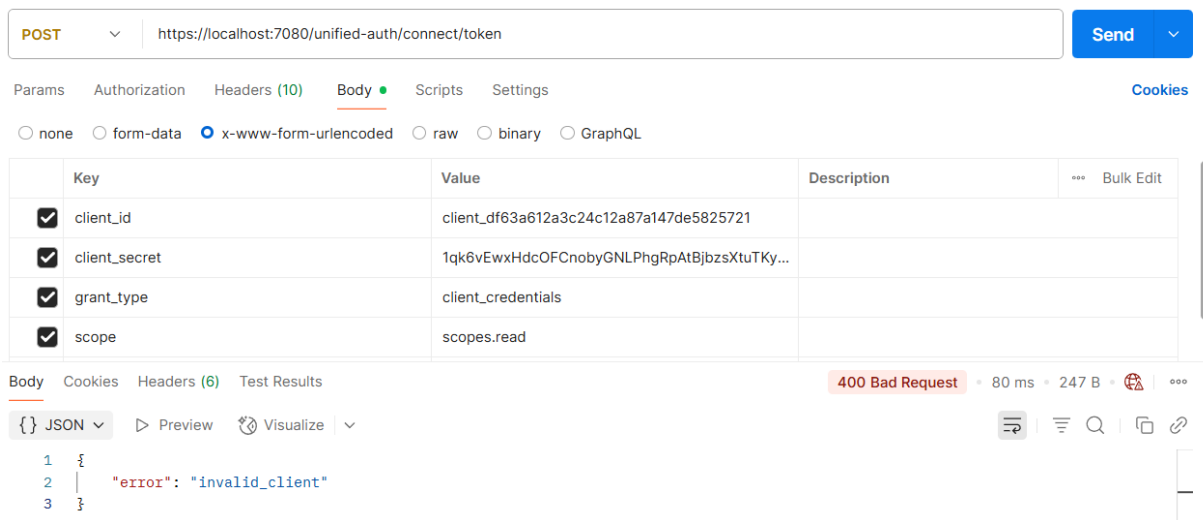


Рисунок Г.5 – Неверный секрет клиента

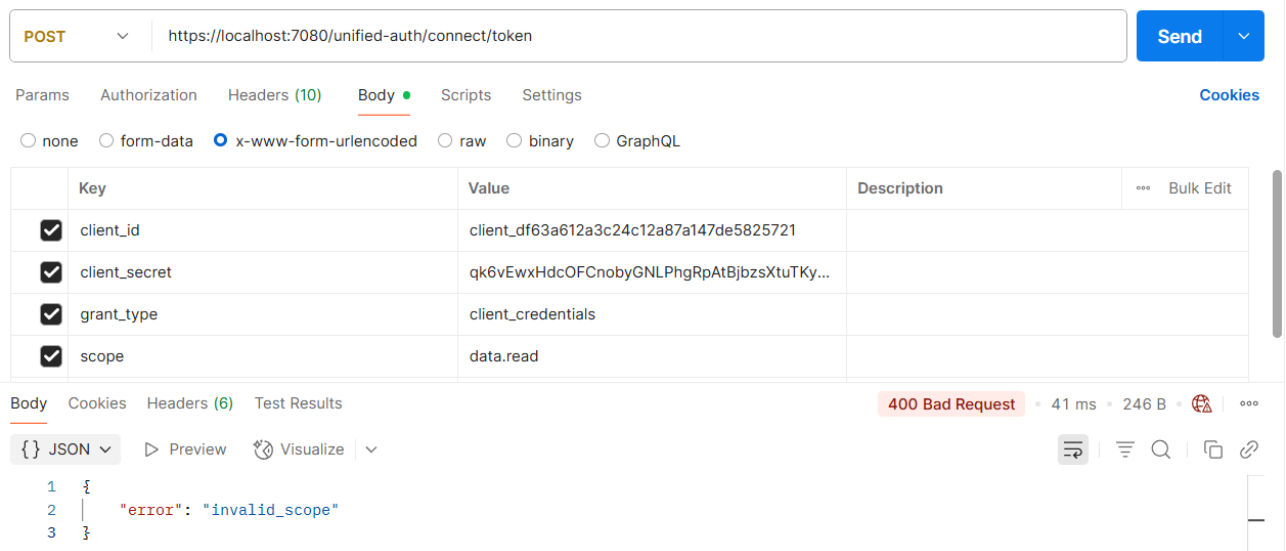


Рисунок Г.6 – Неразрешенная область доступа

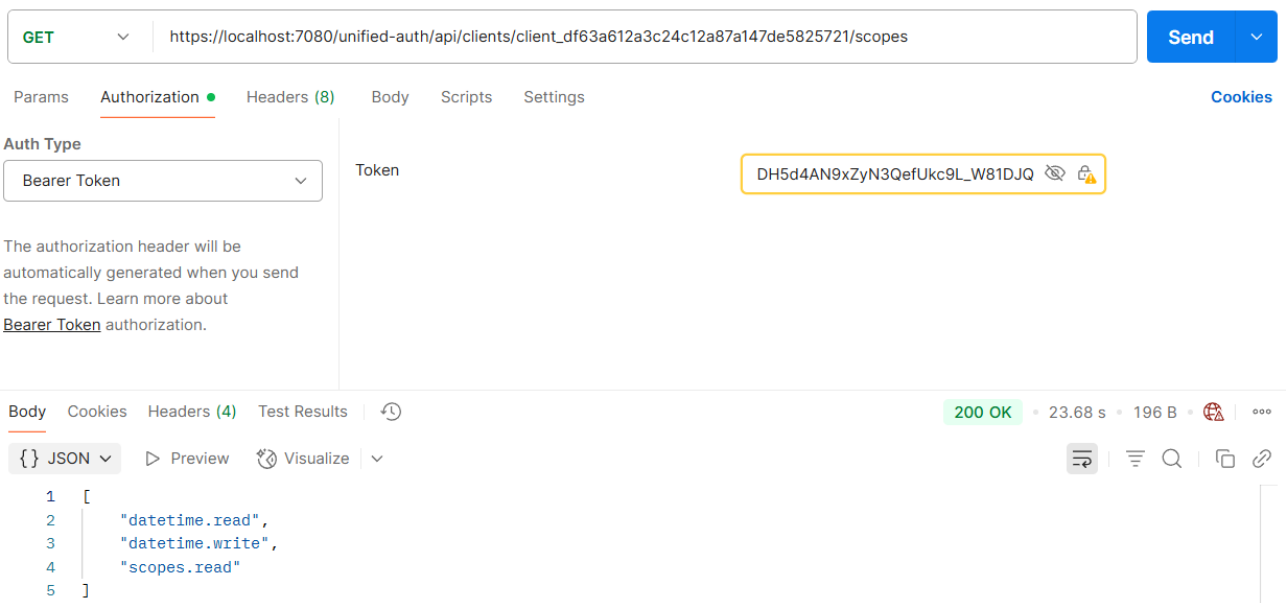


Рисунок Г.7 – Успешное получение клиентом своих областей доступа

```
2025-06-07 14:52:48.4509[INFO]Microsoft.AspNetCore.Authentication.JwtBearer.JwtBearerHandler|ClientBearer was not authenticated. Failure message: IDX10503: Signat
validation failed. Token does not have a kid. Keys tried: '[PII of type 'System.Text.StringBuilder' is hidden. For more details, see https://aka.ms/IdentityModel/PII.]. Numbe
keys in TokenValidationParameters: '1'.
Number of keys in Configuration: '0'.
Exceptions caught:
'[PII of type 'System.Text.StringBuilder' is hidden. For more details, see https://aka.ms/IdentityModel/PII.].
token: '[PII of type 'Microsoft.IdentityModel.JsonWebTokens.JsonWebToken' is hidden. For more details, see https://aka.ms/IdentityModel/PII.]. See https://aka.ms/IDX10
for details.
2025-06-07 14:52:48.4509[INFO]Microsoft.AspNetCore.Authorization.DefaultAuthorizationService|Authorization failed. These requirements were not met:
DenyAnonymousAuthorizationRequirement: Requires an authenticated user.
2025-06-07 14:52:48.4509[INFO]Microsoft.AspNetCore.Authentication.JwtBearer.JwtBearerHandler|AuthenticationScheme: ClientBearer was challenged.
2025-06-07 14:52:48.4509[INFO]Microsoft.AspNetCore.Hosting.Diagnostics|Request finished HTTP/1.1 GET
https://localhost:7085/api/clients/client_df63a612a3c24c12a87a147de5825721/scopes - 401 0 - 2.6063ms
```

Рисунок Г.8 – Токен доступа изменен

```

2025-06-07 15:48:09.9038[INFO]Microsoft.AspNetCore.Authentication.JwtBearer.JwtBearerHandler|ClientBearer was not
authenticated. Failure message: IDX10223: Lifetime validation failed. The token is expired. ValidTo (UTC): '07.06.2025 12:20:39',
Current time (UTC): '07.06.2025 12:48:09'.
2025-06-07 15:48:09.9038[INFO]Microsoft.AspNetCore.Authorization.DefaultAuthorizationService|Authorization failed. These
requirements were not met:
DenyAnonymousAuthorizationRequirement: Requires an authenticated user.
2025-06-07 15:48:09.9038[INFO]Microsoft.AspNetCore.Authentication.JwtBearer.JwtBearerHandler|AuthenticationScheme:
ClientBearer was challenged.
2025-06-07 15:48:09.9038[INFO]Microsoft.AspNetCore.Hosting.Diagnostics|Request finished HTTP/1.1 GET
https://localhost:7085/api/clients/client_df63a612a3c24c12a87a147de5825721/scopes - 401 0 - 3.2642ms

```

Рисунок Г.9 – Токен доступа истек

```

2025-06-07 15:02:13.4327[INFO]Microsoft.AspNetCore.Authentication.JwtBearer.JwtBearerHandler|ClientBearer was not authenticated. Failure message: IDX10214: Audience
validation failed. Audiences: 'resource-proxy api'. Did not match: validationParameters.ValidAudience: 'clients api' or validationParameters.ValidAudiences: 'null'.
2025-06-07 15:02:13.4327[INFO]Microsoft.AspNetCore.Authorization.DefaultAuthorizationService|Authorization failed. These requirements were not met:
DenyAnonymousAuthorizationRequirement: Requires an authenticated user.
2025-06-07 15:02:13.4327[INFO]Microsoft.AspNetCore.Authentication.JwtBearer.JwtBearerHandler|AuthenticationScheme: ClientBearer was challenged.
2025-06-07 15:02:13.4327[INFO]Microsoft.AspNetCore.Hosting.Diagnostics|Request finished HTTP/1.1 GET
https://localhost:7085/api/clients/client_df63a612a3c24c12a87a147de5825721/scopes - 401 0 - 2.5825ms

```

Рисунок Г.10 – Токен доступа не соответствует данному методу

```

2025-06-07 14:56:30.4146[INFO]Microsoft.AspNetCore.Hosting.Diagnostics|Request starting HTTP/1.1 GET
https://localhost:7085/api/clients/client_df63a612a3c24c12a87a147de5825721/scopes - - -
2025-06-07 14:56:30.4146[INFO]Microsoft.AspNetCore.Authorization.DefaultAuthorizationService|Authorization failed. These requirements were not met:
DenyAnonymousAuthorizationRequirement: Requires an authenticated user.
2025-06-07 14:56:30.4146[INFO]Microsoft.AspNetCore.Authentication.JwtBearer.JwtBearerHandler|AuthenticationScheme: ClientBearer was challenged.
2025-06-07 14:56:30.4146[INFO]Microsoft.AspNetCore.Hosting.Diagnostics|Request finished HTTP/1.1 GET
https://localhost:7085/api/clients/client_df63a612a3c24c12a87a147de5825721/scopes - 401 0 - 1.2443ms

```

Рисунок Г.11 – Токен доступа не указан в заголовке запроса

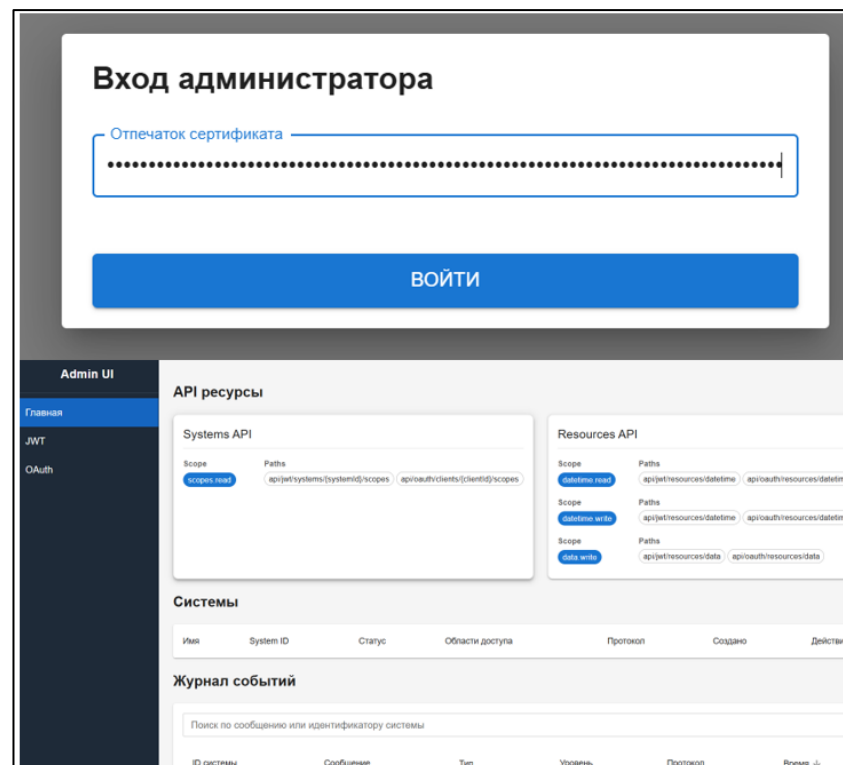


Рисунок Г.12 – Успешная авторизация администратора

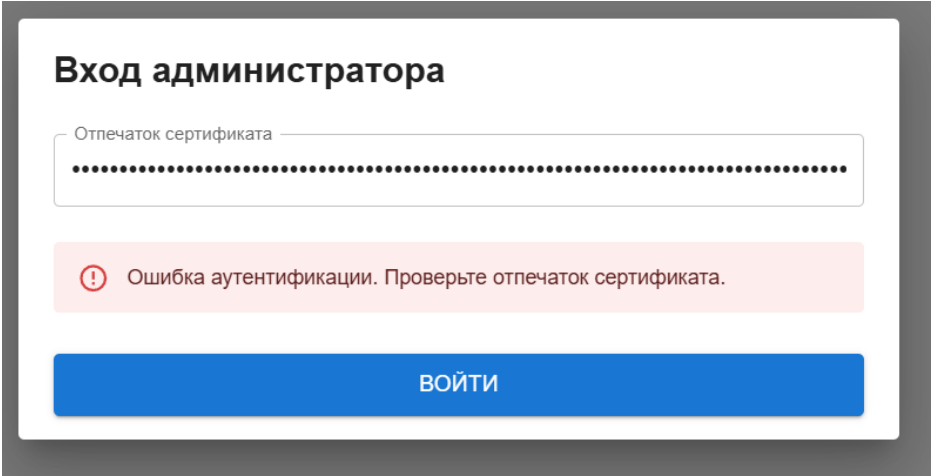


Рисунок Г.13 – Неверный отпечаток сертификата

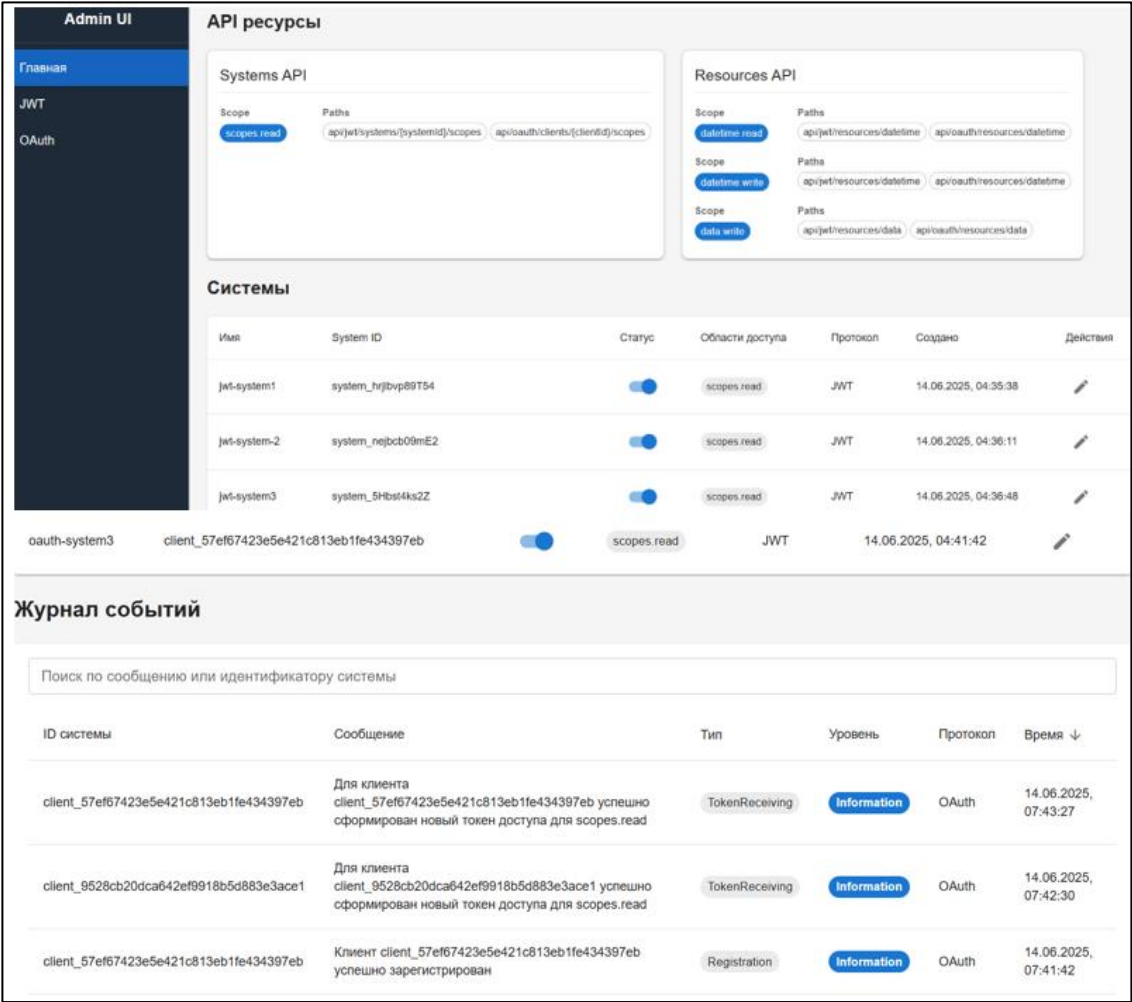


Рисунок Г.14 – Отображение данных главной страницы

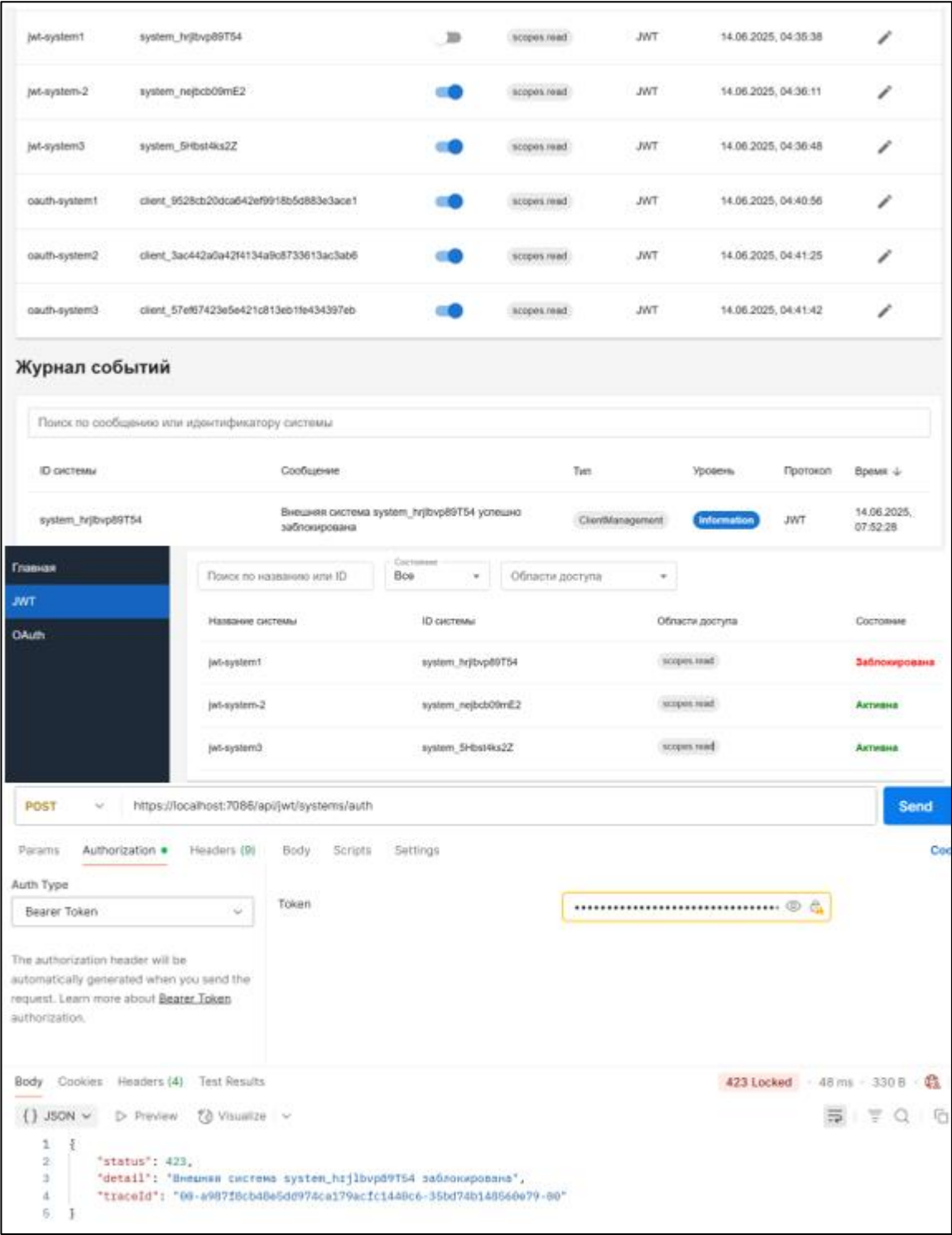


Рисунок Г.15 – Результат блокировки внешней системы»

ID системы	Сообщение	Тип	Уровень	Протокол	Время ↓
system_hrjbvp89T54	Выявлена система system_hrjbvp89T54 успешно разблокирована	ClientManagement	Information	JWT	14.06.2025, 07:55:14

Системы

Поиск по названию или ID:

Статус: Все | Области доступа:

Название системы	ID системы	Области доступа	Статус
jwt-system1	system_hrjbvp89T54	scopes.read	Активна
jwt-system-2	system_nejbc09mE2	scopes.read	Активна
jwt-system3	system_5Hbt4ksZz	scopes.read	Активна

```
POST https://localhost:7086/api/jwt/systems/auth
```

Params Authorization Headers (9) Body Scripts Settings

Auth Type: Bearer Token

The authorization header will be automatically generated when you send this request. Learn more about [Bearer Tokens](#).

Body Cookies Headers (4) Test Results

Raw Preview Visualize

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJzeXBvcmVudDkxLWVucGUldCIsImV4cCI6MTYwODQzMTEyLCJpcnkiOiJzeXBvcmVudDkxLWVucGUldCIsImF1dG8iOiJzeXBvcmVudDkxLWVucGUldCIsImNpdCI6ImF1dG8iOj0KLSkKaSAlG0l0ZwpPU2phLi_k-q02u6J_h266GS197CIII10t1r6GT08Sa2xp2X150MF_gnzawEoq9NM_bvP61I9p
```

Рисунок Г.16 – Результат разблокировки внешней системы

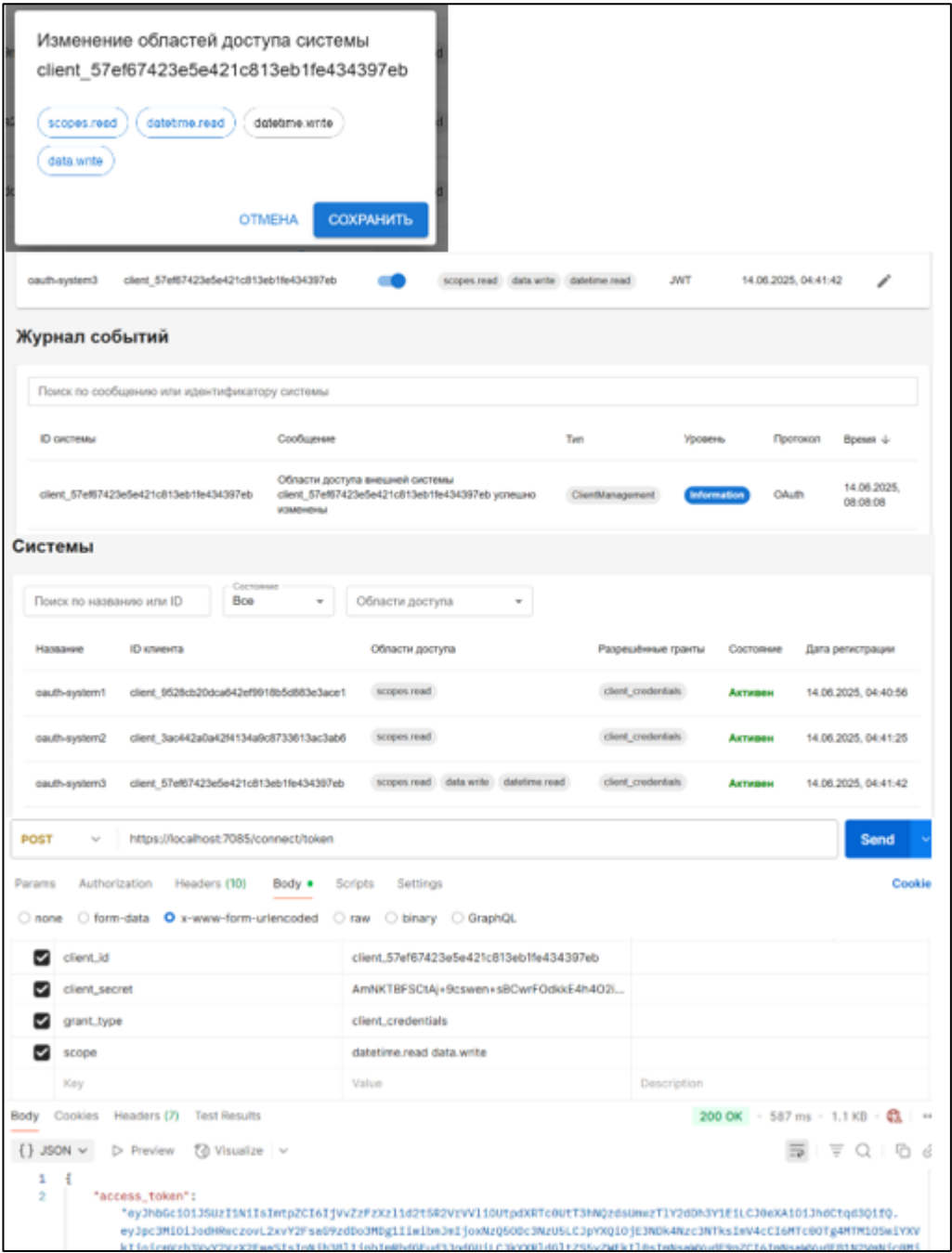


Рисунок Г.17 – Результат установки новых областей доступа внешней системе

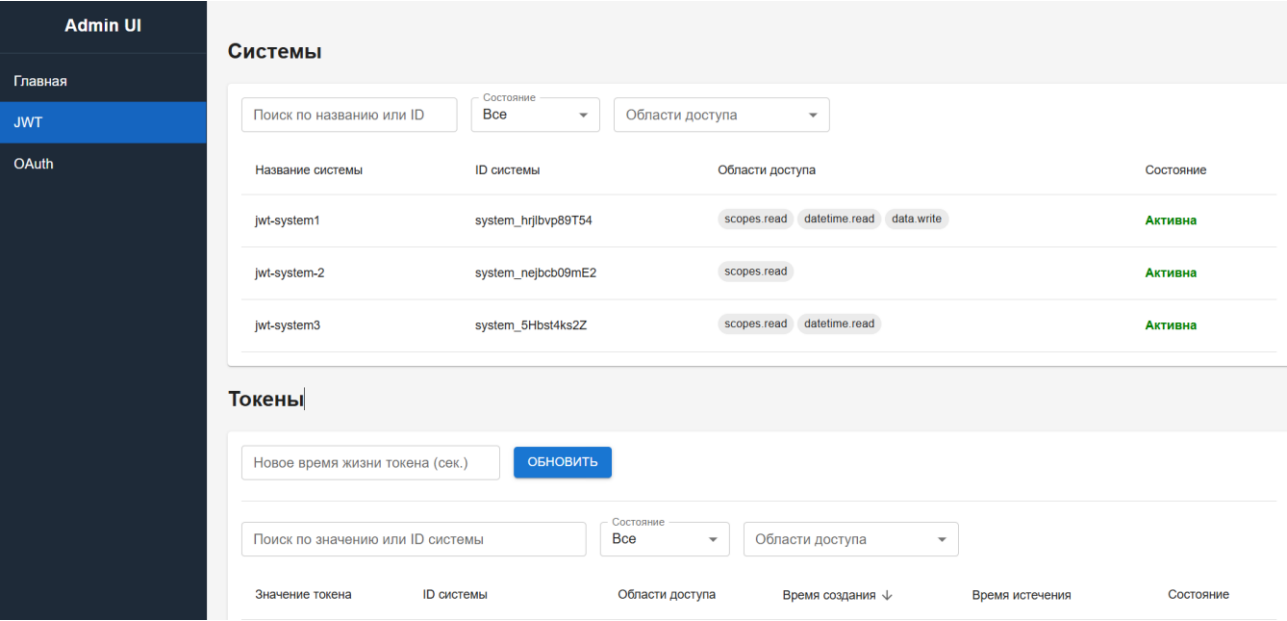


Рисунок Г.18 – Корректное отображение данных вкладки «JWT»

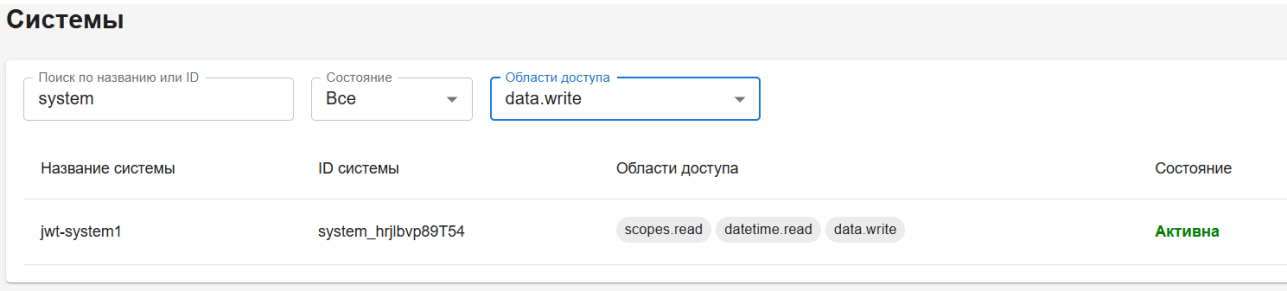


Рисунок Г.19 – Результат применения фильтрации и поиска к таблице систем

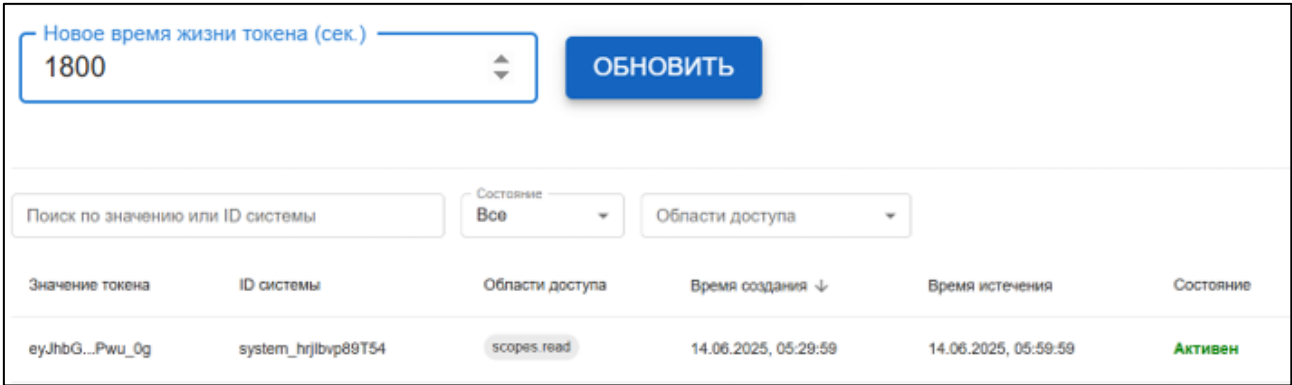


Рисунок Г.20 – Результат изменения срока действия токена JWT

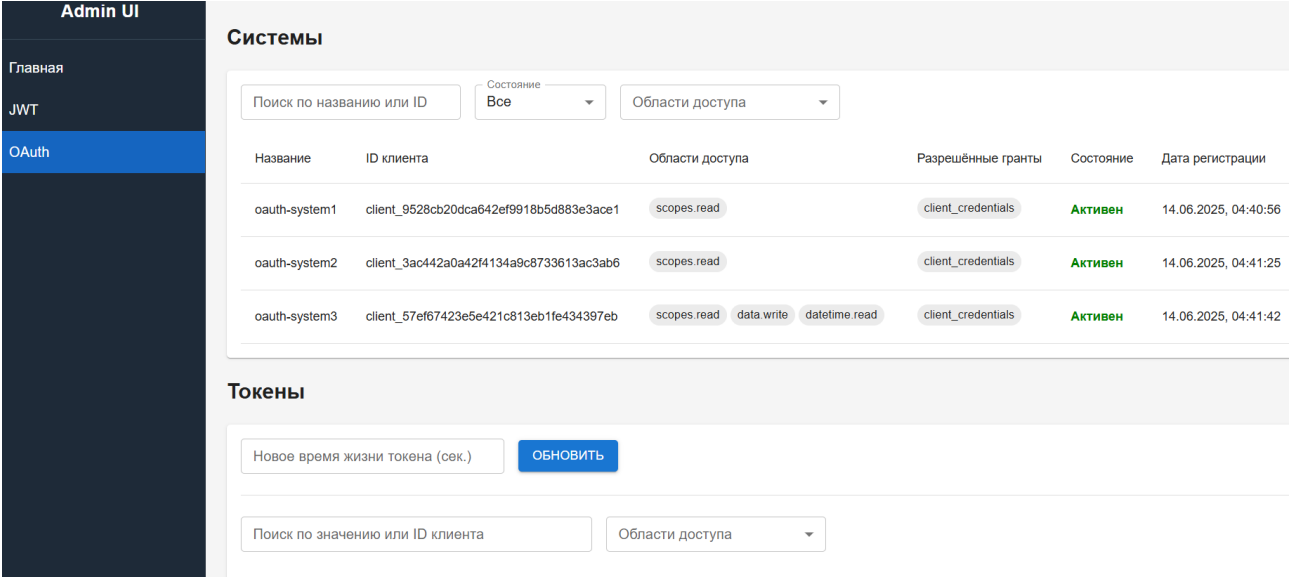


Рисунок Г.21 – Корректное отображение данных вкладки «OAuth»

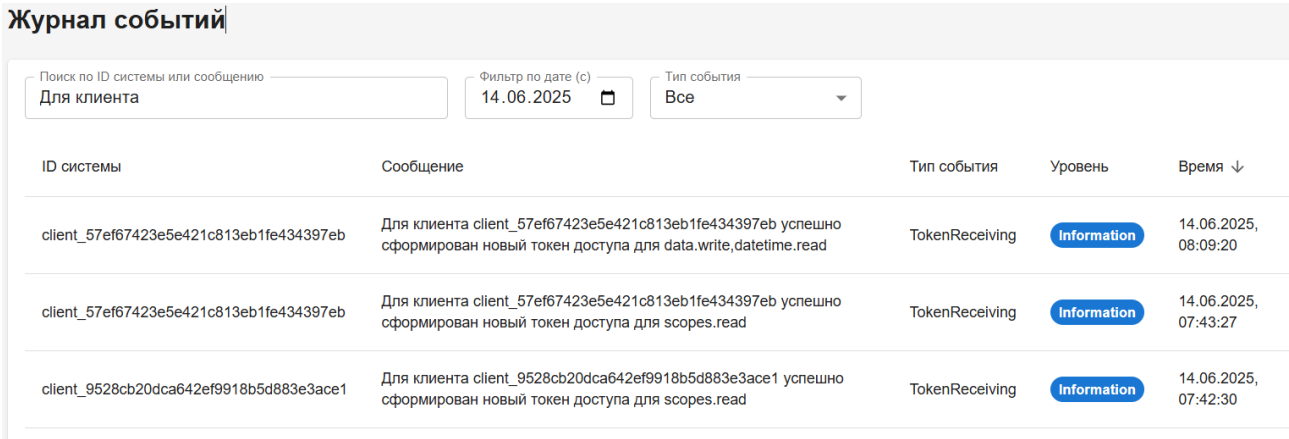


Рисунок Г.22 – Результат применения фильтрации и поиска к таблице журнала протоколирования

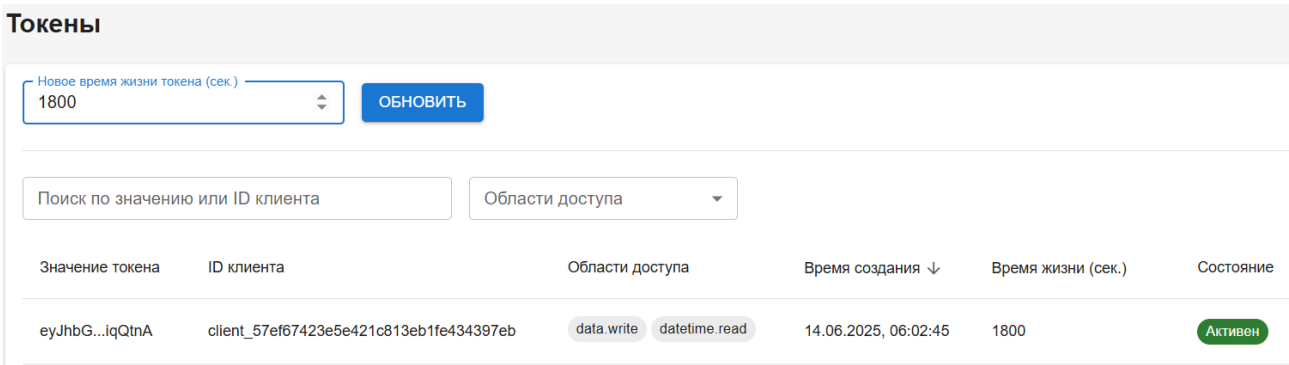


Рисунок Г.23 – Результат изменения срока действия токена доступа

Приложение Д
(рекомендуемое)
Акт о внедрении

infotecs

Акционерное общество
«Информационные технологии
и коммуникационные системы»
(АО «ИнфоТеКС»)

Юридический адрес: ул. Мишина,
дом 56, стр. 2, этаж 2, пом. IX, комната 29

Москва, 127083

Почтовый адрес: а/я № 80,
улица Отрадная, дом 2Б, стр. 1

Москва, 127273

Тел. (495)737-61-92 Факс (495)737-72-78

ОГРН 1027739185066

ИНН/КПП 7710013769/771401001

<http://www.infotecs.ru>

Акционерное общество «Информационные
технологии и коммуникационные системы»
(АО «ИнфоТеКС»)

127083, Москва, улица Мишина, д. 56, стр. 2,
этаж 2, помещение IX, комната 29

.....17.06.2024..... №.....

Акт о внедрении

Настоящим подтверждаю, что Лакеева Софья Романовна является сотрудником компании АО «ИнфоТеКС». Решения, описанные в выпускной квалификационной работе на тему «Программные средства обеспечения механизма единой аутентификации», вошли в состав разрабатываемой технологической платформы.

Выдано Лакеевой Софье Романовне для предъявления по требованию.

Руководитель обособленного
подразделения в г. Пенза



М.М. Грунтович