

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

<i>Факультет</i>	<i>Кафедра</i>
вычислительной техники	МОиПЭВМ
Направление подготовки	09.03.02 «Информационные системы и технологии»
Направленность	Информационные системы и технологии

БАКАЛАВРСКАЯ РАБОТА

на тему

**Программные средства администрирования и
обнаружения событий в системе потоковой
обработки данных**

Студент	_____	<u>Асташкина Татьяна Ильинична</u>
	(подпись, дата)	(ФИО полностью)
Руководитель	_____	<u>Гурьянов Л.В</u>
	(подпись, дата)	(фамилия, инициалы)
Нормоконтролёр	_____	<u>Такташкин Д.В.</u>
	(подпись, дата)	(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 03.06.2026г. № 12)

Заведующий кафедрой	_____	<u>Козлов А.Ю.</u>
	(подпись)	(фамилия, инициалы)

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

Секретарь ГЭК	_____	<u>Попова Н.А.</u>
	(подпись)	(фамилия, инициалы)

Пенза, 2026

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 «26»
декабря 2025 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Асташкина Татьяна Ильинична гр. 22ВИ1 факультета ВТ

<Фамилия, имя, отчество полностью>

2. Тема работы Программные средства администрирования и
обнаружения событий в системе потоковой обработки данных

Тема утверждена приказом ПГУ №1583 от " 30 " декабря 2026 г.

3. Руководитель работы к.т.н., доцент кафедры «МО и ПЭВМ» Гурьянов Л.В.

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

1) *Назначение:*

Администрирование (конструирование, актуализация) и обнаружение событий в режиме реального времени событий в системе потоковой обработки данных

2) *Среда выполнения: Windows 11*

Языковые средства разработки: C#

Инструментальные средства: MS Visual Studio, PostgreSQL, ApacheKafka

Apache Kafka

Технологии разработки: ООП, сервисно-ориентированный подход.

3) *Основные функции разрабатываемого приложения:*

а) Администрирование событий;

б) Обнаружение событий в режиме реального времени;

с) Мониторинг появления событий.

5. Перечень подлежащих разработке вопросов

1) Анализ предметной области и планирование процесса разработки;
2) Анализ требований и постановка задачи на разработку;
3) Разработка программных средств;
4) Оценка качества программных средств;
5) Оформление пояснительной записки.

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и планирование процесса разработки	10%	08.04.2026 – 14.04.2026	
2	Анализ требований и постановка задачи на разработку	20%	15.04.2026 – 25.04.2026	
3	Разработка программных средств	30%	26.04.2026 – 12.05.2026	
4	Оценка качества программных средств	20%	13.05.2026 – 22.05.2026	
5	Оформление пояснительной записки	20%	23.05.2026 – 03.06.2026	

Дата выдачи задания «26» декабря 2025г.

Руководитель бакалаврской работы _____ 26.12.2025 г.
(подпись, дата)

Задание к исполнению принял студент _____ 26.12.2025 г.
(подпись, дата)

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

Реферат

Пояснительная записка содержит 166 листов, 75 рисунков, 34 таблицы, 32 использованных источника и 5 приложений.

Потоковая обработка данных, администрирование событий, обнаружение событий, мониторинг событий, C#, RESTful API, Kafka, ksqlDB.

Объект разработки – программные средства администрирования и обнаружения событий в системе потоковой обработки данных.

Цель выпускной квалификационной работы – разработка программных средств для администрирования, мониторинга и обнаружения событий в системе потоковой обработки данных на платформе Apache Kafka.

Технологии разработки – разработка проводилась на языке программирования C#. В качестве среды разработки использовался редактор программного кода Microsoft Visual Studio.

Среда исполнения – Windows 11, веб-браузер, .NET 10 Windows Desktop Runtime.

Результаты работы – в ходе выполнения выпускной квалификационной работы были созданы программные средства для администрирования, обнаружения и мониторинга событий в рамках потоковой обработки данных.

Все требования к разработке приложения соблюдены.

					ПГУ 09.03.02 – 08BP221.02 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Асташкина Т.И.			Программные средства администрирования и обнаружения событий в системе потоковой обработки данных Пояснительная записка	Лит.	Лист	Листов
Пров.		Гурьянов Л.В.					4	166
						Группа 22ВИ1		
Н. Контр.		Такташкин Д.В						
Утв.		Козлов А.Ю.						

Содержание

Введение.....	8
1 Анализ предметной области и планирование процесса разработки программных средств	10
1.1 Поточковая обработка данных и обнаружение событий	10
1.1.1 Основы организации систем потоковой обработки данных	12
1.1.2 Типология событий в парадигме потоковой обработки данных ...	15
1.1.2.1 Элементарные события	16
1.1.2.2 Сложные составные и агрегированные события	18
1.1.3 Обнаружение сложных составных событий в системах потоковой обработки данных	22
1.2 Сравнительный анализ современных решений для обнаружения событий при потоковой обработке данных.....	24
1.2.1 Обоснование сравнительного анализа	24
1.2.2 Обзор программных платформ для анализа.....	26
1.2.3 Результаты сравнительного анализа и выводы.....	31
1.3 Планирование процесса разработки программных средств	33
2 Анализ требований и постановка задачи на разработку программных средств	44
2.1 Анализ функциональных требований	44
2.2 Анализ нефункциональных требований	51
2.3 Анализ требований к пользовательскому интерфейсу приложений администрирования и диспетчеризации событий	53
2.3.1 Разработка навигационной схемы пользовательского интерфейса приложений администрирования и диспетчеризации событий	53

2.3.2	Разработка прототипов интерфейса приложений администрирования и диспетчеризации событий	54
2.4	Анализ требований к аппаратно-программному обеспечению	71
2.4.1	Требования к аппаратному обеспечению	71
2.4.2	Требования к программному обеспечению	72
2.4.3	Требования к средствам разработки	73
2.5	Постановка задачи на разработку программных средств	75
3	Разработка программных средств	78
3.1	Разработка архитектуры программных средств	78
3.2	Программные средства обнаружения сложных событий в системе поточковой обработки данных Apache Kafka	83
3.2.1	Основные конструкции языка kSQL для реализации программных средств обнаружения событий	83
3.2.2	Программные средства обнаружения элементарных событий	86
3.2.3	Программные средства обнаружения сложных агрегированных событий	87
3.2.4	Программные средства обнаружения составных событий	88
3.3	Разработка базы метаданных и ее интеграция с программными средствами администрирования и обнаружения событий	93
3.4	Разработка серверных программных средств администрирования и обнаружения событий в системе потоковой обработки данных	96
3.5	Разработка клиентских приложений администрирования и диспетчеризации событий	104
3.5.1	Разработка интерфейса	104
3.5.2	Разработка программных средств клиентских приложений	117

4	Контроль качества программных средств	121
4.1	Планирование процесса контроля качества программных средств ..	121
4.1.1	Обзор мероприятий по контролю качества программных средств.....	121
4.1.2	Обоснование выбора модели тестирования	122
4.1.3	Обоснование выбора метрик программного кода	123
4.2	Функциональное тестирование программных средств методом черного ящика	125
4.3	Оценка качества программного кода	128
	Заключение	132
	Список использованных источников	133
	Приложение А (справочное) Глоссарий	135
	Приложение Б (рекомендуемое) Сценарии вариантов использования	138
	Приложение В (обязательное) Листинг серверной части программных средств.....	142
	Приложение Г (обязательное) Листинг клиентской части программных средств.....	149
	Приложение Д (рекомендуемое) Результаты функционального тестирования.....	156

Введение

Стремительный рост объемов данных, с которым приходится сталкиваться в современном мире, неизбежно приводит к тому, что традиционные подходы к обработке информации перестают удовлетворять требованиям оперативности. Во многих областях, от локального бизнеса до крупной промышленности, ценность поступающих данных снижается с течением времени. В подобных условиях требуется обеспечить возможность наиболее быстрой их обработки для последующего реагирования. Для этой цели создаются специальные системы потоковой обработки данных, которые способны анализировать данные в режиме реального времени. Эффективное функционирование подобных систем строится на решении задач администрирования, обнаружения и мониторинга значимых событий в непрерывных потоках.

Целью данной выпускной квалификационной работы является разработка программных средств для администрирования, мониторинга и обнаружения событий в системе потоковой обработки данных на платформе Apache Kafka.

Разрабатываемые программные средства должны предоставлять пользователю инструменты для выявления и мониторинга событий в реальном времени.

Для достижения поставленной цели в работе были определены следующие задачи:

- провести анализ предметной области, изучить архитектурные принципы потоковой обработки данных, понятие события и существующие подходы к их обнаружению;
- выполнить сравнительный анализ программных платформ потоковой обработки;
- спроектировать архитектуру и компоненты разрабатываемых программных средств;
- реализовать программные средства в соответствии с выработанными проектными решениями;

– провести тестирование разработанного продукта и оценить его применимость для решения типовых задач.

В результате выполнения работы были реализованы программные средства, позволяющие специалисту конструировать правила для выявления значимых событий и наблюдать за их обнаружением в реальном времени, что объединит в себе потоковую обработку и обнаружение событий в реальном времени.

В настоящий момент разработан прототип программных средств, который может быть расширен дополнительными модулями и поддержкой более сложных сценариев обнаружения событий.

1 Анализ предметной области и планирование процесса разработки программных средств

1.1 Поточковая обработка данных и обнаружение событий

Термин «поточковая обработка данных» является основополагающим для обширного круга направлений исследований, охватывающих как теоретические аспекты функционирования событийно-ориентированных систем, так и прикладные задачи извлечения ценности из данных в момент их появления.

Поточковая обработка данных – это метод обработки информационных объектов, при котором вычисления выполняются над элементами данных непосредственно в процессе их передачи от источника к получателю, без предварительного сохранения в долговременную память (СУБД или файловую систему) [1]. В отличие от пакетной обработки, которая ожидает накопления определенного объема данных, поточковая обработка использует неограниченные, непрерывно поступающие наборы данных [2]. Это позволяет минимизировать время отклика системы и предоставлять результаты анализа в оперативном режиме, значительно сокращая временной разрыв между возникновением события и реакцией на него.

Поточковая обработка данных нашла свое применение в широком спектре областей, где критически важна скорость реакции на возникающие события. Одной из ключевых областей является реально-временная аналитика, применяемая, например, финансовыми учреждениями для мониторинга торговых операций в момент их совершения и предотвращения несанкционированных транзакций. В сфере обработки событий Интернета вещей (IoT) распределенные устройства (промышленные датчики, бытовые сенсоры и др.) генерируют непрерывные потоки телеметрических данных, а потоковые системы мониторинга применяются для контроля качества обслуживания и оперативного обнаружения сбоев. В промышленности поточковая обработка может применяться для фиксации отклонений в работе до возникновения серьезных неисправностей. В направлении маркетинга

предиктивная аналитика на основе потоковых данных позволяет прогнозировать будущие события на основе текущих наблюдений, формировать тренды, выявлять зависимости и персонализировать предложения.

Например, финансовые учреждения используют потоковые системы для мониторинга торговых операций в момент их совершения, что позволяет предотвращать несанкционированные транзакции. А телекоммуникационные компании применяют данную парадигму для контроля качества обслуживания и оперативного обнаружения сбоев в сетевой инфраструктуре.

Другим значимым направлением выступает обработка событий Интернета вещей (IoT). Распределенные устройства – от промышленных датчиков до бытовых сенсоров – генерируют непрерывные потоки телеметрических данных. В промышленности мониторинг состояния оборудования в реальном времени позволяет фиксировать малейшие отклонения в работе до возникновения серьезных неисправностей.

Предиктивная аналитика на основе потоковых данных позволяет прогнозировать будущие события на основе текущих наблюдений. В маркетинге это используется для анализа поведения пользователей и персонализации предложений, а в медицине – для непрерывного мониторинга жизненных показателей пациентов и раннего обнаружения патологических состояний [1].

Можно сделать вывод, что востребованность потоковой обработки в современном цифровом пространстве подтверждает ее универсальность как инструмента оперативного анализа и принятия решений в реальном времени.

Интерес к потоковой обработке данных как к научной и прикладной дисциплине не ослабевает на протяжении нескольких десятилетий. Однако если на ранних этапах своего развития средства потокового анализа представляли собой узкоспециализированные, чаще всего коммерческие, решения с закрытой архитектурой, то сегодня ландшафт кардинально изменился. Взрывной рост парадигмы Big Data послужил катализатором для создания общедоступных, открытых инструментов в области потоковой обработки. Современные информационные системы генерируют непрерывные и интенсивные потоки

событий [3]. Поток событий представляет собой непрерывный поток данных, который поступает в систему в режиме реального времени. Во многих случаях эти данные либо нецелесообразно хранить длительное время, либо их ценность с течением времени критически снижается. Переход от модели пакетной обработки к модели потоковой обработки спровоцировал фундаментальный сдвиг в проектировании информационных систем.

1.1.1 Основы организации систем потоковой обработки данных

Стремление к стандартизации подходов привело к появлению целого семейства взаимозаменяемых технологических фреймворков с открытым исходным кодом, многие из которых развиваются под эгидой фонда Apache Software Foundation. В качестве ярких примеров можно назвать Apache Kafka (Kafka Streams, ksqlDB), Apache Flink, Apache Storm. Amazon Kinesis [4]. Исторические пути, которые привели создателей этих продуктов к реализации именно потоковой модели, были различны. Одни системы эволюционировали из высокопроизводительных очередей сообщений, другие появились из движков для пакетной обработки данных. Это многообразие, с одной стороны, предоставляет инженерам широкий выбор платформ, но с другой – порождает проблему обоснованного подбора инструмента под конкретную задачу.

Однако, независимо от конкретной реализации, любая потоково-ориентированная система подчиняется определенной последовательности шагов, представленной на рисунке 1.



Рисунок 1 – Последовательность шагов потоковой обработке данных

Процесс работы потоковой системы начинается с захвата данных из источников, например, датчиков, сенсоров, устройств IoT, логов приложений, веб-сервисов, социальных сетей и других генераторов событий, порождающих потоки данных. Далее, по мере поступления, данные агрегируются и преобразуются в единообразный формат для более удобной обработки. Это может включать в себя фильтрацию, преобразование формата, агрегацию и другие операции, необходимые для получения информации в необходимом для обработки виде. Процесс обработки состоит из применения алгоритмов и логики, которые анализируют данные, выявляют закономерности, аномалии и тренды. После обработки данных система должна принять решение на основе полученных результатов. Это может быть автоматическое действие, например, коррекция производственного процесса при выявлении несоответствий, или предоставление информации человеку для принятия решения. В зависимости от результата обработки, система может выполнять дополнительные действия. Это может быть отправка уведомлений, запись данных в хранилище, архивирование информации или внесение изменений в рабочие процессы [1].

В качестве иллюстрации можно рассмотреть систему мониторинга оборудования на промышленном предприятии. Датчики, установленные на станках, непрерывно фиксируют параметры их работы (температуру, вибрацию, давление и т. д.). Полученные телеметрические данные в режиме реального времени направляются в конвейер потоковой обработки, образуя непрерывный поток событий о состоянии оборудования. На этапе обработки система сопоставляет текущие показатели с исторически сложившимися паттернами нормального функционирования и обнаруживает аномалии, свидетельствующие о возможных дефектах. Основываясь на этих вычислениях, система рассчитывает вероятность отказа автоматически формирует оповещения для ремонтных служб.

В приведенном сценарии парадигма обработки данных в потоках позволяет предприятию перейти от планово-предупредительной стратегии к обслуживанию по фактическую состоянию, что минимизирует внеплановые

простой дорогостоящего оборудования, сокращает затраты на ремонт и в конечном счете повышает общую эффективность производства.

Принцип потоковой обработки позволяет избавиться от недостатков пакетной обработки данных, таких как:

- неспособность своевременного реагирования в сценариях, требующих мгновенной реакции на события;
- накопление больших объемов данных и использование ресурсоемких операций;
- высокие задержки в системах, где они могут иметь критические последствия.

Но, вследствие динамичной природы данных и асинхронности процессов, потоковая обработка остается сложной задачей, требующей оптимизации алгоритмов и сложных вычислительных процессов для достижения высокой производительности. Минимизация задержек между моментом генерации данных и их обработкой требует использования специализированных методов работы с данными. Системы потоковой обработки должны быть устойчивы к отказам, высоким нагрузкам, обладать устойчивостью, высокой производительностью, а также механизмами буферизации и кэширования.

На данный момент существует ряд популярных и надежных платформ, позволяющих обрабатывать данные в реальном времени. К ним относят Apache Kafka, Apache Flink, Apache Storm, Amazon Kinesis и др.

Apache Kafka является распределенной системой обмена сообщениями между серверными приложениями в режиме реального времени. Благодаря высокой пропускной способности, масштабируемости и надежности применяется в компаниях, работающих с большими объемами данных. Написана на языках Java и Scala.

Apache Flink – это отказоустойчивая распределенная платформа обработки информации с открытым исходным кодом, поддерживающая как пакетную, так и потоковую обработку данных. Отличается высокой производительностью, низким временем задержек, имеет веб-интерфейс, гарантирует

отказоустойчивость и поддержку итерационных вычислений на основе машинного обучения и анализа графов.

Apache Storm – это Big Data фреймворк с открытым исходным кодом для распределенных потоковых вычислений в реальном времени, разработанный на языке программирования Clojure. Позволяет создавать сложные топологии обработки и обеспечивает надежность и устойчивость к сбоям.

Amazon Kinesis представляет собой платформу обработки потоковых данных в реальном времени. Это полностью управляемый сервис Amazon Web Services (AWS), что избавляет пользователей от сложных задач администрирования и сопровождения этой платформы. Среда Amazon Kinesis является надежной, безопасной и долговечной.

Выбор подходящей платформы потоковой обработки зависит от требований проекта, масштаба задачи и инструментов, с которыми предстоит работать.

1.1.2 Типология событий в парадигме потоковой обработки данных

События являются одним из важнейших классов объектов в рамках потоковой обработки данных. Это сигналы, которые указывают на изменения в состоянии объектов наблюдения. События включают информацию о возникновении аномалий, достижении пороговых значений, обновлении параметров, а также других значимых изменениях, требующих реакции.

В потоках данных события классифицируются по их временной природе и сложности структуры.

По временной природе следует различать мгновенные (точечные) события, которые происходят в конкретный, не имеющий продолжительности момент времени, и интервальные события, которые характеризуются длительностью и имеют явно выраженные начало и конец. Например, событие «превышение порога скорости» является мгновенным, в то время как событие «торможение» – интервальным. Понимание этой разницы принципиально, поскольку обработка

интервальных событий требует иных подходов, например, использование операторов временной логики Аллена.

В наиболее общем виде все множество событий можно разделить по структурной сложности на две большие группы: простые (элементарные) и сложные (комплексные).

Простое событие рассматривается как неделимая, атомарная запись, фиксирующая единичный факт в определенный момент времени, например, показание датчика температуры. Сложное событие является результатом логического или временного сопоставления нескольких простых событий и отражает некоторую значимую ситуацию или закономерность.

Более детальная классификация событий по уровню сложности представлена на рисунке 2.

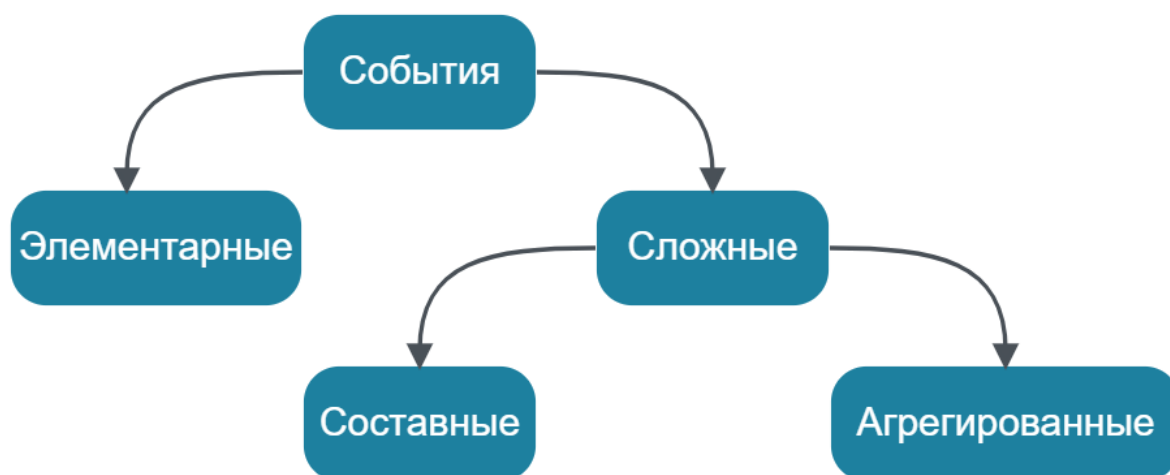


Рисунок 2 – Классификация событий

Далее будет рассмотрена классификация событий именно с точки зрения их структурной сложности, так как она является определяющей для проектирования конвейеров обработки данных.

1.1.2.1 Элементарные события

Элементарные события представляют собой базовые, атомарные единицы информации, поступающие непосредственно от внешних наблюдаемых систем.

Они фиксируют единичный факт или изменение состояния объекта мониторинга в конкретный момент времени и не подлежат дальнейшей декомпозиции в рамках обрабатывающей системы. Отдельные входные кортежи часто обозначаются именно как простые события, в отличие от композитных, формируемых на их основе.

Элементарное событие характеризуется следующим набором обязательных атрибутов:

- уникальный идентификатор события, который обеспечивает однозначную ссылку на конкретное событие в рамках всей системы;
- контекст события, который определяет набор параметров, описывающих природу события и его смысловое содержание (параметры могут быть как простых типов, так и представлять собой вложенные структуры);
- идентификатор источника события, то есть ссылка на объект мониторинга, породивший данное событие;
- тип события или классификационная метка, позволяющая маршрутизировать событие к соответствующему обработчику;
- временная метка, а именно точные дата и время возникновения события.

Структура элементарного события может быть описана через модель «заголовок» и «полезная нагрузка», где заголовок содержит метаданные (тип события, временные метки), а полезная нагрузка – данные, ассоциированные с событием. Поля полезной нагрузки определяются пользователем исходя из предметной области.

Элементарное событие представляет собой необработанную единицу данных, поступающую от датчиков, IoT устройств, веб-сервисов. Например, показание датчика температуры, установленного на двигателе автомобиля, передаваемое в электронный блок управления, является элементарным событием. Показание само по себе единично и несет в себе только один вид информации – значение текущей температуры.

Подобные события служат в качестве источников данных для последующего конструирования более сложных ситуаций, обнаружение которых сводится к применению фильтрации, агрегации и объединению потоков и таблиц элементарных событий.

1.1.2.2 Сложные составные и агрегированные события

Сложные события являются вычисляемыми событиями, которые фиксируются платформой в результате выполнения операций над элементарными и/или другими сложными событиями. В отличие от элементарных, которые лишь констатируют отдельные факты, сложные события отражают значимые ситуации или закономерности, выявленные путем анализа совокупности входящих данных.

Сложные события могут быть дополнительно классифицированы на агрегированные и составные события. Основанием для такого разделения служит способ формирования результирующего события: агрегированные события являются продуктом применения вычислительных функций к группам событий, в то время как составные события формируются путем логического или временного сопоставления нескольких независимых событий.

Агрегированные события формируются на основе агрегирующего запроса к входному потоку элементарных событий. В отличие от составных, для возникновения которых требуется совпадение определенных условий, агрегированное событие является результатом применения вычислительной функции (суммирование, подсчет количества, вычисление среднего значения, поиск максимума или минимума). Такое событие легко обнаружить, не используя для этого сложных конструкций и специфических отношений.

Примером агрегированного события может служить ситуация, когда необходимо вычислить суммарное количество превышений значения датчика давления в течение часа. В качестве входных данных используется поток элементарных событий с устройства. Подсчет количества пришедших

сообщений за установленный промежуток времени представляет собой агрегированное событие.

Составное событие представляет собой комбинацию элементарных и/или сложных событий с возможным использованием временных ограничений и логических операций. Например, обнаружение ситуации, когда вскоре после того, как светофор переключился на красный, автомобильный номер фиксируется камерой видеонаблюдения, является результатом сопоставления двух независимых событий с наложением временного ограничения на их взаимное расположение.

Структура сложного составного события в виде дерева представлена на рисунке 3.

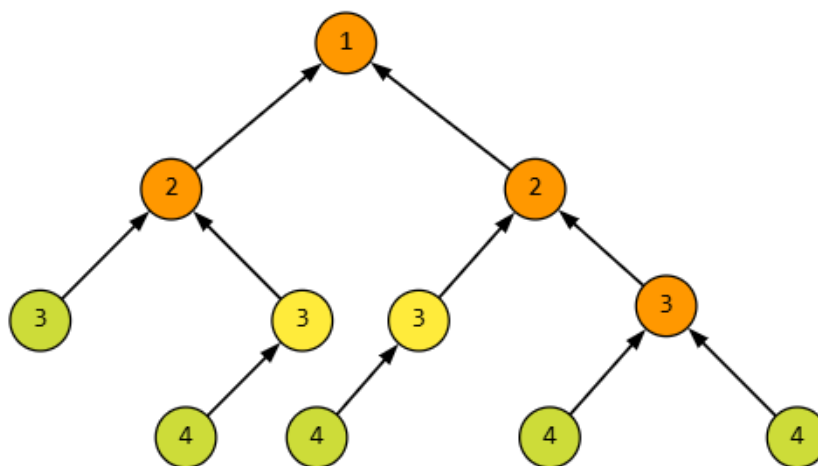


Рисунок 3 – Схематичное представление дерева сложного составного события

Оранжевым цветом отмечены сложные составные события, имеющие два входных источника. Желтым цветом отмечены сложные агрегированные события, имеющие ровно по одному входному потоку в качестве источника данных. Зеленым – элементарные события, по сути являющимися передатчиками сырых данных от непосредственных источников данных системы (генераторов, датчиков и пр.). Сложное составное событие представляет из себя последовательность подобных объединений.

Для формального описания закономерностей, приводящих к формированию составных событий, используются специализированные

операции. Набор операций, применяемых для описания отношений между событиями для составления сложных составных событий представлен в таблице 1.

Таблица 1 – Операции отношений между событиями в составе составного события

Операция	Формальная интерпретация	Описание
AND(E1,E2)	$E1.st \wedge E2.et \vee E1.et \wedge E2.st \vee st \leq et$	Событие E истинно, если истинны E1 и E2
OR(E1,E2)	$E1 \vee E2$	Событие E истинно, если истинно E1 (и) или E2
NOT(E1)	$\neg E1$	Событие E истинно, если E1 ложно
SEQ(E1,E2)	$E2.et \wedge E1.st \wedge st < et$	Событие E равно последовательности событий E1, E2 и E3
BEFORE(E1,E2)	$E1.st < E2.st \wedge E1.et < E2.st$	Событие E ₁ раньше события E ₂
AFTER(E1,E2)	$E2.st < E1.st \wedge E2.et < E1.st$	Событие E ₁ после события E ₂
MEETS(E1,E2)	$E1.st < E2.st \wedge E1.et = E2.st$	Событие E ₁ встречается событием E ₂
METBY(E1,E2)	$E2.st < E1.st \wedge E2.et < E1.st$	Событие E ₁ встречено событием E ₂
OVERLAPS(E1,E2)	$E1.st < E2.st < E1.et < E2.et$	Событие E ₁ перекрывается событием E ₂
OVERLAPPEDBY(E1,E2)	$E2.st < E1.st < E2.et < E1.et$	Событие E ₁ перекрыто событием E ₂
STARTS(E1,E2)	$E1.st = E2.st < E1.et < E2.et$	Событие E ₁ начинается с событием E ₂ , но заканчивается раньше
STARTEDBY(E1,E2)	$E2.st = E1.st < E2.et < E1.et$	Событие E ₁ начинается с событием E ₂ , но заканчивается позже
DURING(E1,E2)	$E2.st < E1.st < E1.et < E2.et$	Событие E ₁ во время события E ₂
CONTAINS(E1,E2)	$E1.st < E2.st < E2.et < E1.et$	Событие E ₁ содержит событие E ₂
FINISHES(E1,E2)	$E2.st < E1.st < E1.et = E2.et$	Событие E ₁ заканчивает событие E ₂
FINISHEDBY(E1,E2)	$E1.st < E2.st < E1.et = E2.et$	Событие E ₁ закончено событием E ₂
EQUALS(E1,E2)	$E1.st = E2.st \wedge E1.et = E2.et$	Событие E ₁ равно событию E ₂

Описанные бинарные операции (AND, OR, BEFORE, OVERLAPS, DURING и др.) задают базовые примитивы. Каждый из них задает строгое интервальное соотношение между началом и окончанием двух событий [5].

Последовательное выполнение событий без пересечения во времени описывается оператором SEQ (от англ. sequence). Формально он означает, что каждое следующее событие начинается строго после окончания предыдущего. Это позволяет детектировать жесткие цепочки действий.

Примером может служить событие «Сброс пароля злоумышленником». Его можно представить в виде следующего выражения:

$$E = \text{SEQ}(E1, E2, E3),$$

где E1 – это событие «Неудачная попытка входа»;

E2 – событие «Успешный вход»;

E3 – «Изменение пароля».

Событие E фиксируется, когда три указанных действия совершаются в строго указанном порядке. Такой паттерн часто свидетельствует о краже учетных данных злоумышленником.

Когда важен не только порядок, но и временные рамки, базовое интервальное отношение может дополняться другими операциями. Событие «Компрометация карты сотрудника» можно выразить следующей комбинацией операций:

$$E = BEFORE(E1, E2) [0, 6] \wedge E1.cardId = E2.cardId,$$

где E1 – событие «Проход на КПП А»;

E2 – событие «Проход на КПП Б»;

интервал $[0, 6]$ – временное окно от 0 до 6 минут;

E1.cardId, E2.cardId – идентификаторы карты.

Событие E считается истинным, если проход на КПП А полностью завершается до начала прохода на КПП Б (BEFORE), при этом разрыв между окончанием E1 и началом E2 составляет не более шести минут. Кроме того, оба прохода должны быть выполнены одной и той же картой, что определяется по ее идентификатору. Если в реальной физической обстановке переход между двумя КПП занимает примерно шесть минут, а карта используется быстрее, это указывает на ее копирование или несанкционированное использование.

Комбинация из механизмов последовательных операторов, временных окон и атрибутивных условий позволяют обнаруживать сложные значимые ситуации.

Таким образом, агрегированные и составные события, являясь подклассами сложных событий, различаются прежде всего природой операции, приводящей к их возникновению: для агрегированных событий это вычислительная обработка данных от их источника, а для составных – логическая композиция нескольких событий в новое, отражающее более высокоуровневую ситуацию в предметной области. Обнаружение сложного

составного события является более сложной задачей, требующей использования специфических подходов.

Четкая классификация событий по структурной сложности и способу формирования служит концептуальной основой для проектирования систем потоковой обработки, область применения которых охватывает широкий спектр прикладных задач.

1.1.3 Обнаружение сложных составных событий в системах потоковой обработки данных

Потоковая обработка данных покрывает большинство задач анализа в реальном времени. Однако существует класс задач, требующий обнаружения не просто отдельных фактов, а значимых последовательностей событий, распределенных во времени. Для решения таких задач применяется технология комплексной обработки событий (Complex Event Processing, CEP), которая играет значимую роль в современных событийно-ориентированных архитектурах [6].

Стандартная потоковая обработка отвечает на вопросы вроде «каково среднее значение температуры за последние пять минут?» или «сколько событий входа зафиксировано для данного пользователя за час?». Это вычисления, выполняемые либо над отдельными событиями, либо над их агрегатами в рамках временных окон. Комплексная обработка событий решает качественно иную задачу: произошла ли заданная цепочка событий в определенном порядке и в пределах заданного временного интервала.

Примером может служить вопрос: «было ли событие А, за которым в течение тридцати секунд последовало событие В, но при этом в предшествующие десять минут не было зафиксировано событие С?». Этот подход часто сравнивают с регулярными выражениями, применяемыми не к тексту, а к потокам событий: шаблон определяется один раз, после чего система непрерывно оценивает каждое входящее событие на соответствие всем активным шаблонам и выдает результат в момент полного совпадения.

В CEP данные «прогоняются» через хранимые запросы: события, не соответствующие ни одному активному шаблону, отбрасываются немедленно, благодаря чему система сохраняет высокую эффективность даже при больших объемах входного потока.

Одной из наиболее ценных возможностей CEP является обнаружение событий, которые не произошли. В производственных системах критически важно зафиксировать отсутствие ожидаемого сигнала. Примерами могут служить: датчик, обязанный передавать показания с заданной периодичностью, перестал выходить на связь; подтверждение доставки не поступило в установленный срок; этап бизнес-процесса не был завершен в нормативное время. В каждом из этих случаев требуется немедленное оповещение. Для обнаружения отсутствующих событий на практике часто применяется операция LEFT JOIN в потоковом SQL, позволяющая выявить случаи, когда событие А зафиксировано, а ожидаемое событие В в пределах заданного окна так и не наступило.

Одна из ведущих платформ потоковой обработки, экосистема Apache Kafka, через движок ksqlDB позволяет описывать с помощью потокового языка запросов корреляции между событиями, объединяя несколько потоков, а также выявлять значимые последовательности событий, опираясь на механизмы обработки с сохранением промежуточного состояния [7].

Комплексная обработка событий востребована во многих отраслях. В финансовом секторе CEP применяется для обнаружения мошеннических цепочек: использование карты в Берлине, затем через две минуты попытка бесконтактной оплаты в Лондоне и сразу после этого дорогостоящая транзакция онлайн — такая последовательность физически невозможна и требует немедленной блокировки. В промышленности CEP позволяет выявлять предаварийные сигнатуры оборудования: постепенный рост температуры, за которым следует увеличение вибрации и отклонение давления — комбинация, которая при обнаружении в заданном временном окне запускает упреждающее техническое обслуживание. В логистике и управлении цепочками поставок CEP

контролирует последовательность этапов бизнес-процесса, фиксируя пропуск шагов или нарушения регламентных сроков.

Еще одним подходом к обнаружению сложных составных событий является интервальная алгебра Джеймса Аллена, предложенная им в 1983 году. Данное исчисление определяет тринадцать бинарных отношений, которые могут существовать между двумя произвольными временными интервалами: отношения следования (before, after), пересечения (overlaps), включения (during, contains), совпадения границ (meets, starts, finishes) и другие [8]. Интервальная алгебра позволяет описывать событийные ситуации в логических формулах, оперирующих отношениями между целыми интервалами, на протяжении которых происходят наблюдаемые события. Это дает возможность зафиксировать не просто факт наступления события В после события А, а выявить ситуацию, когда, например, интервал события В полностью совпадает с началом или концом интервала события А.

Применение алгебры Аллена демонстрирует лучшую масштабируемость по сравнению с СЕР, что делает его применимым для систем, обрабатывающих высокоинтенсивные потоки в реальном времени. Технология комплексной обработки событий расширяет возможности потоковой аналитики, позволяя перейти от отслеживания отдельных показателей к выявлению сложных, распределенных во времени ситуаций, имеющих прикладное значение для бизнеса и промышленности. Понимание подходов комплексной обработки событий и интервальной алгебры Аллена является необходимым фундаментом для проектирования программных средств администрирования и обнаружения событий, разработке которых посвящена практическая часть настоящей работы.

1.2 Сравнительный анализ современных решений для обнаружения событий при потоковой обработке данных

1.2.1 Обоснование сравнительного анализа

Сравнительный анализ современных программных средств, ориентированных на администрирование и обнаружение событий в системах

поточковой обработки данных, необходим для выбора оптимального набора функций. Рынок предлагает множество инструментов для работы с Apache Kafka, являющимся стандартом потоковой передачи данных.

Учитывая, что разрабатываемая система предназначена для управления потоками, конструирования событий и их отслеживания, важно сравнить ее потенциальные возможности с существующими открытыми решениями, предоставляющими схожий функционал.

Целями проведения сравнительного анализа являются:

- выявление функциональных достоинств каждой из рассматриваемых систем;
- фиксация их ограничений и недостатков;
- определение характерного базового набора возможностей, присущего современным средствам администрирования и обнаружения событий в системах потоковой обработки.

Главной задачей анализа выступает формирование перечня обязательных функций, которые должны быть реализованы в разрабатываемой системе, а также уникальных возможностей, способных повысить его конкурентоспособность.

Для анализа были отобраны четыре программных продукта, каждый из которых по-своему решает задачи администрирования Kafka и работы с событиями:

- AKHQ – open-source инструмент для управления экосистемой Kafka с широкими возможностями администрирования;
- Redpanda Console – современный веб-интерфейс для просмотра сообщений и отладки потоков данных с удобной визуализацией;
- Nussknacker – low-code среда для визуального построения сценариев потоковой обработки и обнаружения событий поверх Apache Kafka и Flink;
- IBM Event Automation – корпоративная платформа для событийно-ориентированной обработки с возможностью визуального описания правил обнаружения событий без написания кода.

Подробная информация о выбранных для анализа продуктах представлена в таблице 2.

Таблица 2 – Информация о аналоговых системах

Система	Доступность	Наличие веб-интерфейса	Фирма производитель
AKHQ	Бесплатная (Apache 2.0)	+	Сообщество, tchiotludo
Redpanda Data	Бесплатная (BSL), Enterprise-функции платные	+	Redpanda Data
Nussknacker	Бесплатная (Open Source)	+	Сообщество Nussknacker
IBM Event Automation	Платная, есть пробная версия	+	IBM

Таким образом, для дальнейшего анализа выбраны решения, охватывающие спектр от чисто административных инструментов до полноценных low-code платформ для конструирования и обнаружения событий, что позволяет провести многомерное сравнение с разрабатываемой системой.

1.2.2 Обзор программных платформ для анализа

AKHQ (ранее известный как KafkaHQ) – это зрелый open-source инструмент для управления экосистемой Apache Kafka, построенный на фреймворке Micronaut [9].

Он предоставляет широкие возможности администрирования через веб-интерфейс: управление топиками (создание, редактирование, удаление), мониторинг групп потребителей (consumer groups) и их смещений (offsets), работу со Schema Registry и Kafka Connect. AKHQ поддерживает подключение к нескольким кластерам одновременно и предоставляет базовые средства разграничения доступа (RBAC) с аутентификацией через LDAP, OAuth2/OIDC и GitHub SSO.

Конфигурация управляется через YAML-файлы, что упрощает развертывание по модели GitOps. Среди ограничений отмечается отсутствие встроенных средств маскирования конфиденциальных данных и развитых механизмов аудита действий пользователя.

AKHQ также не имеет средств визуального конструирования событий, являясь прежде всего инструментом администрирования.

Интерфейс платформы представлен на рисунке 4.

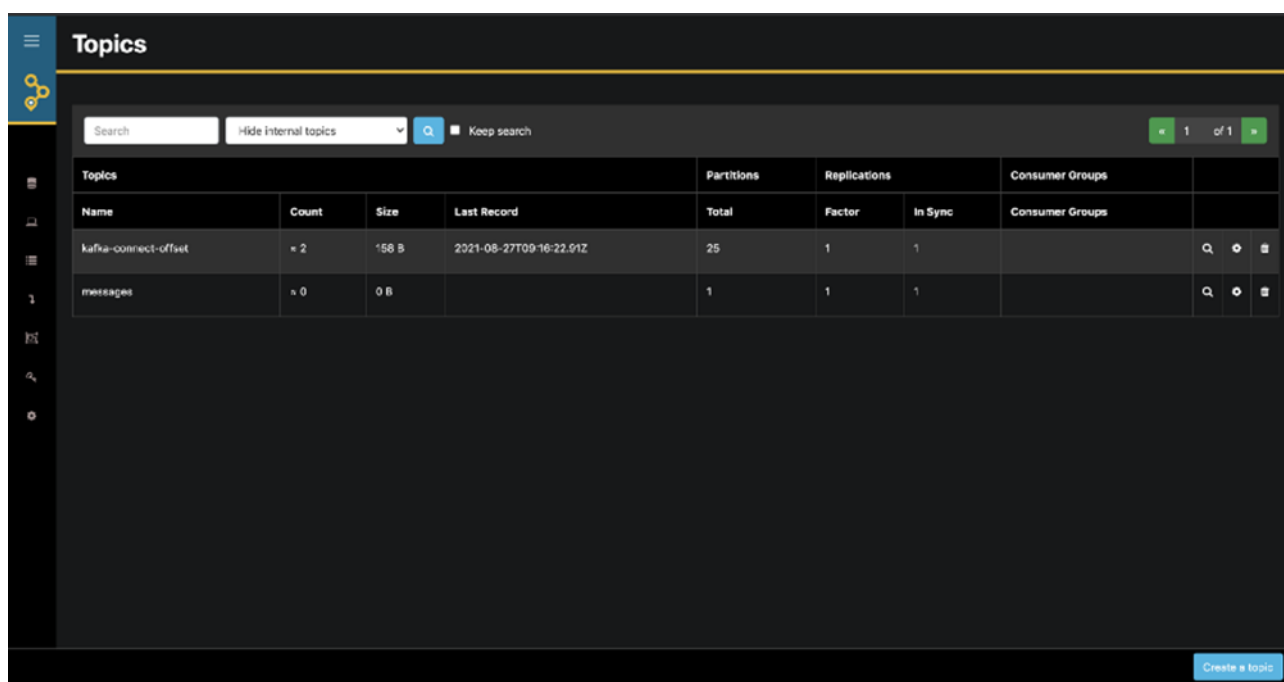


Рисунок 4 – Интерфейс веб-приложения AKHQ

Redpanda Console – это современный веб-интерфейс для управления кластерами Apache Kafka, изначально разработанный как проект Kowl, а впоследствии принятый компанией Redpanda Data [10]. Инструмент написан на языке Go, что обеспечивает низкое потребление памяти и быстрый запуск. Ключевые возможности включают:

- развитый просмотрщик сообщений с автоматической десериализацией форматов Avro, Protobuf, MessagePack и JSON;
- управление топиками и группами потребителей;
- маскирование конфиденциальных данных в Enterprise-версии;
- отладку сообщений с возможностью «перемещения во времени» (time-travel debugging).

Redpanda Console предоставляет удобные средства мониторинга состояния кластера и визуализации потоков данных. Однако важно отметить, что функции корпоративного уровня (SSO, RBAC, data masking) доступны только в платной Redpanda Enterprise-версии, а бесплатная версия ограничена базовым просмотром сообщений. Средств визуального конструирования событий или CEP-правил инструмент не предоставляет.

Интерфейс платформы показан на рисунке 5.

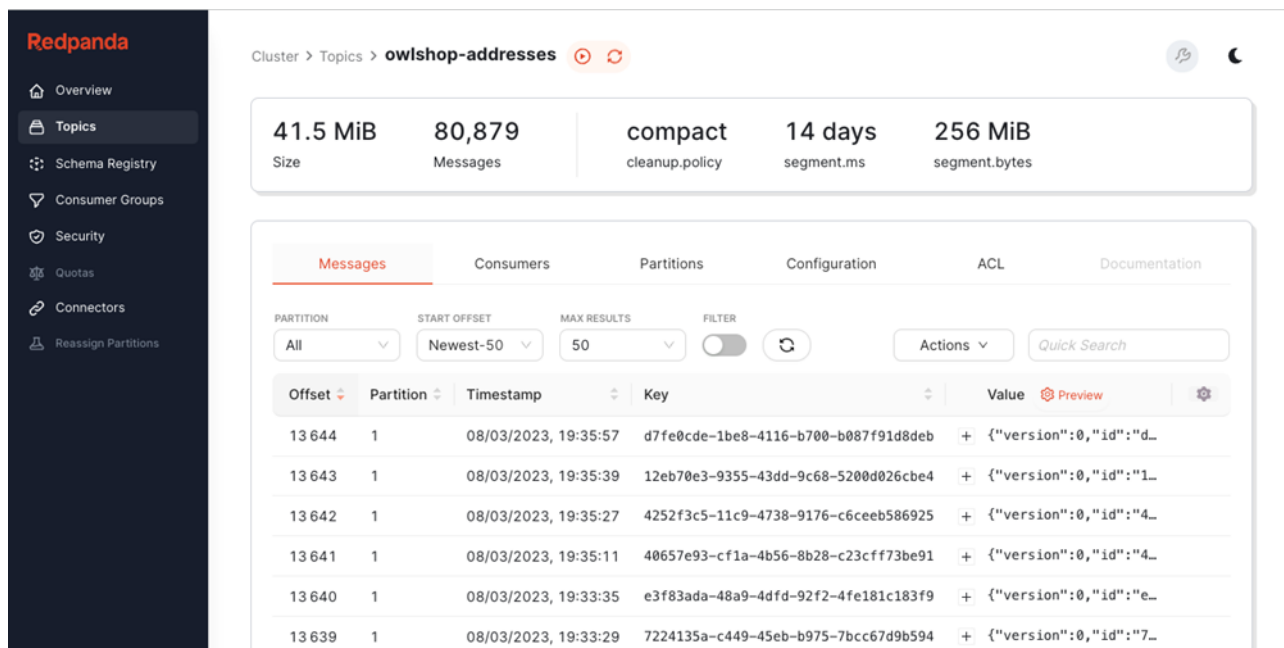


Рисунок 5 – Интерфейс платформы Redpanda Console

Nussknacker – это low-code среда для визуального построения, запуска и мониторинга сценариев потоковой обработки данных в реальном времени. Инструмент позиционируется как мост между domain-экспертами и потоковыми технологиями, позволяя описывать алгоритмы принятия решений без программирования [11].

Nussknacker работает поверх Apache Kafka (как источника и приемника событий) и Apache Flink (как вычислительного движка). Пользователь строит сценарий обработки в визуальном редакторе, перетаскивая блоки фильтрации, маршрутизации, обогащения данных из внешних источников (SQL, OpenAPI), агрегации во временных окнах и применения ML-моделей. Наиболее важной для данной работы возможностью является визуальный редактор CEP, позволяющий описывать сложные шаблоны событий (последовательности, временные ограничения, отсутствие событий) без написания кода. Nussknacker обеспечивает мониторинг выполнения сценариев с возможностью просмотра событий на каждом шаге обработки, встроенное тестирование с подстановкой mock-данных, one-click deploy и версионирование.

Интерфейс представлен на рисунке 6.

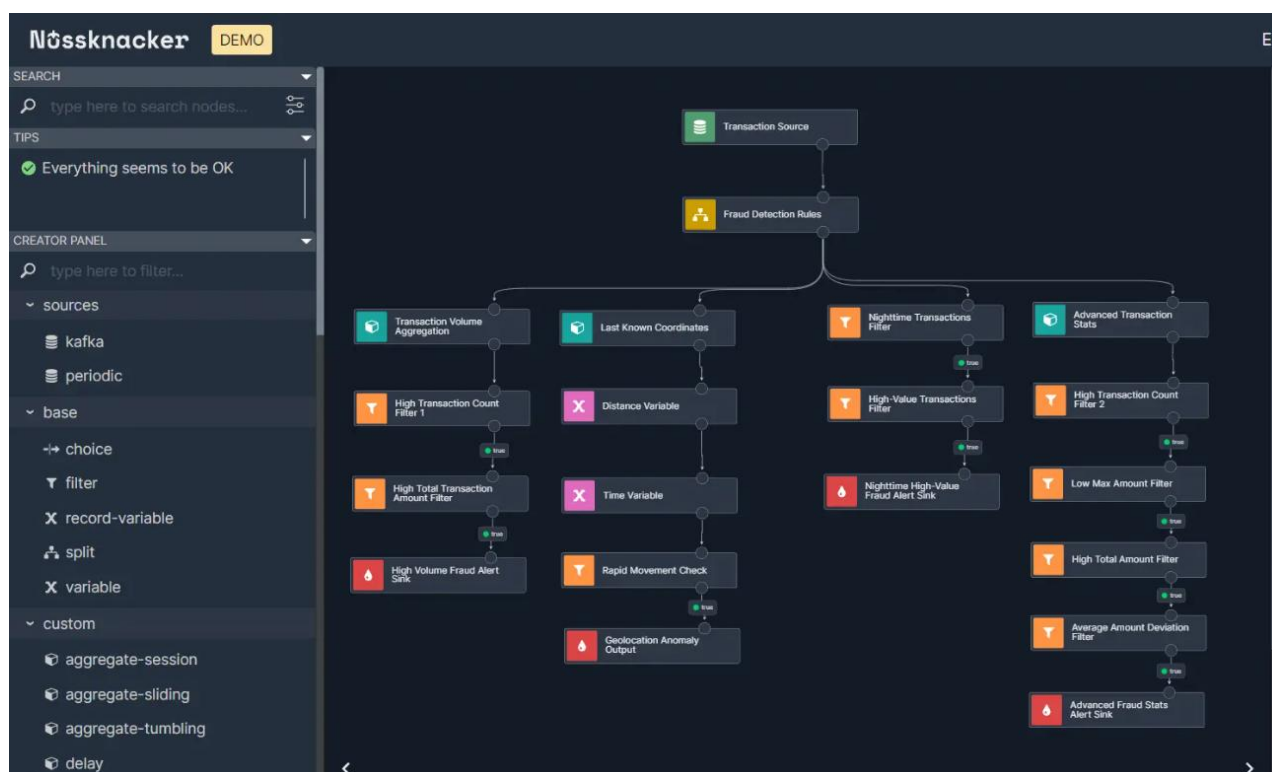


Рисунок 6 – Интерфейс Nussknacker

IBM Event Automation – это композитная корпоративная платформа для событийно-ориентированной обработки, включающая три основных компонента: Event Streams (на базе Apache Kafka), Event Endpoint Management (каталог событий и управление доступом) и Event Processing (средства потоковой аналитики) [12].

Особый интерес представляет компонент Event Processing, который позволяет пользователям создавать потоки обработки событий в визуальном редакторе без написания кода. Через drag-and-drop интерфейс можно соединять источники событий Kafka, выполнять фильтрацию, обогащение и, что наиболее ценно для задачи обнаружения событий, определять правила корреляции событий во временных окнах с учетом порядка их следования.

Платформа автоматически обрабатывает ситуации с запаздывающими или неупорядоченными событиями, позволяя указывать надежный источник времени и допустимую задержку ожидания. IBM Event Automation предоставляет развитые средства мониторинга выполнения потоков обработки, а также каталог событий для их обнаружения и повторного использования.

Интерфейс показан на рисунке 7.

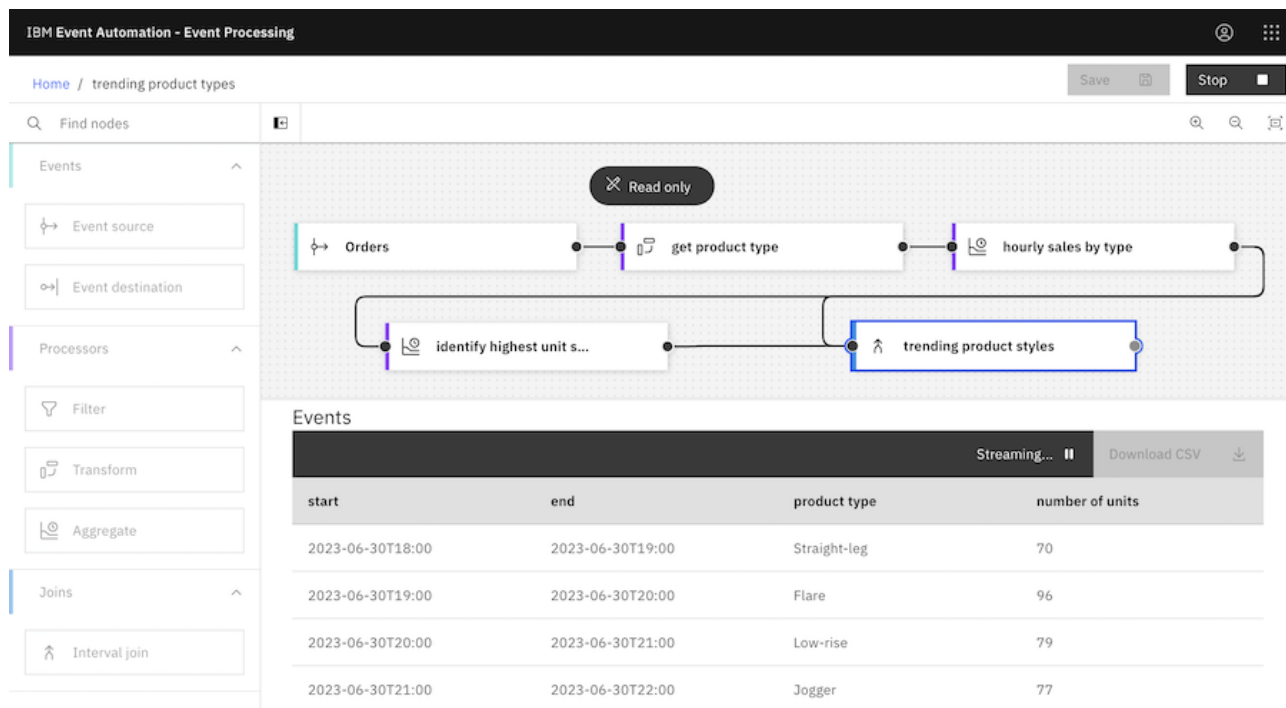


Рисунок 7 – Интерфейс IBM Event Automation

IBM Event Automation распространяется по коммерческой лицензии. Стоимость IBM Event Automation варьируется в зависимости от модели развертывания и объема использования, с начальными ценами от \$3,396 за VPC (Virtual Processor Core) на AWS Marketplace [13]. Итоговая стоимость внедрения определяется индивидуально под масштабы проекта, конфигурацию, длительность контракта и другие индивидуальные условия.

Для многофакторного сравнения были выделены следующие критерии, соответствующие как базовым, так и уникальным задачам разрабатываемой системы:

- управление топиками и потоками, а именно создание, редактирование и удаление сущностей для передачи данных в Kafka;
- просмотр сообщений в реальном времени, реализующий возможность отслеживать данные непосредственно в момент их поступления;
- визуальное конструирование событий (Low-code CEP) с помощью встроенного графического редактора для описания правил обнаружения событий без написания кода;

- интеграция с внешними системами;
- визуализация топологии обработки при помощи графического представления структуры потоковых конвейеров и связей между компонентами.

Результаты анализа выделенных программных средств представлены в таблице 3.

Таблица 3 – Результирующая таблица анализа систем по критериям

Функции	Система			
	AKHQ	Redpanda Console	Nussknacker	IBM Event Automation
Управление топиками и потоками	+	+	-	+
Просмотр сообщений в реальном времени	+	+	+	+
Визуальное конструирование событий (Low-code CEP)	-	-	+	+
Визуализация топологии обработки	+	+	+	+
Управление доступом (RBAC)	Базовый	Только Enterprise	-	+
Интеграция с внешними системами	+	+	+	+

На основании результирующей таблицы можно наглядно оценить сильные и слабые стороны рассмотренных систем в контексте задач администрирования и обнаружения событий.

1.2.3 Результаты сравнительного анализа и выводы

В результате анализа AKHQ и Redpanda Console продемонстрировали покрытие базовых административных функций: управление топиками, просмотр сообщений в реальном времени, мониторинг состояния кластера и визуализация топологии.

Оба инструмента подходят в качестве «окна в Kafka» для разработчиков и администраторов. Однако ни один из них не предоставляет средств конструирования правил обнаружения событий. Их функциональность ограничивается наблюдением за данными, но не их обработкой. Redpanda Console предлагает более современный интерфейс и встроенные средства маскирования данных, но лишь в коммерческой версии, в то время как AKHQ компенсирует это более доступной поддержкой мультикластерных конфигураций.

Nussknacker и IBM Event Automation – это low-code платформы, которые выходят за рамки администрирования и предлагают визуальные средства построения логики обработки событий. Их ключевое преимущество заключается в возможности описывать правила обнаружения сложных событийных паттернов без написания программного кода. Nussknacker, будучи открытым продуктом, ориентирован на работу поверх Apache Kafka и Flink и предлагает визуальный редактор с живым предпросмотром данных на каждом шаге сценария.

IBM Event Automation, в свою очередь, представляет собой корпоративное решение с развитым каталогом событий, управлением доступом и встроенной поддержкой обработки неупорядоченных событий. Из двух low-code платформ только IBM Event Automation предоставляет средства управления топиками, объединяя их в одном интерфейсе с визуальным конструктором событий. Nussknacker не обладает подобным функционалом.

Однако IBM Event Automation имеет ряд существенных ограничений для российских пользователей. Прежде всего, это высокая стоимость лицензирования, делающая решение экономически недоступным для многих отечественных компаний. Кроме того, документация и справочные материалы представлены исключительно на английском языке, а официальная русскоязычная поддержка отсутствует.

Исходя из проведенного анализа, в разрабатываемой системе необходимо поддержать следующий базовый функционал:

- просмотр и настройка объектов системы;
- возможность конструирования событий;
- мониторинг обнаружения событий в реальном времени.

Кроме того, для обеспечения уникальности и конкурентоспособности, в систему необходимо включить дополнительные возможности:

- предоставление удобного интерфейса для создания сложных цепочек из потоков и таблиц для конструирования агрегированных и сложных событий на основе ksqlDB;

– возможность вывода аналитики по временным периодам для выбранных событий, зарегистрированных в системе.

Таким образом, разрабатываемая система займет нишу, объединяющую административные возможности AKNQ и Redpanda Console с событийно-ориентированной логикой Nussknacker и IBM Event Automation, предоставляя пользователю единый интерфейс для полного цикла работы с событиями в потоковой обработке данных. Ценность настоящей разработки заключается в идее объединения системы потоковой обработки данных и систему обнаружения сложных событий в единое программное обеспечение.

В отличие от рассмотренных зарубежных аналогов, разрабатываемое решение будет характеризоваться низким порогом входа за счет русскоязычного интерфейса и документации, что делает его доступным для широкого круга отечественных специалистов. Ориентация на стек открытых технологий и отсутствие лицензионных отчислений обеспечат невысокую стоимость как разработки, так и последующего обслуживания системы. Кроме того, независимость от санкционных ограничений гарантирует стабильность эксплуатации и возможность адаптации под изменяющиеся требования российских заказчиков.

1.3 Планирование процесса разработки программных средств

1.3.1 Цели разработки и назначение программных средств

Целью разработки является набор программных средств для администрирования, мониторинга и обнаружения событий в системе потоковой обработки данных на платформе Apache Kafka. Разрабатываемые программные средства будут включать в себя несколько модулей, обладающих разнонаправленным функционалом.

Архитектура программного решения предусматривает четкое разделение функций между двумя основными модулями, ориентированными на разные категории пользователей и непересекающиеся задачи. Модуль администрирования будет выполнять роль инструмента конфигурирования

инфраструктуры потоковой обработки. Через него осуществляется работа с источниками данных, топиками, конструирование событий определенного типа и прочие операции, направленные на работу с внутренней структурой системы.

Модуль диспетчеризации данных предназначен для наблюдения за обнаружением событий в реальном времени без прямого воздействия на конфигурацию кластера Apache Kafka.

Разделение этих модулей является важным архитектурным решением, которое изолирует административные задачи от задач диспетчеризации, исключая риск случайного изменения структуры потоков со стороны оператора.

Разрабатываемые программные средства для обнаружения и мониторинга событий в системе потоковой обработки данных должны обеспечивать пользователя набором удобных инструментов для администрирования объектов Kafka и управления обнаружением событий в реальном времени.

Основные задачи разработки:

- реализация модуля администрирования, обеспечивающего конфигурирование потоковой инфраструктуры: просмотр доступных в системе топиков, активация передачи данных от источников данных в Kafka, конструирование сложных событий, настройка их параметров и активация их обнаружения;
- реализация модуля диспетчеризации, работающего на основе Kafka API и предоставляющего оператору возможность в реальном времени наблюдать за потоками событий, отслеживать динамику поступающих сообщений без внесения изменений в структуру системы;
- реализация сервера RESTful API для централизованного управления системой и осуществления бизнес-логики;
- реализация сервера ksqlDB, позволяющего осуществлять потоковую обработку данных.

Кроме того, после окончания основного процесса разработки и публикации сервиса требуется поддерживать его функционал, проверять на неисправности, проводить анализ работы с помощью сбора статистики

использования, а также внедрять новый функционал при необходимости для упрощения работы администраторов и операторов при выходе новых версий протокола Kafka.

Ключевым преимуществом разрабатываемого решения, по сравнению с существующими аналогами, является его ориентированность на объединение потоковой обработки и обнаружения сложных событий. При этом разрабатываемые средства будут характеризоваться простотой использования, четким разделением ролей и минимальным порогом входа. Модульная архитектура desktop-приложений позволяет обеспечить высокую скорость отклика интерфейса и дает возможность эксплуатации системы персоналом без глубоких знаний низкоуровневых команд администратора брокера. Таким образом, разработка позволяет администратору гибко настраивать потоковую инфраструктуру, а оператору – непрерывно отслеживать события, не опасаясь нарушить работоспособность системы случайным изменением конфигурации, что снизит эксплуатационные издержки и минимизирует риски человеческого фактора.

1.3.2. Классы пользователей программных средств

Проектирование любой информационной системы начинается с определения круга ее пользователей и их функциональных ролей.

Для разрабатываемой системы потоковой обработки данных предполагается два основных класса пользователей, различающихся по уровню компетенций, спектру решаемых задач и характеру взаимодействия с платформой: администратор данных (событий) и оператор данных (событий). Такое разделение соответствует общепринятой практике построения многопользовательских систем, где задачи конфигурирования инфраструктуры отделены от задач повседневной эксплуатации.

Администратор данных (событий) – это технический специалист, отвечающий за управление инфраструктурой потоковой обработки и конструирование новых событий.

Разрабатываемая система должна предоставлять администратору данных следующий набор возможностей:

- просмотр подробной информации о топиках (логических каналах, по которым передаются потоки событий) и связанных с ними потоков, обновление списка топиков;
- активизация и деактивация передачи данных от внешних источников, внесенных в базу данных и имеющих возможность передавать данные в потоки Kafka;
- конструирование сложных агрегированных и составных событий при помощи гибкого конструктора, позволяющего отображать иерархию дерева потоков и таблиц;
- инициация начала отлова нового созданного события в Kafka.

Оператор данных (событий) – это специалист предметной области, для которого модуль диспетчеризации является инструментом для мониторинга системы. Оператор сможет в реальном времени наблюдать за отловом событий, состоянием потоков и получать аналитику на основе приходящих данных.

Разрабатываемая система проектируется таким образом, чтобы минимизировать порог входа для ее пользователей. Русскоязычный интерфейс, интуитивно понятный визуальный конструктор сложных событий и встроенная документация позволяют пользователям сосредоточиться на управлении потоковой обработкой, не отвлекаясь на технические детали реализации Kafka. В то же время администратор получает полный контроль над инфраструктурой через единый интерфейс, что избавляет его от необходимости использования нескольких разрозненных инструментов.

1.3.3 План-график разработки и планирование бюджета

Формирование плана выполнения работ, отражающего порядок действий и распределение необходимых ресурсов, составляет сущность планирования процесса разработки. Его главной целью выступает определение целей и задач проекта, а также путей их реализации.

Жизненный цикл программного обеспечения представляет собой совокупность этапов, через которые проходит программный продукт от момента формулирования требований до его полного вывода из эксплуатации.

В современной практике разработки информационных систем выделяют несколько основных моделей жизненного цикла программного обеспечения:

- каскадная модель;
- V-образная модель;
- модель прототипирования;
- инкрементная модель;
- спиральная модель.

В каскадной модели этапы проходят однократно и в жесткой линейной последовательности: каждый следующий начинается только после того, как полностью завершен предыдущий и подготовлен соответствующий пакет документации. Такой процесс несложен в управлении, однако плохо адаптирован к постоянно меняющимся требованиям.

Развитием этого подхода стала V-образная модель. Она сохраняет поэтапную структуру, но выводит на первый план тестирование и верификацию, выстраивая непосредственные связи между аналитическими стадиями и отвечающими им стадиями испытаний.

При прототипировании в основу закладывается создание экспериментальной версии системы, а требования к итоговому продукту итеративно уточняются прямо по ходу доработок этого прототипа. Благодаря такому подходу заказчик знакомится с работающим образцом и может вовремя внести коррективы в постановку задачи.

Инкрементная модель строится на идее планомерного улучшения продукта, так как функциональность системы расширяется шаг за шагом, последовательными приращениями. Каждая новая версия обогащает предыдущую определенным набором возможностей до тех пор, пока не будут реализованы все требования. Каждый инкремент проходит через полный цикл работ – от проектирования до эксплуатации.

Спиральная модель относится к итерационным моделям, где на каждом витке спирали прорабатываются и конкретизируются все аспекты системы, а особое внимание уделяется анализу рисков и созданию прототипов для проверки принимаемых решений [14].

Выбор конкретной модели жизненного цикла должен опираться на анализ комплекса характеристик: требований к системе, особенностей команды разработчиков, типа проекта и сопутствующих рисков.

Для данного проекта была выбрана инкрементальная (итеративная) модель жизненного цикла. Прежде всего, функциональность разрабатываемой системы естественным образом сегментируется на несколько модулей: программные средства обнаружения событий, модуль администрирования, а также модуль диспетчеризации для отслеживания событий в реальном времени. Такая структура соответствует идее инкрементной модели, так как каждый модуль может быть реализован в виде отдельного инкремента, последовательно проходящего через все стадии от проектирования до тестирования.

Общие требования к системе были определены и зафиксированы на начальном этапе, что свойственно традиционному подходу. Однако детализация требований к каждому последующему инкременту может уточняться по результатам реализации и демонстрации предыдущего. Инкрементная модель позволяет продемонстрировать работающий прототип базовой функциональности уже на ранних стадиях, не дожидаясь полного завершения проекта, и при необходимости скорректировать требования к последующим инкрементам.

Таким образом, инкрементальная модель обеспечивает разумный баланс между предсказуемостью традиционного подхода и гибкостью, необходимой для последовательного наращивания функциональности системы.

Для демонстрации плана разработки подходит диаграмма Ганта, отображающая длительность и сроки выполнения этапов разработки на шкале времени, а также требуемые для их реализации ресурсы. В качестве инструмента управления проектом была выбрана программа Microsoft Project.

В таблице 4 приведены результаты определения задач проекта, их тип, длительность их выполнения и задачи, предшествующие им.

Таблица 4 – Задачи проекта

№	Задача	Тип	Длительность	Предшественники
1	2	3	4	5
1	Начало проекта	Веха	0	
2	Анализ предметной области и планирование процесса разработки программных средств	Фаза	7	
3	Анализ основ организации систем потоковой обработки данных	Задача	1	1
4	Анализ типологии событий	Задача	1	3
5	Анализ процесса обнаружения сложных составных событий	Задача	2	4
6	Сравнительный анализ аналогов	Задача	2	5
7	Постановка цели разработки	Задача	1	6
8	Анализ предметной области и планирование процесса разработки программных средств завершены	Веха	0	7
9	Анализ требований и постановка задачи на разработку программных средств	Фаза	7	
10	Анализ функциональных требований	Задача	1	8
11	Анализ нефункциональных требований	Задача	1	10
12	Анализ требований к пользовательскому интерфейсу панелей администрирования и диспетчеризации событий	Задача	1	11
13	Анализ требования к аппаратному обеспечению	Задача	1	12
14	Анализ требований к системному программному обеспечению	Задача	1	13
15	Анализ требований к средствам разработки	Задача	1	14
16	Постановка задачи на разработку программных средств	Задача	1	15
17	Анализ требований и постановка задачи на разработку программных средств завершены	Веха	0	16
18	Разработка ПС администрирования и обнаружения событий	Фаза	29	
19	Разработка архитектуры развертывания программных средств	Задача	4	17
20	Реализация развертывания событий в системе потоковой обработки данных Apache Kafka декларативными средствами языка kSQL	Задача	6	18
21	Разработка серверных программных средств администрирования и развертывания событий в системе потоковой обработки данных	Задача	7	19
22	Разработка клиентского приложения администрирования событий	Задача	6	20
23	Разработка клиентского приложения диспетчеризации событий	Задача	6	21
24	Разработка ПС администрирования и обнаружения событий завершены	Веха	0	22
1	2	3	4	5
25	Контроль качества программных средств администрирования и обнаружения событий	Фаза	10	
26	Выбор методов тестирования	Задача	2	24
27	Тестирование ПС	Задача	5	25
28	Оценка качества программного кода	Задача	3	26

Продолжение таблицы 4

1	2	3	4	5
29	Контроль качества ПС администрирования и обнаружения событий завершен	Веха	0	27
30	Завершение работы над проектом	Фаза	5	
31	Оформление документации	Задача	4	29
32	Сдача проекта	Задача	1	31
33	Проект завершен	Веха	0	33

После составления плана выполнения работ необходимо оценить ресурсы проекта и распределить их по имеющимся задачам. Информация об имеющихся в проекте ресурсах и их распределение представлены в таблицах 5 и 6 соответственно.

Таблица 5 – Ресурсы проекта

Название ресурса	Тип ресурса	Затраты		Начисление
		Стандартная ставка	Ставка сверхурочных	
Программист	Трудовой	600 р/ч	1000 р/ч	Пропорциональное
Аналитик	Трудовой	500 р/ч	800 р/ч	Пропорциональное
Тестировщик	Трудовой	500 р/ч	800 р/ч	Пропорциональное
Компьютер	Материальный			Пропорциональное
Интернет	Затраты			Пропорциональное
Печать	Затраты			Пропорциональное

Таблица 6 – Распределение ресурсов по задачам

Название задачи	Ресурсы	Единицы (затраты)
1	2	3
Начало проекта	Компьютер	1
Анализ предметной области и планирование процесса разработки программных средств		100%
Анализ основ организации систем потоковой обработки данных	Аналитик	100%
Анализ типологии событий	Аналитик	100%
Анализ процесса обнаружения сложных составных событий	Аналитик	100%
Сравнительный анализ аналогов	Аналитик, Интернет	100%
Постановка цели разработки	Аналитик, программист	100%
Анализ предметной области и планирование процесса разработки программных средств завершены	Аналитик, программист	
Анализ требований и постановка задачи на разработку программных средств		100%
Анализ функциональных требований	Аналитик	100%
Анализ нефункциональных требований	Аналитик	100%
Анализ требований к пользовательскому интерфейсу панелей администрирования и диспетчеризации событий	Аналитик	100%
Анализ требования к аппаратному обеспечению	Аналитик, программист	100%
Анализ требований к системному программному обеспечению	Аналитик, программист	100%
Анализ требований к средствам разработки	Аналитик, программист	100%
Постановка задачи на разработку программных средств	Аналитик	100%

Продолжение таблицы 6

Анализ требований и постановка задачи на разработку программных средств завершены		
Разработка ПС администрирования и обнаружения событий		100%
Разработка архитектуры развертывания программных средств	Программист	100%
Реализация развертывания событий в системе потоковой обработки данных Apache Kafka декларативными средствами языка kSQL	Программист	100%
Разработка серверных программных средств администрирования и развертывания событий в системе потоковой обработки данных	Программист	100%
Разработка клиентского приложения администрирования событий	Программист	100%
Разработка клиентского приложения диспетчеризации событий	Программист	100%
Разработка ПС администрирования и обнаружения событий завершены		
Контроль качества программных средств администрирования и обнаружения событий		100%
Выбор методов тестирования	Тестировщик	100%
Тестирование ПС	Тестировщик	100%
Оценка качества программного кода	Тестировщик	100%
Контроль качества ПС администрирования и обнаружения событий завершен		
Завершение работы над проектом		100%
Оформление документации	Аналитик, печать	100%
Сдача проекта		100%
Проект завершен		

Диаграмма Ганта – это визуальный инструмент управления проектами, который предназначен для фиксации задач, контрольных точек, сроков выполнения и других элементов проекта сроки для эффективного планирования и отслеживания проектной деятельности.

Данный вид диаграмм упрощает сложное планирование, улучшает распределение ресурсов и обеспечивает общий обзор для всех заинтересованных сторон. Ключевые компоненты включают списки задач, сроки, зависимости и контрольные точки, поддерживающие совместную работу и мониторинг прогресса.

Проект разделен на логичные фазы, каждая из которых включает в себя набор задач, необходимый для ее полного и успешного завершения. На рисунке 8 представлена разработанная диаграмма Ганта, которая наглядно отображает план выполнения задач проекта, суммарные затраты и распределенные для них ресурсы.

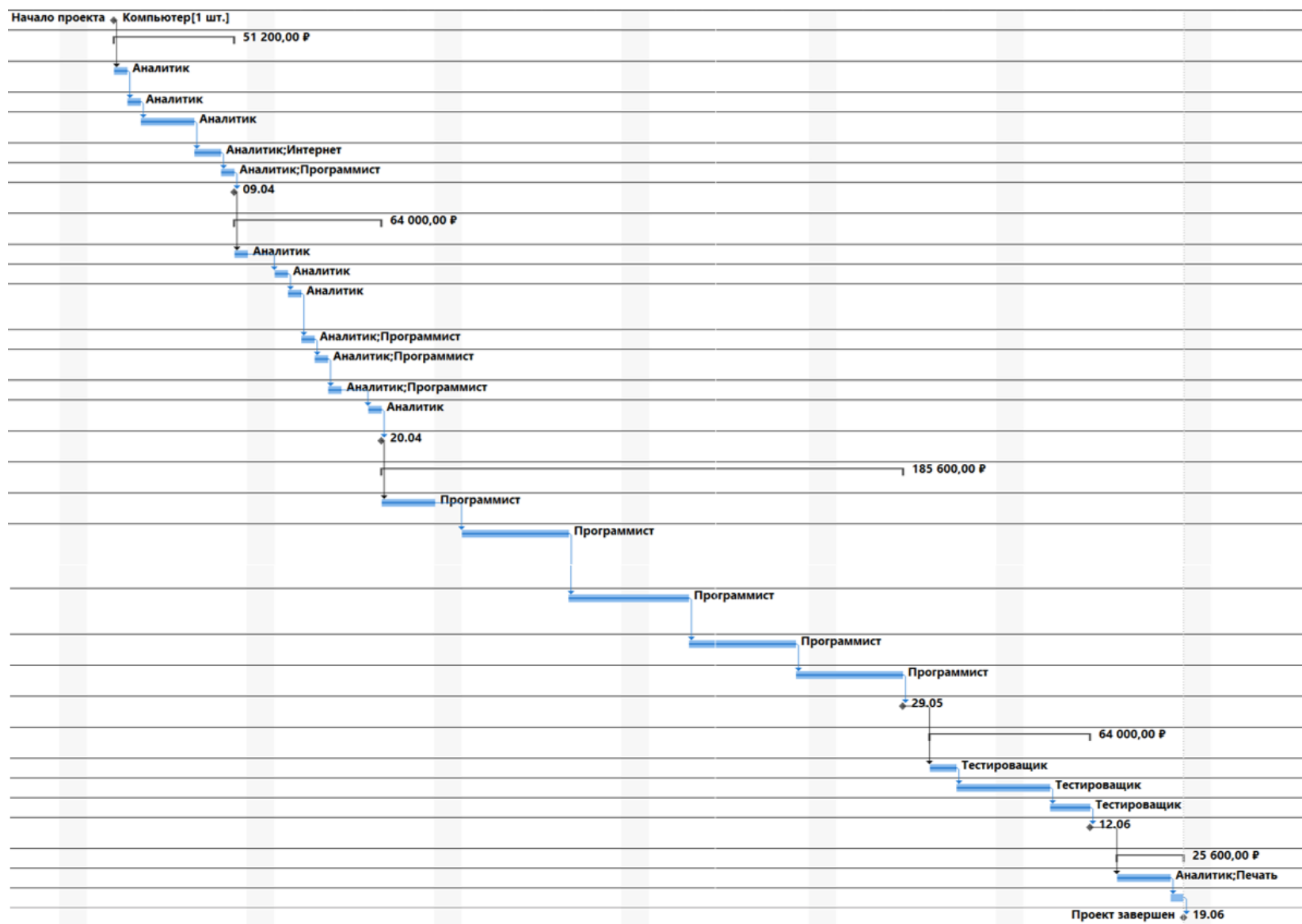


Рисунок 8 – Диаграмма Ганта

В Microsoft Project есть возможность формирования отчетов разных типов, среди которых наиболее распространенными являются обзор о затратах ресурсов и обзор проекта. Обзор о затратах ресурсов отражает стоимость ресурсов для создания проекта. Обзор по проекту показывает прогресс и длительность основных задач по проекту. На рисунках 9 и 10 представлены обзор о затратах ресурсов и обзор по проекту соответственно.

ОБЗОР ЗАТРАТ РЕСУРСОВ

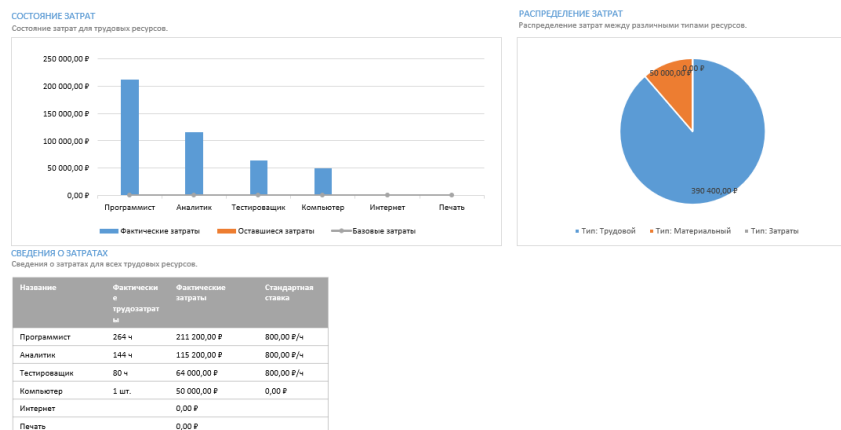


Рисунок 9 – Обзор затрат ресурсов проекта

ОБЗОР ПРОЕКТА

CP 01.04.26 - ПТ 19.06.26

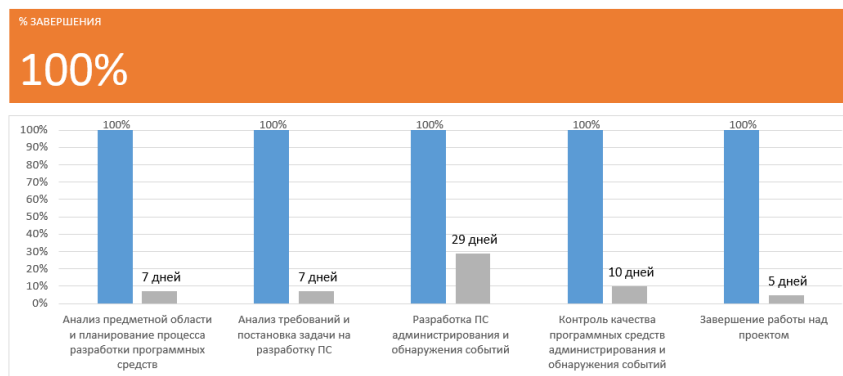


Рисунок 10 – Обзор проекта

Сформированные отчеты показали, что общая стоимость разработки ПС администрирования и развертывания событий в системе потоковой обработки данных составляет 440400 рублей. В ходе реализации проекта было выполнено 33 задачи. Проект был выполнен на 100% за 79 дней.

2 Анализ требований и постановка задачи на разработку программных средств

2.1 Анализ функциональных требований

Функциональные требования (functional requirements) определяют, каким должно быть поведение разрабатываемой системы в тех или иных условиях. Они определяют, что разработчики должны создать, чтобы пользователи смогли выполнить свои задачи. Функциональные требования описываются в форме традиционных утверждений со словами «должен» или «должна» [15].

На этапе определения назначения программных средств было выделено два основных класса пользователей для разрабатываемых модулей системы:

- администратор данных (событий), отвечающий за управление объектами системы, а также конструирование новых событий для последующего обнаружения;
- оператор данных (событий), отвечающий за мониторинг отлавливаемых в потоках событий.
- Для каждого класса пользователь должна разрабатываться отдельная клиентская панель, так как функционал, доступный администратору и оператору не пересекается.

Диаграммы вариантов использования в UML описывают функциональное назначение системы или то, что система должна делать. Разработка диаграммы преследует следующие цели:

- определить общие границы и контекст моделируемой предметной области;
- сформулировать общие требования к функциональному поведению проектируемой системы;
- разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей;
- подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями [16].

Результаты анализа функциональных требований к панели администратора представлены в виде диаграммы вариантов использования на рисунке 11.

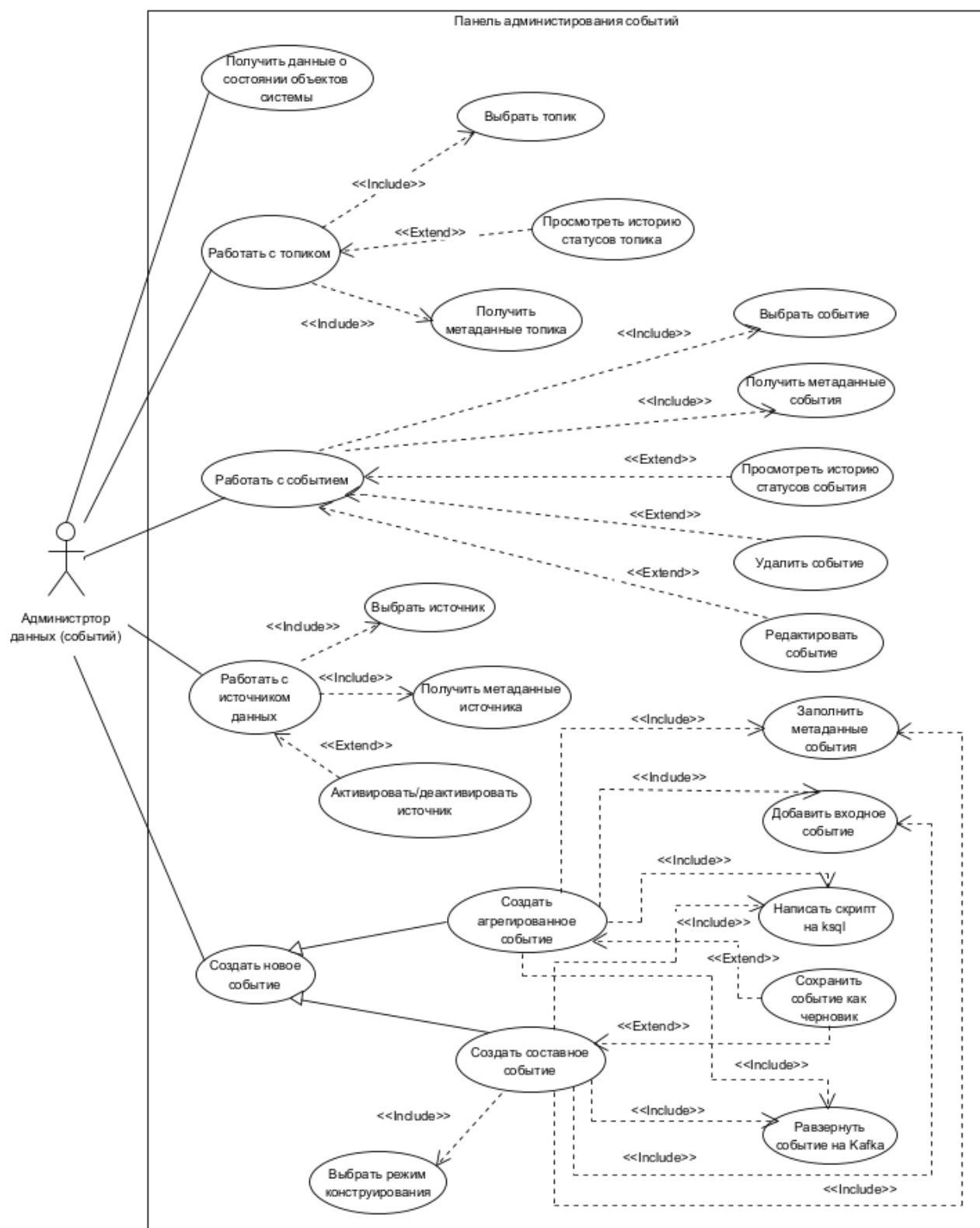


Рисунок 11 – Диаграмма вариантов использования для панели администратора данных

Результаты анализа функциональных требований к панели оператора представлены в виде диаграммы вариантов использования на рисунке 12.

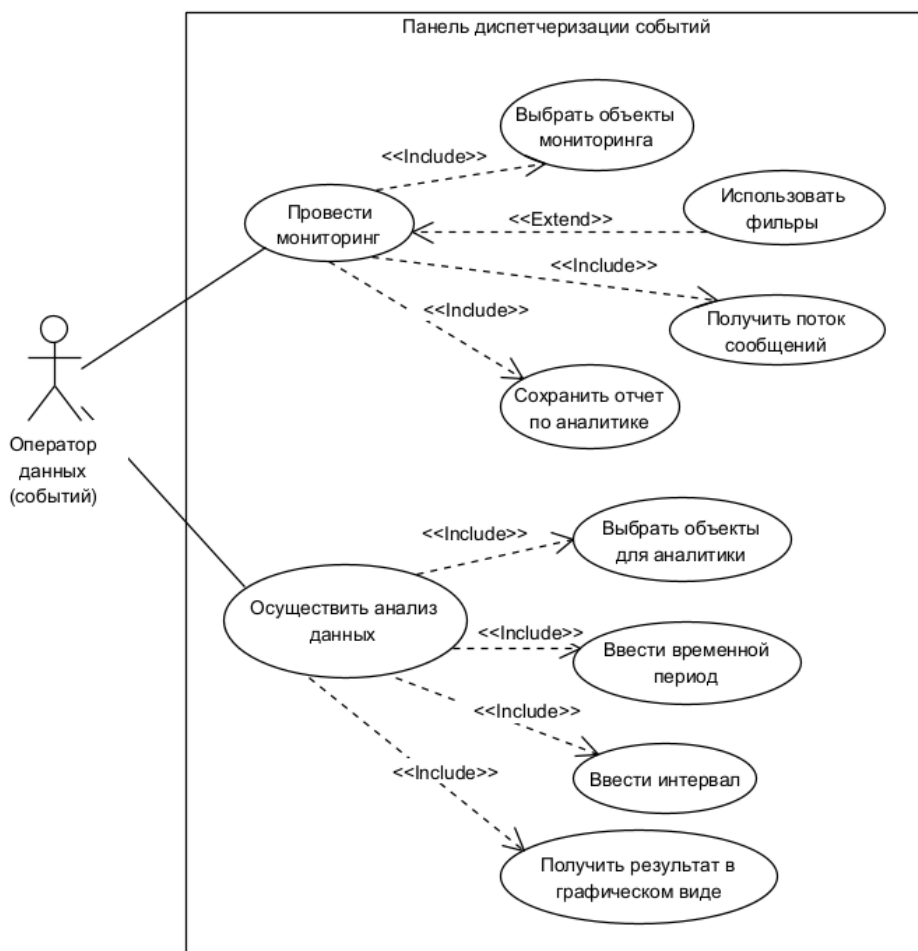


Рисунок 12 – Диаграмма вариантов использования для панели оператора данных

Диаграмма вариантов использования позволяет легко визуализировать взаимодействие между пользователем и системой, а также выявить ключевые функции, которыми они пользуются.

Администратор данных должен иметь возможность управлять топиками, а именно просматривать их список, обновлять его, осуществлять поиск в этом списке, просматривать подробную информацию по выбранному топикам, в том числе и набор связанных с ним событий. Так же администратор должен иметь возможность активировать начало приема входных данных от источника на стороне Kafka. Администратор должен иметь возможность инициировать начало отлова созданного события, что означает развертывание соответствующих

скриптов на сервере `ksqlDB`. Главное требование к панели администратора – предоставление пользователю возможности создания нового события посредством удобного визуального конструктора с несколькими режимами.

Оператор данных должен иметь доступ к спискам отлавливаемых в системе событий. Таким образом, к функциональным требованиям для оператора относятся возможность проведения мониторинга по выбранному событию или набору событий, а также просмотр поступающих сообщений в реальном времени. На основе этих данных он должен иметь возможность получения аналитики и сохранения отчетов для последующего анализа.

В таблице 7 были описаны все спецификации бизнес-процессов администратора системы.

Таблица 7 – Спецификации бизнес-процессов для администратора системы

Актор	Бизнес-процесс	Описание
Администратор данных	Получить данные о состоянии объектов системы	Пользователь получает доступ к общему текущему состоянию источников данных, топиков и событий. Он может просматривать количественные показатели по активной работе или ошибкам всех зарегистрированных в системе объектов. Пользователь просматривает текущее состояние соединений с внешними компонентами программных средств, а именно с базой метаданных, кластером Kafka и сервером <code>ksqlDB</code> .
Администратор данных	Работать с топиком Kafka	Пользователь просматривает полный список источников и имеет возможность просмотреть метаданные по выбранному источнику. Каждый из источников пользователь может активировать или деактивировать в зависимости от его текущего статуса.
Администратор данных	Работать с источником данных	Пользователь просматривает полный список топиков и имеет возможность просмотреть метаданные по выбранному топик и его историю статусов.
Администратор данных	Работать с событием	Пользователь просматривает полный список событий и имеет возможность просмотреть метаданные по выбранному событию и его историю статусов. При необходимости пользователь может редактировать событие или удалить его, если это не противоречит установленным ограничениям.
Администратор данных	Создать новое событие	Пользователь выбирает тип нового события: сложное агрегированное или сложное составное, и режим конструирования. После этого он заполняет метаданные события (название, краткий код, топик результата и т.д.). Пользователь выбирает входные потоки или таблицы, пишет скрипт <code>ksql</code> с помощью автоматически собираемого шаблона и может сохранить событие в качестве черновика или сразу развернуть его на Kafka.
Оператор данных	Провести мониторинг	Пользователь выбирает объекты для мониторинга – активные события – вручную или с помощью фильтров и подписывается на их Kafka топик. После этого он начинает получать поток сообщений в реальном времени по отлавливаемым в системе событиям. Пользователь имеет возможность остановить мониторинг, тем самым сохранив о нем подробный отчет.
Оператор данных	Осуществить анализ данных	Пользователь выбирает объекты для аналитики – активные события – и получает статистику за выбранный временной период с указанным интервалом в виде столбчатой диаграммы и графика.

В таблице 8 приведен сценарий основного для администратора данных варианта использования «Создать новое событие».

Таблица 8 – Сценарий варианта использования «Создать новое событие»

Прецедент	Создать новое событие
Краткое описание	Пользователь создает сложное агрегированное или сложное составное событие
Главные актеры	Администратор данных
Второстепенные актеры	Нет
Предусловие	Пользователь авторизован в панели администратора; в системе существуют входные события или потоки, которые могут быть использованы при создании нового события
Основной поток	<ol style="list-style-type: none"> 1) пользователь переходит на вкладку «События»; 2) пользователь нажимает кнопку «Создать»; 3) система открывает окно создания события; 4) пользователь выбирает тип создания события; 5) пользователь заполняет основные данные события, включая имя события, краткий код, объект результата, топик результата, описание и другие параметры; 6) пользователь выбирает входное событие или два входных события в зависимости от режима создания; 7) система автоматически подставляет контекст из входного события, если он указан; 8) пользователь формирует SQL-шаблон события; 9) пользователь при необходимости просматривает SQL-код входных событий во всплывающем окне; 10) пользователь нажимает кнопку «Создать черновик» или «Создать и развернуть»; 11) система сохраняет событие в базе метаданных; 12) при выборе развертывания система отправляет запросы к ksqlDB и Kafka для создания результирующих объектов.
Альтернативные потоки	<ol style="list-style-type: none"> 4.а если пользователь выбирает агрегированное событие, система разрешает выбрать только одно входное событие; 4.б если пользователь выбирает сложное составное событие в ручном режиме, система требует выбрать ровно два входных события; 4.в если пользователь использует конструктор Аллена, система формирует SQL-запрос на основе выбранных событий, отношения и временного интервала; 7.а если у входного события не указан контекст, пользователь выбирает контекст вручную; 9.а если входное событие не выбрано, просмотр SQL-кода входа недоступен; 12.а если при развертывании возникла ошибка, система сохраняет событие как черновик и отображает текст ошибки в нижней области окна; 12.б если событие было создано, но не развернуто из-за ошибки, окно переходит в режим редактирования созданного черновика.
Постусловие	Новое событие сохранено как черновик или создано и развернуто на Kafka и ksqlDB

В таблице 9 приведен сценарий основного для оператора данных варианта использования «Провести мониторинг».

Таблица 9 – Сценарий варианта использования «Провести мониторинг»

Прецедент	Провести мониторинг
Краткое описание	Пользователь наблюдает за поступлением сообщений по выбранным активным событиям в реальном времени
Главные актеры	Оператор данных
Второстепенные актеры	Нет

Продолжение таблицы 9

Предусловие	Пользователь авторизован в панели оператора; в системе существуют активные события, доступные для мониторинга
Основной поток	1) пользователь переходит на вкладку «Главная» панели оператора; 2) система отображает список активных событий; 3) пользователь находит нужные события вручную или с помощью поиска и фильтров; 4) пользователь отмечает одно или несколько событий для мониторинга; 5) пользователь нажимает кнопку перехода к мониторингу; 6) система открывает вкладку «Мониторинг»; 7) пользователь нажимает кнопку «Старт»; 8) клиент отправляет запрос к серверу для проверки выбранных событий и получения списка результирующих Kafka topics; 9) клиент устанавливает WebSocket-соединение с сервером; 10) сервер подписывается на соответствующие Kafka топики; 11) система отображает поступающие сообщения в таблице; 12) пользователь выбирает сообщение и просматривает его в отформатированном виде; 13) пользователь нажимает кнопку «Стоп»; 14) система останавливает мониторинг и сохраняет отчет по сессии.
Альтернативные потоки	4.а если пользователь выбирает больше допустимого количества событий, система запрещает выбор лишних событий; 7.а если события не выбраны, система не запускает мониторинг и выводит сообщение; 9.а если WebSocket-соединение не удалось установить, система выводит сообщение об ошибке; 10.а если Kafka недоступна, система останавливает мониторинг и выводит ошибку; 13.а если пользователь очищает поток, система удаляет сообщения с экрана, но не останавливает мониторинг.
Постусловие	Пользователь просмотрел поступающие сообщения в реальном времени; после остановки мониторинга сформирован и сохранен отчет

Моделирование процесса выполнения сценариев использование может быть выполнено при помощи диаграммы деятельности.

Диаграмма деятельности используется для визуализации последовательностей реализации операций, входящих в состав выполняемых системой сценариев. Деятельность в этом случае означает совокупность вычислений, выполняемых объектом и приводящих к некоторому результату или действию. На диаграмме деятельности отображается последовательность переходов от одного действия к другому, а также фиксируется логика этих переходов. Результат деятельности может привести к изменению состояния системы, возвращению некоторого значения или выполнению иного действия.

Ключевым сценарием в разрабатываемых программных средствах является процесс создания нового сложного события администратором.

Для более наглядного представления алгоритма действий администратора при создании нового события была разработана диаграмма деятельности, представленная на рисунке 13.

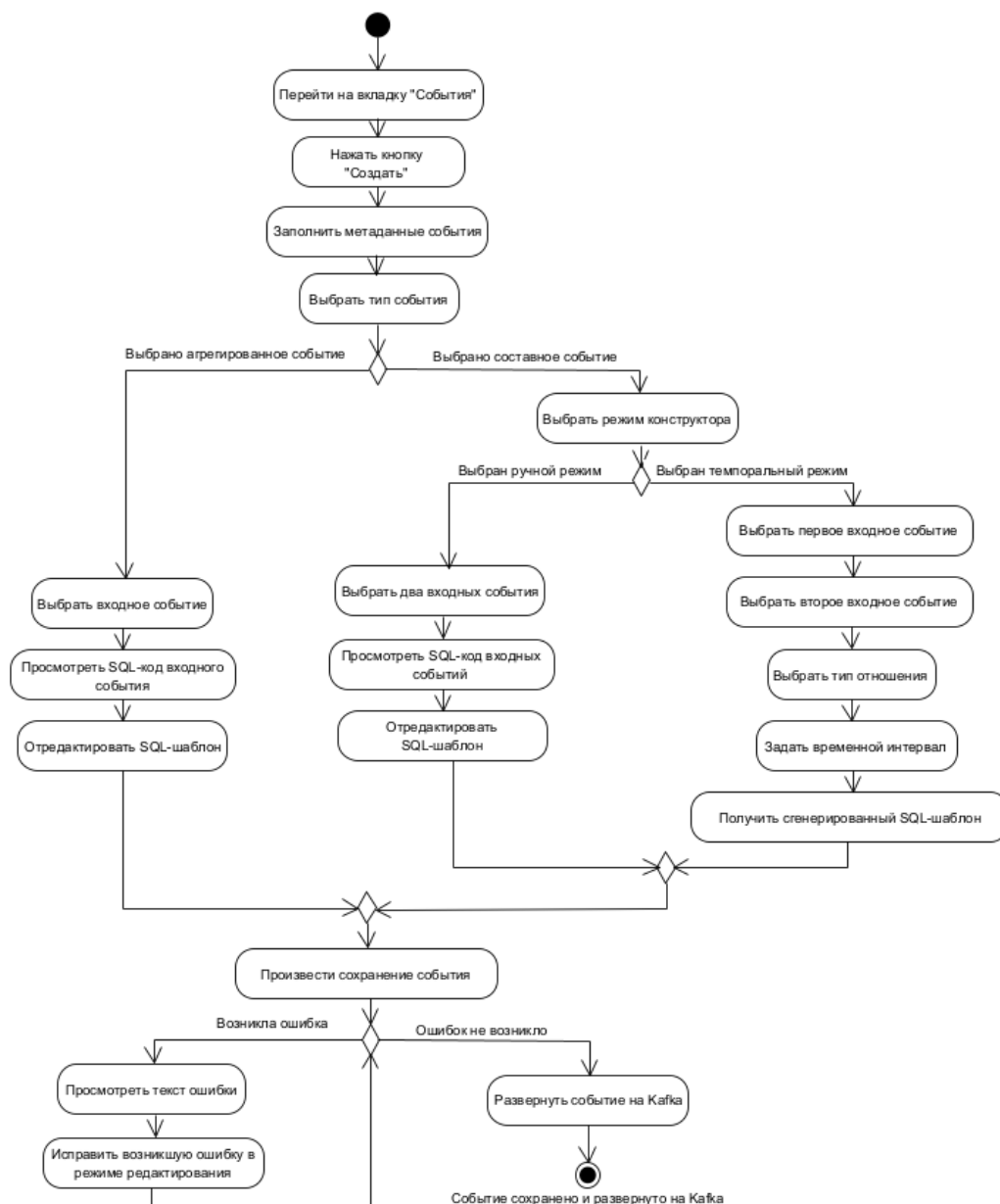


Рисунок 13 – Алгоритм действий администратора для создания нового события

Таким образом, возможности администратора обладают вариативностью, что повышает качество системы. Он может выбирать не только тип создаваемого события, но и использовать разные режимы, просматривать код тех событий и таблиц, которые будут являться входными для нового события, пользоваться автоматической генерацией шаблонов, и получать доступ к редактированию события в случае возникновения ошибок.

В разрабатываемой системе администратор имеет доступ к управлению всеми объектами системы: источниками данных, топиками и событиями, а оператор может проводить мониторинг и получать аналитику за интересующий

период. Данный функционал в полной мере отражает идею объединения потоковой обработки и обнаружения событий.

2.2 Анализ нефункциональных требований

Нефункциональные требования описывают различные свойства системы или ограничения на ее поведение, включая ограничения на среду разработки, надежность, производительность, безопасность, удобство использования и внешний вид [15]. Это критерии, по которым оценивают не саму логику работы, а работу системы в целом.

С точки зрения архитектуры программного обеспечения нефункциональные требования – это свойства архитектуры (или архитектурные характеристики).

Существует множество нефункциональных требований. К ним относят:

- доступность – время доступности системы (если предполагается работа в режиме 24/7, необходимо сделать все возможное для быстрого восстановления работоспособности в случае сбоя);
- бесперебойность, что означает способность аварийного восстановления;
- производительность, включающая стресс-тестирование, анализ пиковых нагрузок, анализ частоты использования функций, потенциальных возможностей и времени отклика;
- восстанавливаемость, как требование к непрерывности работы по прямому назначению (например, скорость выхода системы из аварийных ситуаций), что влияет на стратегию резервного копирования и на требования к дублированию оборудования;
- надежность, то есть потребность в безотказной работе системы или особая важность ее функционирования в структурах жизнеобеспечения;
- удобство использования, то есть интуитивно понятный интерфейс системы, доступный для пользователей с разным уровнем подготовки;

- сопровождаемость, означающая возможность легко обновлять и поддерживать программное обеспечение, обеспечивая минимальные затраты ресурсов на усовершенствование и исправление ошибок;
- робастность, что означает способность к обработке ошибок в ходе выполнения программы и предельных условий работы при отключении от интернета или же при потере сетевого напряжения или аппаратном сбое;
- масштабируемость, то есть работоспособность системы при росте числа пользователей или ужесточении уровня технических требований [17].

Для программных средств администрирования и мониторинга событий в системах потоковой обработки были установлены нефункциональные требования, представленные в таблице 9.

Таблица 9 – Нефункциональные требования для ПС администрирования и обнаружения событий

№ п/п	Требование	Значение
1	Производительность	Минимальная задержка при просмотре потоков и топиков: время от появления сообщения в Kafka до его отображения в интерфейсе пользователя должно составлять не более 2-3 секунд
2	Надежность	97% времени система должна работать корректно
3	Удобство использования	Порог обучения нового пользователя до 30 минут, благодаря интуитивно понятному интерфейсу, контекстной помощи и валидации
4	Масштабируемость	Возможность горизонтального масштабирования за счет добавления новых узлов кластера Apache Kafka без прерывания работы и деградации производительности при росте объемов обрабатываемых потоков событий
5	Сопровождаемость	Минимальные затраты времени и ресурсов на обновления, улучшения и исправления ошибок

Данный набор нефункциональных требований, состоящий из производительности, надежности, удобства использования, масштабируемости и сопровождаемости позволит разработать надежную и функциональную систему для обнаружения событий и мониторинга поступающих сообщений.

Таким образом, выделенные нефункциональные требования фокусируются на создании безопасного и интуитивно понятного инструмента с высоким уровнем производительности и масштабируемости, что обеспечит его стабильную работу в будущем.

2.3 Анализ требований к пользовательскому интерфейсу приложений администрирования и диспетчеризации событий

2.3.1 Разработка навигационной схемы пользовательского интерфейса приложений администрирования и диспетчеризации событий

Пользовательский интерфейс – это набор программных и аппаратных средств, обеспечивающих взаимодействие пользователя с компьютером (или приложением). Основу такого взаимодействия составляют диалоги. Под диалогом в данном случае понимают регламентированный обмен информацией между человеком и компьютером (приложением), осуществляемый в реальном масштабе времени и направленный на совместное решение конкретной задачи [18]. Корректно разработанный интерфейс способствует минимизации количества ошибок, ускорению процесса работы и повышению комфорта пользователей.

В пользовательском интерфейсе панели администратора необходимо предусмотреть набор из шести окон, главное из которых включает в себя четыре вкладки. Для панели оператора будет разработан интерфейс, состоящий из двух окон, где главное окно включает в себя три вкладки.

Схема навигации по панели оператора отображена на рисунке 14.

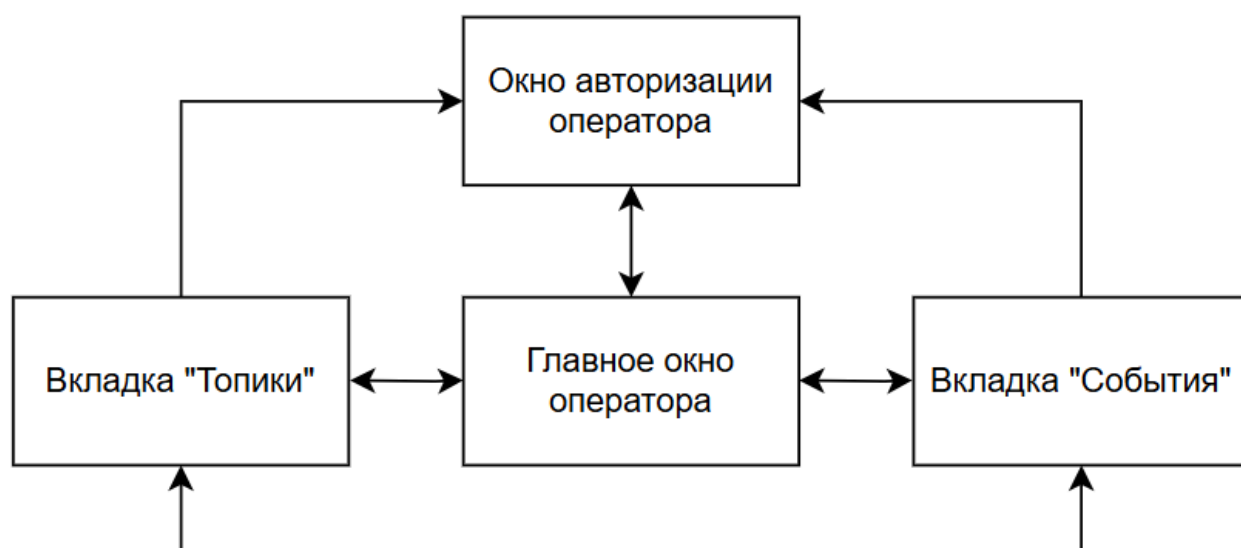


Рисунок 14 – Схема навигации панели администратора данных

Полная схема навигации по панели администратора отображена на рисунке 15.

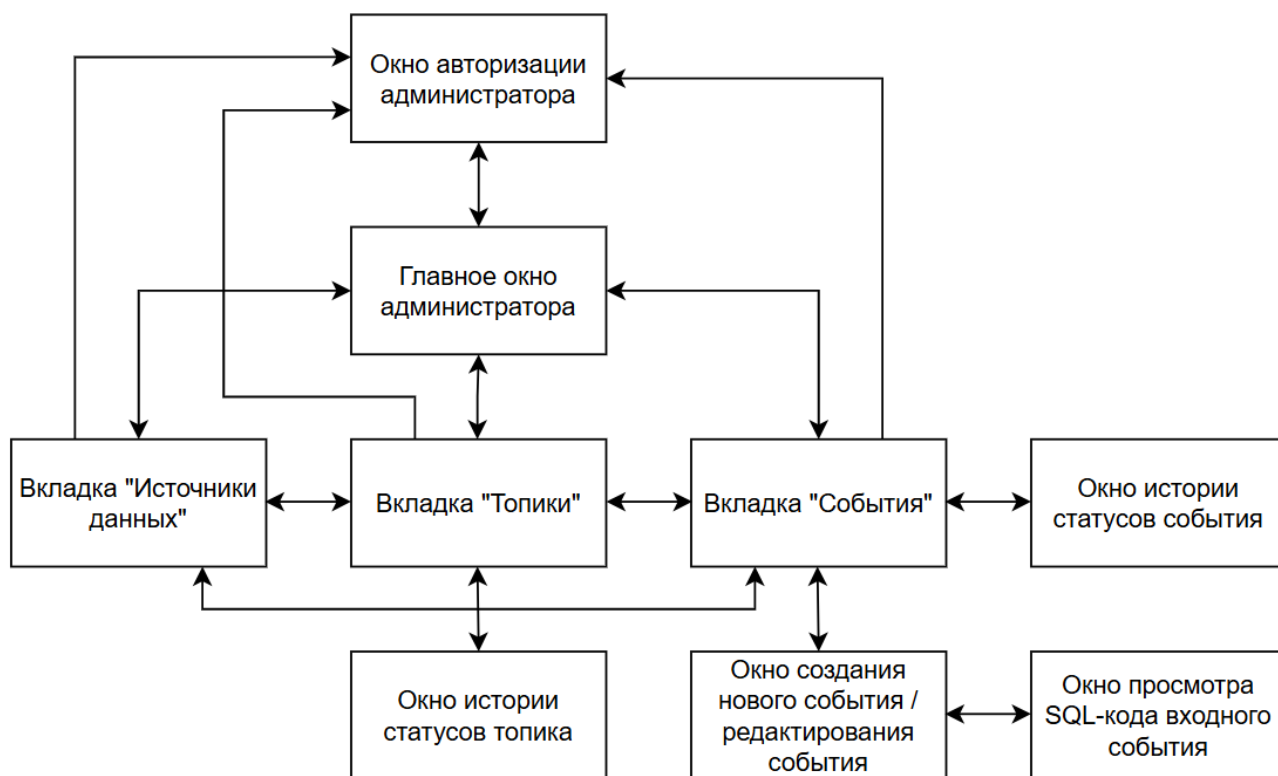


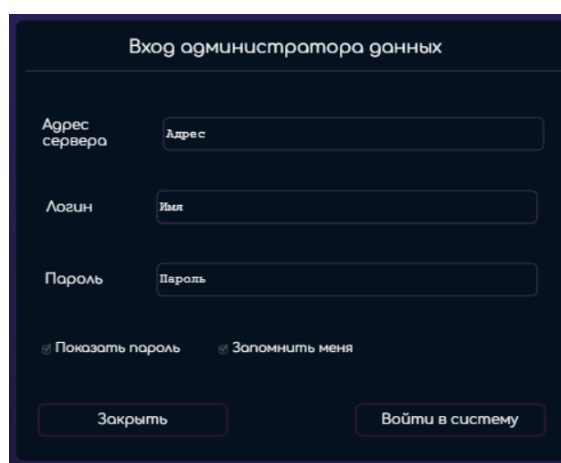
Рисунок 15 – Схема навигации панели администратора данных

Приведенные схемы демонстрируют интуитивно понятный интерфейс для новых пользователей, который позволяет легко ориентироваться в системе, находить нужные функции и эффективно перемещаться между окнами. Такое структурированное представление упрощает выполнение операций, минимизирует риск ошибок и повышает общую производительность приложений.

2.3.2 Разработка прототипов интерфейса приложений администрирования и диспетчеризации событий

Первое окно, на которое попадет пользователь сразу после запуска приложения, является окном авторизации. Его функциональность для администратора и оператора не имеет различий. В обоих приложениях пользователь должен будет ввести заранее зарегистрированные в системе логин

и пароль. Используя функцию «запомнить меня», пользователь получит возможность сохранить введенные логин и пароль для повторного входа без необходимости заполнять соответствующие поля повторно в рамках текущего запуска приложения. Так как программные средства предназначены для использования конкретными лицами (сотрудниками, обслуживающими систему), а не свободным кругом пользователей, окна регистрации не предусмотрены. Предполагается, что регистрация сотрудников будет проводиться отдельно, как и внесение данных о наблюдаемой системе. Прототип окна авторизации администратора представлен на рисунке 16.



Вход администратора данных

Адрес сервера

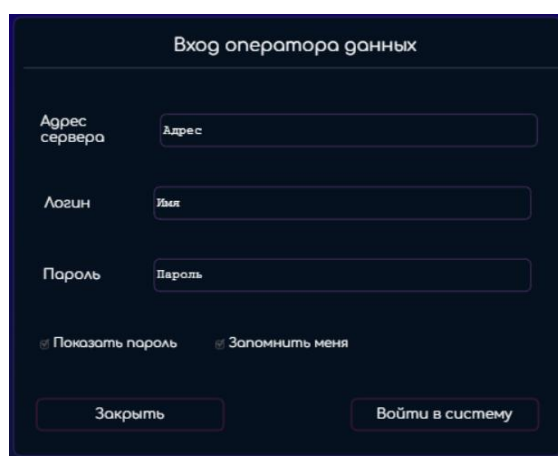
Логин

Пароль

☒ Показать пароль ☒ Запомнить меня

Рисунок 16 – Прототип окна авторизации администратора

Прототип окна авторизации оператора представлен на рисунке 17.



Вход оператора данных

Адрес сервера

Логин

Пароль

☒ Показать пароль ☒ Запомнить меня

Рисунок 17 – Прототип окна авторизации оператора

После ввода данных и успешного прохождения авторизации и аутентификации пользователь попадет на главное окно своего приложения.

Главное окно панели администратора должно содержать актуальную информацию касательно текущего пользователя и состояния наблюдаемой системы. В верхней панели будет отображаться число запущенных в данный момент источников данных. На трех панелях ниже должна отображаться общее текущее состояние всех объектов системы, а именно: источников, топиков и событий. В этих панелях будет выводиться не только количество активных объектов, но и число ошибок в их работе. Также там будет отображаться статус, который чаще всего встречается среди объектов выбранной категории.

Эта информация позволит отслеживать и поддерживать корректный статус системы в целом. В нижней панели администратор сможет проверить доступность инфраструктуры системы администрирования и обнаружения событий. Наиболее важными компонентами в ней являются база данных, кластер Kafka и ksqlDB сервер. Именно они будут обеспечивать полноценную работу панели администратора. В специальной таблице должны отображаться статус доступности компонента, его отклик в миллисекундах, а также сведения о текущем состоянии или возникших ошибках. Это позволит администратору сразу обнаружить неисправности в работе системы до того, как он приступит к выполнению своих непосредственных задач. Прототип главного окна панели администратора представлен на рисунке 18.

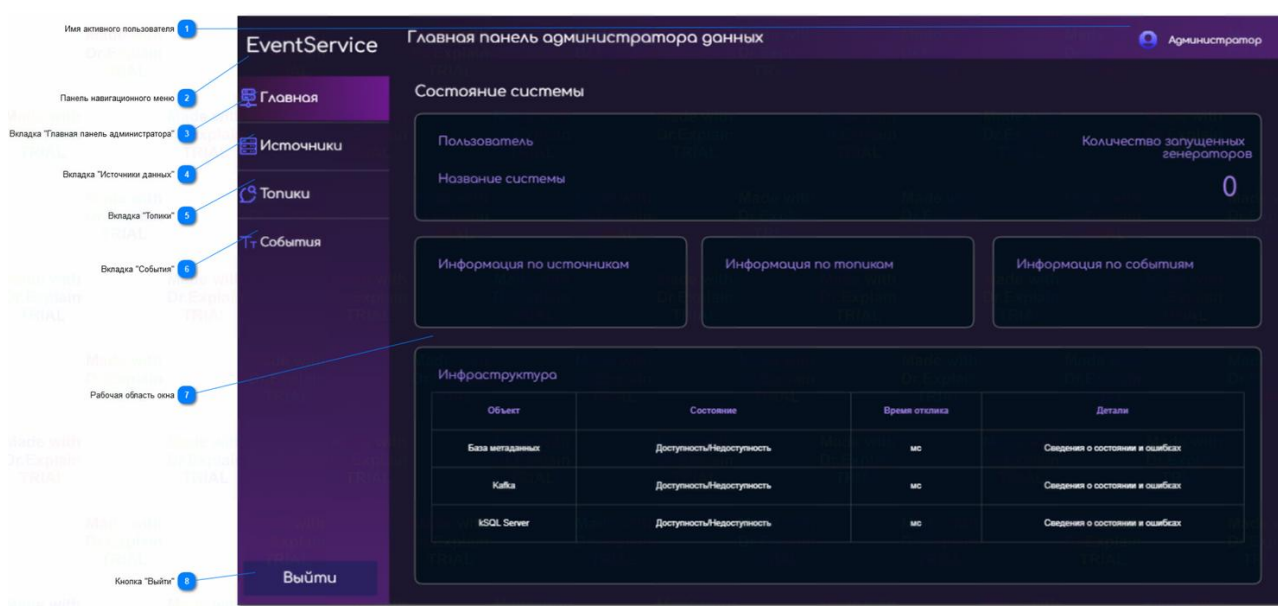


Рисунок 18 – Прототип главного окна панели администратора

Интерфейс главного окна панели администратора, представленный на рисунке 18, данных предусматривает следующий набор элементов:

- 1 – имя активного пользователя;
- 2 – навигационное меню;
- 3 – вкладка «Главная»;
- 4 – вкладка «Источники»;
- 5 – вкладка «Топики»;
- 6 – вкладка «События»;
- 7 – рабочая область окна;
- 8 – кнопка «Выйти».

Имя активного пользователя должно отображаться в верхнем правом углу главного окна. В левой части окна будет располагаться навигационное меню, состоящее из четырех вкладок. Администратор может быстро перемещаться по вкладкам, распределенным под объекты наблюдаемой системы. Также в нижней части меню расположена кнопка «Выйти», возвращающая пользователя на окно авторизации без полного закрытия приложения.

Перейдя на вкладку «Источники», администратор данных получит доступ к списку источников, генерирующих непрерывные потоки данных. Регистрация этих источников происходит заранее, до введения программных средств в полноценную эксплуатацию. Добавление новых устройств, датчиков и других источников, а также настройка их коннекторов через панель администратора не предусмотрены.

На данной вкладке пользователь сможет отслеживать текущее состояние всех доступных источников в их полном списке, просматривать их метаданные в правой панели «Карточка источника». Кнопки, расположенные в нижней панели окна, позволят ему активировать или, наоборот, деактивировать передачу данных от источника в Kafka. Это необходимо для того, чтобы в случае сбоя или неполадок на стороне наблюдаемой системы администратор смог отключить передачу неправильных показаний в потоки, что предотвратит отлов событий на некорректных данных и накопление ошибочных сообщений.

При первой активации нового источника система должна запрашивать у пользователя краткий код для автоматически создающегося элементарного события. Практическая ценность данных кодов будет рассмотрена при описании работы окна создания событий.

Логи, выводящиеся в нижней панели, позволят администратору контролировать результаты своих действий и вовремя обнаруживать ошибки в случае их возникновения (например, неудачный запуск генерации источника при внезапном разъединении с кластером Kafka). Прототип вкладки «Источники данных» данных представлен на рисунке 19.

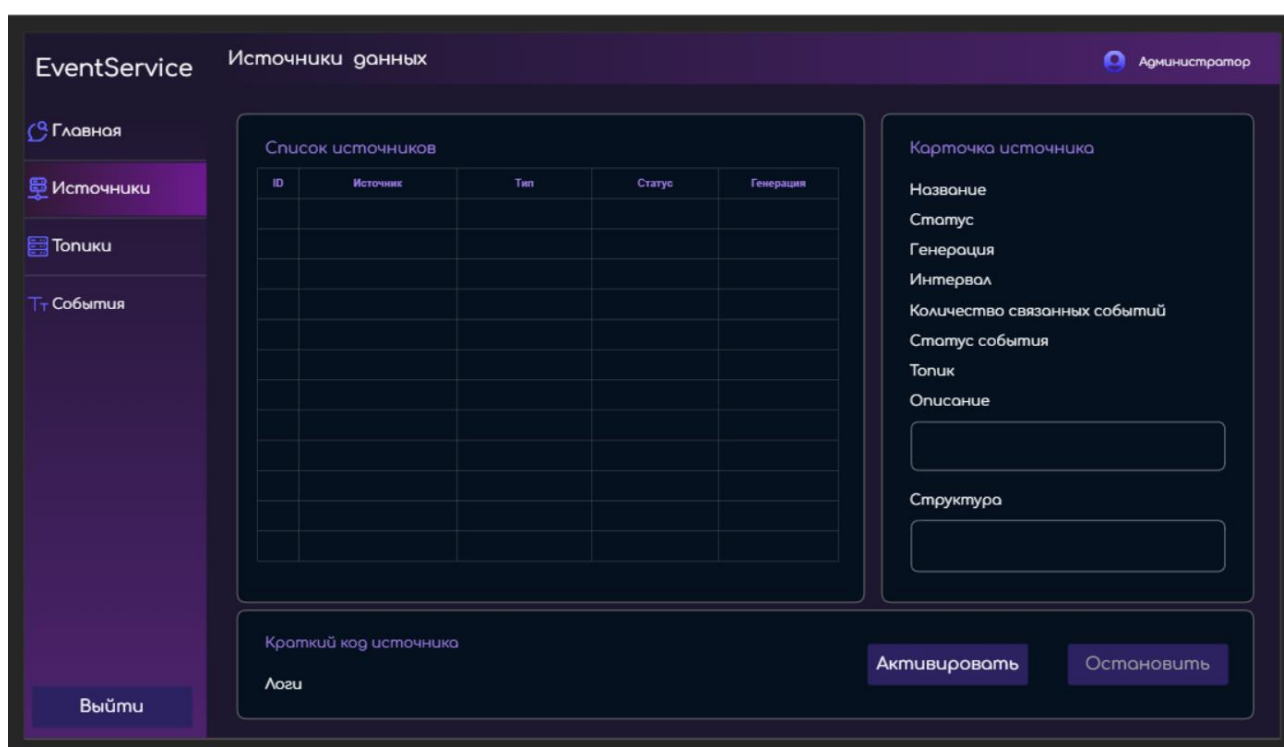


Рисунок 19 – Прототип вкладки «Источники данных»

Перейдя на вкладку «Топики», администратор получит доступ к полному списку топиков Kafka. Более детальную информацию можно будет получить посредством клика на строку таблицы. Панель «Карточка топика» в правой части рабочей области должна отображать набор метаданных выбранного топика.

В нижней панели должны выводиться логи с результатами действий пользователя в данном окне. На рисунке 20 представлен прототип вкладки «Топики Kafka».

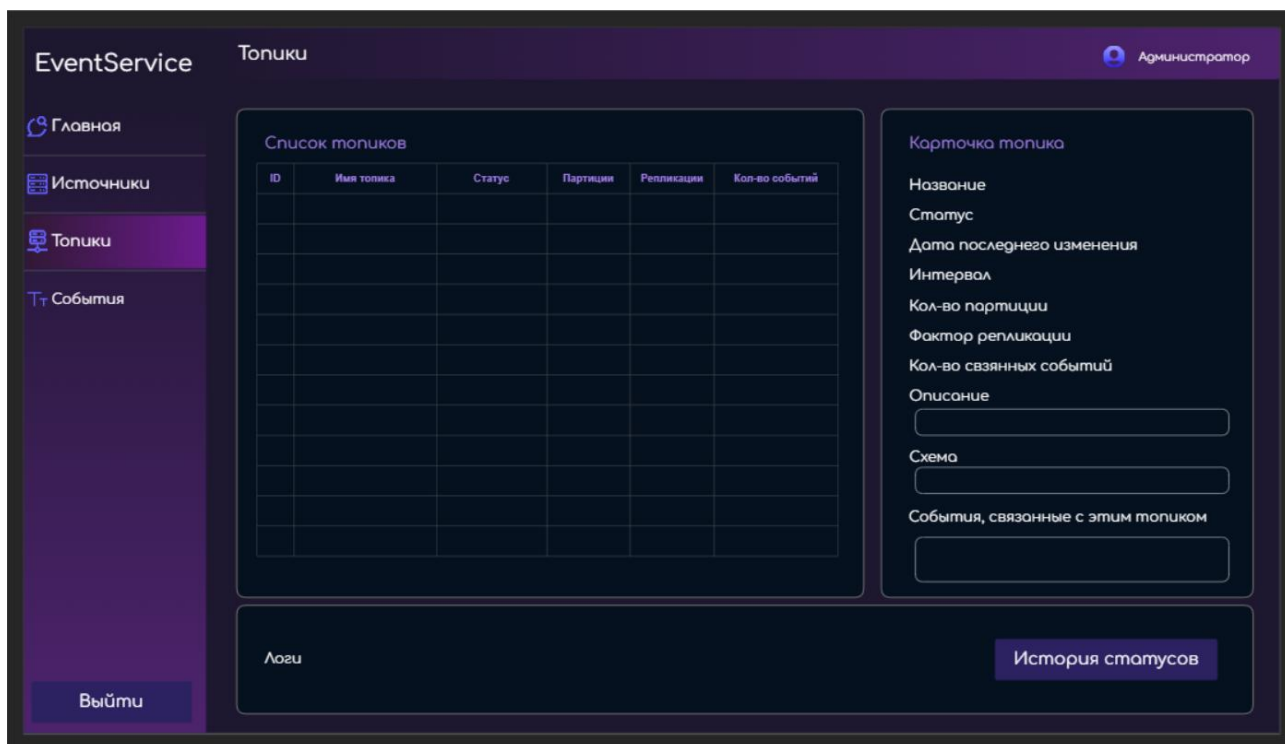


Рисунок 20 – Прототип вкладки «Топики Kafka»

Также там будет располагаться кнопка «История статусов», позволяющая открыть всплывающее окно с полной историей изменений по конкретному топика, например, дату и время его создания, развертывания или возникших ошибок. Эти данные будут отображаться в виде наглядной таблицы. Прототип окна истории статусов топика показан на рисунке 21.

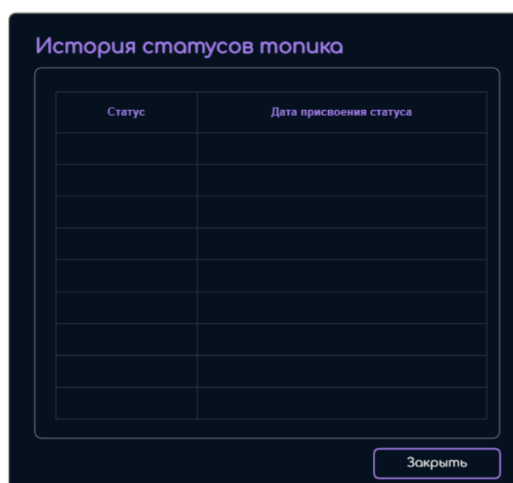


Рисунок 21 – Прототип окна истории статусов топика

При переходе на вкладку «События», администратор получит доступ к информации о созданных и развернутых событиях наблюдаемой системы. При

помощи таблицы в данном окне должен отображаться полный список событий, а при клике на нужную строчку информация о событии должна отображаться в панели «Карточка события» в правой части окна. В этой же панели находятся четыре кнопки. Кнопки «Редактировать» и «Удалить черновик» будут доступны только для событий, которые еще не были развернуты на Kafka. Первая из кнопок позволит перейти на окно редактирования, аналогичную окну создания нового события, описание которого будет представлено позже. Вторая же кнопка позволит удалить неразвернутый черновик из базы метаданных.

Кнопка «Удалить активное событие» предоставит возможность удалить развернутое событие, включая соответствующий ему выходной топик. Удалять можно будет только те активные события, которые не входят в состав других сложных составных событий. Если над выбранным событием нельзя совершить то или иное действие, кнопки должны блокироваться автоматически.

В самой нижней панели должна выводиться таблица дерева состава события. Любое сложное событие обладает входным потоком информации. Сложное агрегированное событие имеет один такой источник. Составное событие может состоять из двух и более входных потоков или таблиц. Для того, чтобы администратор имел представление, из каких источников каждое событие берет данные, какие потоки и таблицы на каких уровнях объединяются для получения нового результата, будет предложена концепция «дерева». В первом столбце таблицы отображается уровень входа. В первой строке всегда выводится выбранное событие с уровнем номер 1. Это корень дерева. Для элементарных событий вывод дерева на этом завершится. Во второй строке для агрегированного события будет отображаться поток, послуживший для него источником. Для составного события число уровней может быть неограниченным.

Подобное представление позволяет не только сохранять полный состав любого события, но и позволяет визуализировать состав события в понятной и удобной форме. При ошибке, возникшей при обнаружении сложного составного события, сможет посмотреть таблицу его состава и определить, на каком уровне

и в каком именно входном потоке произошел сбой благодаря отображению текущего статуса для каждого события, входящего в дерево.

В верхней части рабочей области должна находиться кнопка «Создать новое», позволяющая администратору перейти на окно создания нового события. Прототип вкладки «События» представлен на рисунке 22.

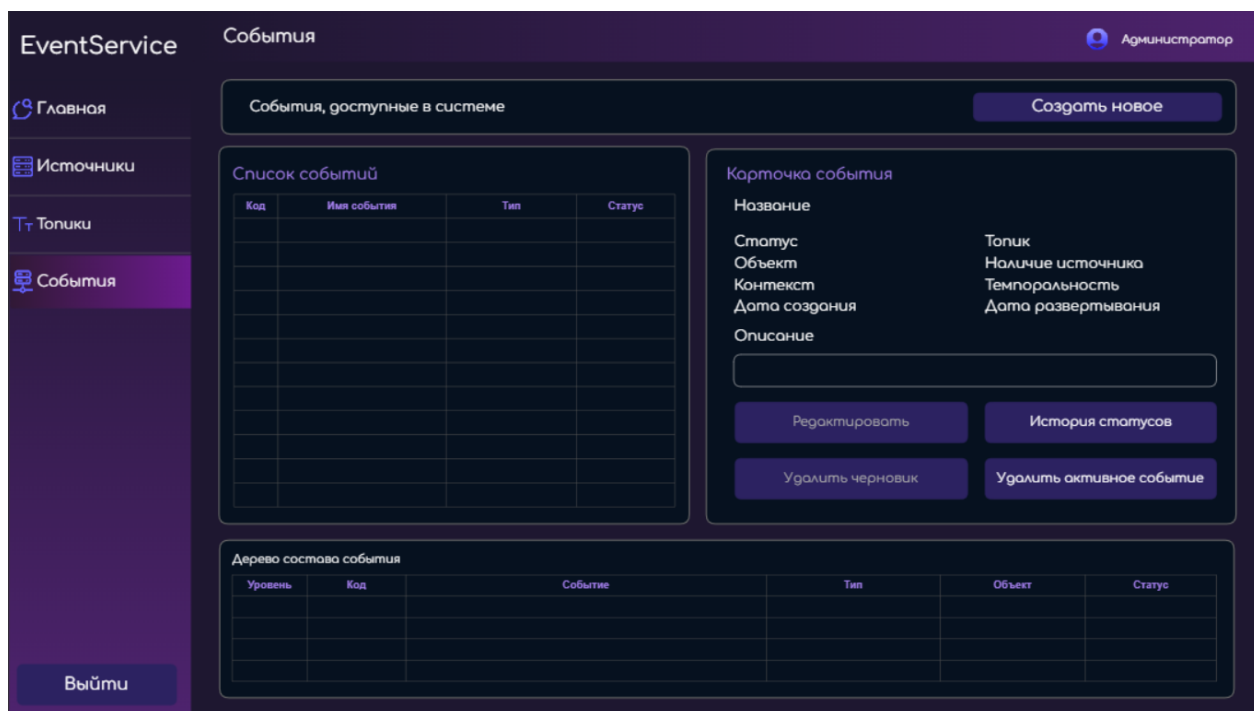


Рисунок 22 – Прототип вкладки «События»

Кнопка «История статусов», по аналогии с подобной кнопкой на окне топиков, позволит администратору перейти на окно истории статусов топика. Прототип окна истории статусов события представлен на рисунке 23.

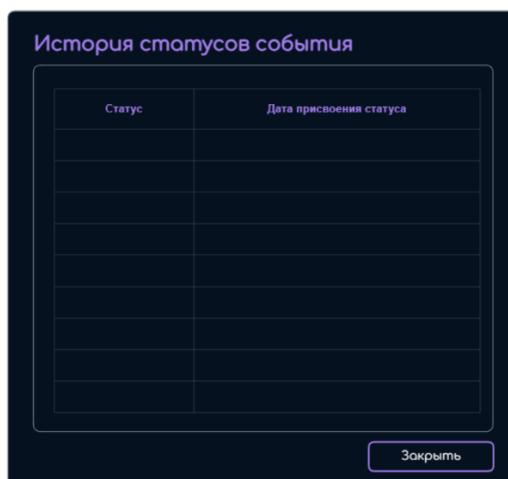


Рисунок 23 – Прототип окна истории статусов события

Окно создания событий должно представлять из себя конструктор, позволяющий администрировать события двух типов: сложные агрегированные и сложные составные. Для создания нового события любого из этих типов пользователю должен быть предложен ввод метаданных события в левой части окна. Панель «Основные данные» обязана включать в себя набор полей, таких как «Название», «Краткий код», «Объект результата», «Топик результата», «Описание», «Временная совместимость» и «Контекст». Поле «Имя события» будет предназначено для указания полного технического имени события. Поле «Краткий код» позволит задать короткое обозначение, которое в дальнейшем используется при построении выражений и визуальном отображении состава события. Поле «Объект результата» должно определять тип результирующего объекта, который будет создан после выполнения SQL-запроса. В поле «Тopic результата» администратор сможет указать имя результирующего топика Kafka. Поле «Описание» будет предназначено для текстового пояснения назначения события и логики его обнаружения.

Флажок «Временная совместимость» позволит указать, может ли событие в дальнейшем использоваться в конструкторе временных отношений. Поле «Контекст» будет необходимо для выбора контекста события. Если у выбранного входного события уже указан контекст, он должен автоматически подставляться в данное поле. Если контекст у входного события отсутствует, нужно предоставить пользователю возможность выбрать его самостоятельно. Это необходимо для того, чтобы создаваемое событие могло использоваться в дальнейшем в конструкторе сложных составных событий, где требуется сопоставление событий, относящихся к одному контексту.

В правой части окна должна находиться панель, содержащая поле «Тип создания», где с помощью раскрывающегося списка можно будет выбрать, каким образом будет формироваться новое событие. Для агрегированного события используется один входной поток. В этом режиме в интерфейсе должен отображаться блок «Входное событие». Ниже будет размещаться таблица, в первом столбце которой будет находиться флажок выбора, а во втором

отображаться название события. Для агрегированного события будет допускаться выбор только одного входного события. Это ограничение необходимо, поскольку агрегированное событие строится поверх одного исходного потока.

Для сложного составного события в ручном режиме пользователь должен выбрать ровно два входных события, так как текущая реализация сложного составного события предполагает объединение только двух входов. После выбора входов пользователь сможет сформировать или изменить SQL-шаблон будущего события.

Для создания сложного составного события с использованием временных отношений будет предусмотрен отдельный режим конструктора Аллена. В нем будут использоваться поля «Первое событие», «Второе событие», «Отношение», «Мин. интервал, мин», «Макс. интервал, мин» и «Выражение». Поля «Первое событие» и «Второе событие» позволят выбрать два события, между которыми будет строиться временное отношение. В поле «Отношение» нужно предоставить возможность выбора с помощью раскрывающегося списка одного из четырех доступных отношений интервальной алгебры: BEFORE, AFTER, DURING или OVERLAPS. Поля минимального и максимального интервала будут задавать допустимую временную границу между событиями. Поле «Выражение» должно заполняться автоматически и показывать логическое выражение, автоматически сформированное на основе выбранных ранее параметров.

В нижней части окна должна находиться панель «SQL-шаблон события». В ней будет размещено многострочное поле для SQL-кода, в котором пользователь сможет просмотреть и при необходимости изменить сформированный SQL-запрос.

В нижней панели окна должны быть расположены кнопки «Создать черновик», «Создать и развернуть» и «Отмена». Кнопка «Создать черновик» позволит сохранить событие в базе метаданных без развертывания на Kafka и ksqlDB. Такой вариант необходим в случае, если администратор хочет

подготовить описание события и SQL-код, но еще не готов запускать его в работу. Кнопка «Создать и развернуть» должна выполнять сохранение события и сразу запускать процедуру развертывания. Если в процессе развертывания возникнет ошибка, она будет выводиться в нижней области окна. Текст ошибки будет приходить от ksqlDB и содержать подробное объяснение проблемы в SQL-запросе. Прототип окна создания события представлен на рисунке 24.

Рисунок 24 – Прототип окна создания нового события

Также стоит предусмотреть всплывающие окна просмотра SQL-кода входов, которые открываются при нажатии на кнопку «SQL входа». Данная кнопка будет отображаться напротив выбранного входа для нового события на окне его создания. В всплывающем окне должно отображаться выбранное событие и его SQL-код. Такое окно нужно для того, чтобы администратор мог посмотреть, какие поля и какая логика уже используются во входном событии, не покидая окно создания нового события.

При открытом окне просмотра SQL-кода изменение входных событий блокируется, чтобы пользователь случайно не сменил вход в момент анализа его SQL-запроса. Прототип окна отображения SQL кода представлен на рисунке 25.

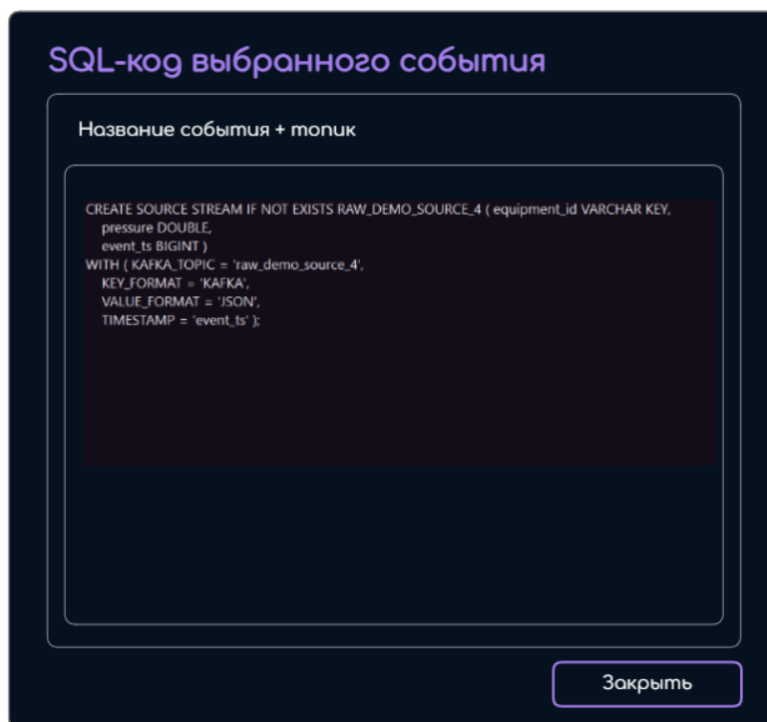


Рисунок 25 – Прототип окна отображения SQL кода

Если событие было создано, но при попытке развертывания возникла ошибка, приложение не должно позволять исправить параметры события. В этом случае администратору будет предоставлена возможность по кнопке «Редактировать событие» на окне событий перейти в режим редактирования уже созданного черновика. Благодаря этому администратор сможет исправить SQL-код или описание события и снова нажать кнопку развертывания. Такой подход делает обработку ошибок более простой и удобной.

По структуре окно редактирование должно повторять окно создания события, однако часть полей в нем должна быть заблокирована. Это необходимо для того, чтобы пользователь не изменил базовые свойства черновика, от которых зависит его место в дереве событий и связи с входными потоками. В режиме редактирования доступными должны остаться поля «Имя события», «Топик результата», «Описание» и многострочное поле «SQL-шаблон события». Также пользователь сможет пользоваться кнопками просмотра SQL-кода входных событий, чтобы сверить редактируемый запрос с запросами входов. В нижней части окна редактирования будут использоваться те же кнопки, что и при создании события.

Кнопка «Создать черновик» в данном режиме фактически будет выполнять сохранение изменений черновика, а кнопка «Создать и развернуть» будет сохранять изменения и запускать развертывание события. Таким образом, одно и то же окно будет использоваться и для создания, и для исправления ранее сохраненных, но еще не развернутых событий.

Задача панели оператора данных должна заключаться в возможности наблюдения за уже развернутыми событиями, просмотра входящих сообщений и анализа истории поступления данных. Прототип главного окна панели оператора данных представлен на рисунке 26.

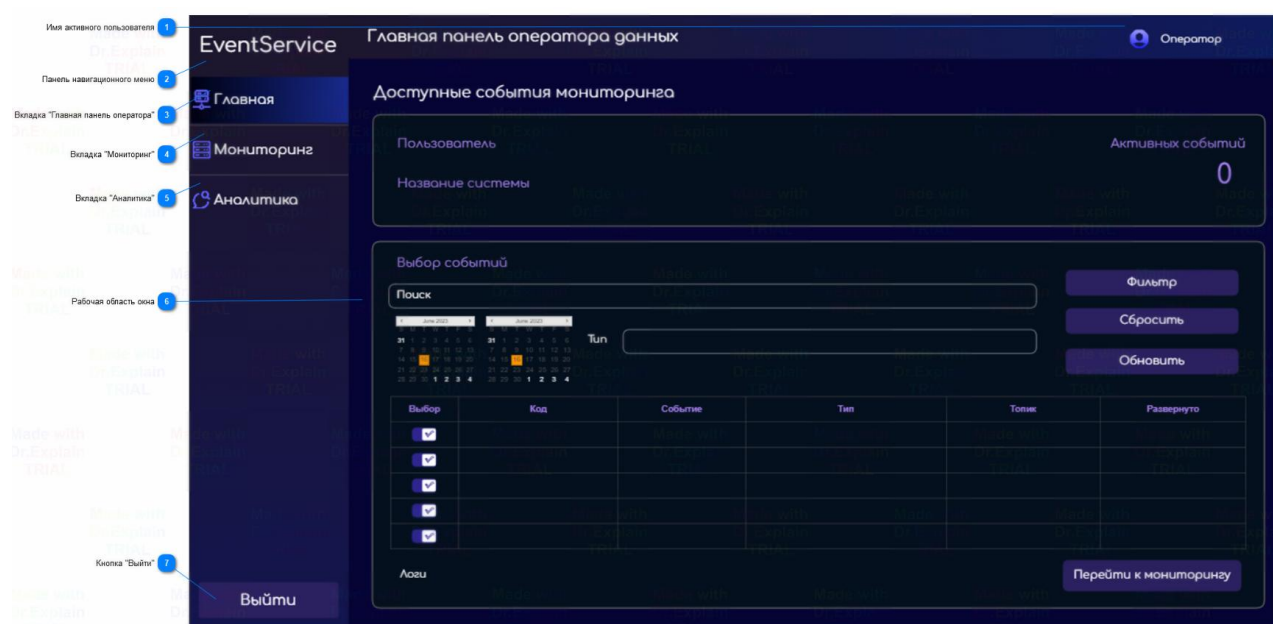


Рисунок 26 – Прототип главного окна панели оператора

Интерфейс главного окна панели оператора, представленный на рисунке 26, предусматривает следующий набор элементов:

- 1 – имя активного пользователя;
- 2 – навигационное меню;
- 3 – вкладка «Главная»;
- 4 – вкладка «Мониторинг»;
- 5 – вкладка «История и аналитика»;
- 6 – рабочая область окна;
- 7 – кнопка «Выйти».

Имя активного пользователя должно отображаться в верхнем правом углу главного окна. В левой части главного окна панели оператора будет расположено навигационное меню, состоящее из трех вкладок. Также в нижней части меню должна быть расположена кнопка «Выйти».

На вкладке «Главная» будут отображаться доступные для мониторинга события. В верхней панели должны выводиться сведения о текущем пользователе, системе и количестве активных на данный момент событий. Основную часть рабочей области будет занимать таблица событий, развернутых на Kafka. В этой таблице оператор сможет отметить от одного до десяти событий, по которым в дальнейшем будет запущен мониторинг. Для выбора нужно создать столбец с флажками-чекбоксами. В остальных столбцах будет отображаться краткий код события, его имя, тип, связанный топик и дата развертывания. Над таблицей расположены элементы поиска и фильтрации: поле «Поиск», раскрывающийся список «Тип», а также кнопки «Фильтр», «Сбросить» и «Обновить список». Благодаря этому оператор может быстро найти нужное событие по названию, коду или другим доступным параметрам.

В нижней части рабочей области будет отображаться количество выбранных событий в поле «Выбрано». После выбора событий оператор сможет нажать кнопку «Перейти к мониторингу». Ограничение в десять событий для одновременного мониторинга необходимо для того, чтобы отслеживание оставалось читаемым и не превращалось в слишком плотный поток сообщений, который будет сложно визуализировать.

Вкладка «Мониторинг» будет предназначена для просмотра входящих сообщений в режиме реального времени. В верхней части рабочей области должен отображаться список выбранных событий и их выходных топиков, на которые будет оформлена подписка. Там же должны быть расположены кнопки «Старт», «Стоп» и «Очистить поток».

После нажатия кнопки «Старт» операторский клиент будет обращаться к серверу, сервер должен будет проверить выбранные события и определить результирующие топика Kafka. После этого между клиентом и сервером будет

устанавливаться WebSocket-соединение, по которому клиент сможет получать новые сообщения.

Основная часть рабочей области вкладки мониторинга будет разделена на две области. Слева должна быть расположена динамическая таблица «Входящие сообщения». В ней будут отображаться время получения сообщения, событие, топик, краткое содержимое сообщения, офсет и признак ошибки.

При выборе строки в таблице в нижней панели «Форматированное сообщение» данные должны отображаться в более удобном виде: указываются время получения, временная метка Kafka, событие, тип события, топик, число партиций, офсет, ключ, данные JSON и признак ошибки. Сообщение, являющееся JSON-объектом, сервер должен форматировать с переносами строк и выделять поля верхнего уровня. Это позволит оператору просматривать не сырые строки из Kafka, а более читаемое представление данных. При обнаружении очередного события в этом поле отображаются его данные.

В нижней части вкладки мониторинга должны отображаться динамические показатели текущей сессии мониторинга: «Всего сообщений», «Ошибок» и «Последнее сообщение». Нужно предоставить возможность очистки потока сообщений при помощи кнопки «Очистить поток», которая не будет прерывать саму сессию мониторинга. При нажатии на эту кнопку таблица с поступающими сообщениями очистится, и начнет отображать новые сообщения сразу после их поступления. Кнопка «Стоп» должна будет завершать WebSocket-подключение и останавливать чтение сообщений полностью.

После остановки мониторинга должен автоматически сформироваться отчет. Нужно обеспечить его сохранение в папку MonitoringReports в локальном хранилище приложения. Название файла должно формироваться с указанием точного времени начала и окончания мониторинга. В нижней строке логов будет выводиться сообщение о сохранении отчета. Такой файл позволяет после завершения наблюдения вернуться к полученным данным и использовать их для анализа или демонстрации. Прототип вкладки «Мониторинг» представлен на рисунке 27.

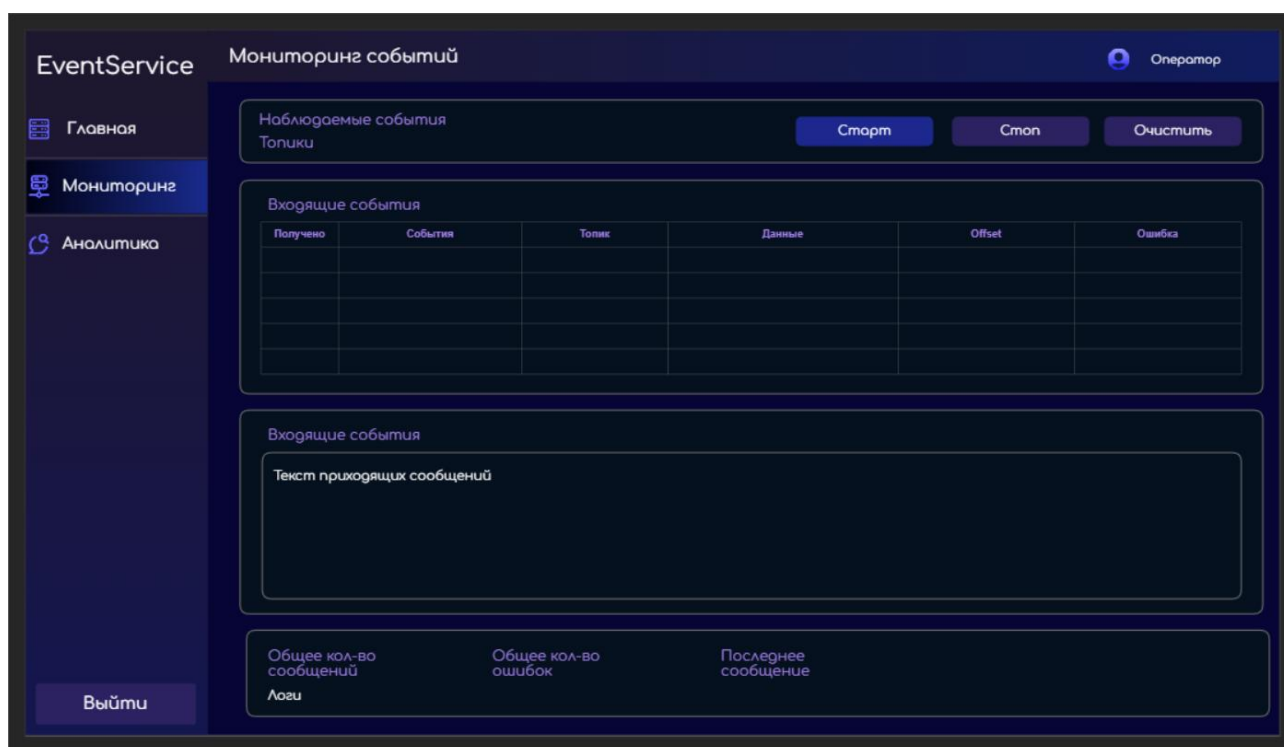


Рисунок 27 – Прототип вкладки «Мониторинг»

Вкладка «История и аналитика» будет предназначена для анализа сообщений за выбранный период. В верхней части рабочей области должен отображаться блок «Аналитика по событиям». В нем пользователь сможет задать период с точностью до минут. Для этого должны быть предусмотрены отдельные поля выбора даты и времени для начала и конца периода. Также оператор сможет указать интервал в минутах, по которому данные будут группироваться на временные промежутки. Например, если выбран интервал в десять минут, то система будет считать количество сообщений в каждом десятиминутном промежутке выбранного периода.

Слева в окне аналитики должна находиться таблица событий с флажками выбора. Над ней будет расположено поле поиска по названию или коду, позволяющее быстро найти нужное событие. Одновременно для построения аналитики можно выбрать не более десяти событий. Справа расположена область диаграмм с заголовком «Количество сообщений по событиям и общий поток по времени». В ней отображается распределение сообщений по выбранным событиям, а также общий поток сообщений по временным

интервалам. Диаграммы позволяют визуально оценить, какие события происходили чаще, и в какие моменты поток данных был наиболее активным.

В нижней панели вкладки аналитики отображаются основные показатели: «Всего сообщений», «Событий в отчёте» и «Пик за интервал». Показатель «Всего сообщений» отражает суммарное количество сообщений, найденных за выбранный период по выбранным событиям. Поле «Событий в отчёте» показывает количество событий, по которым были построены данные. Показатель «Пик за интервал» отображает максимальное количество сообщений, пришедших в один временной интервал, а также границы этого интервала. Например, значение «311, 12:30 – 12:40» означает, что максимальная нагрузка составила 311 сообщений в промежутке с 12:30 до 12:40. Если интервал попадает на переход между датами, в значении дополнительно отображается дата. Прототип вкладки «История и аналитика» представлен на рисунке 28.

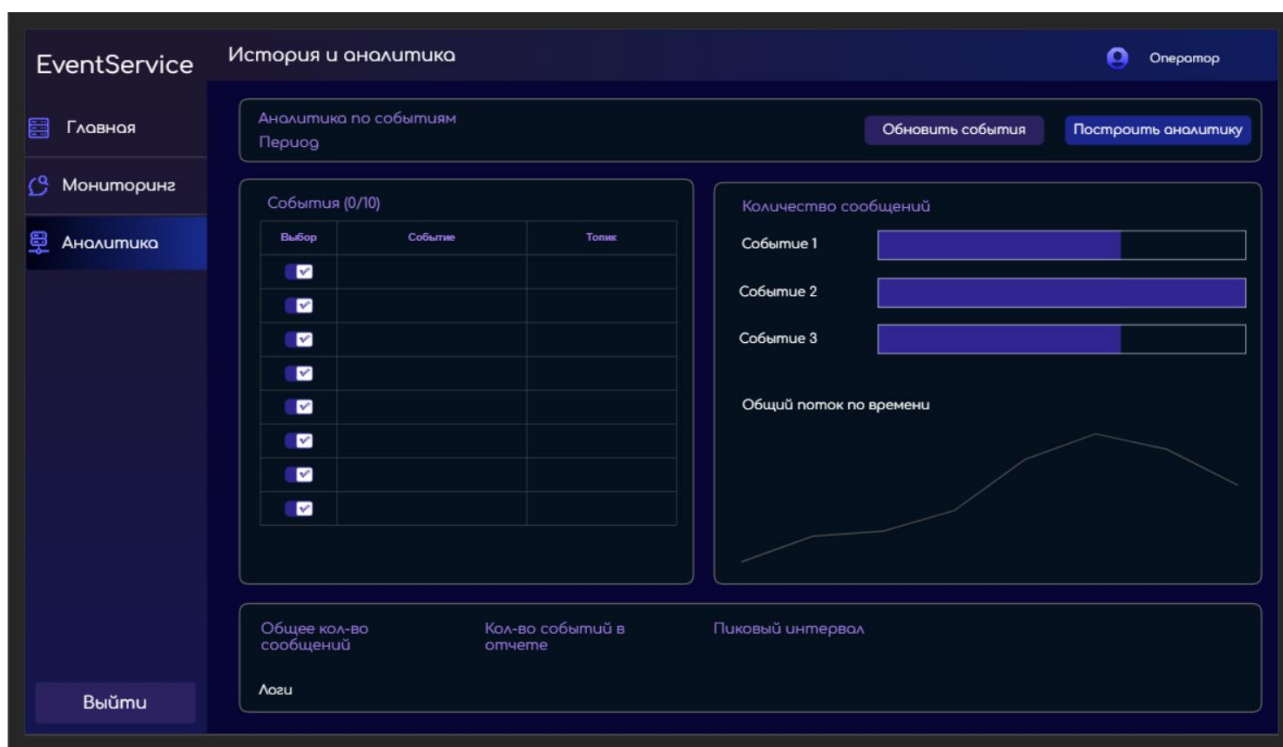


Рисунок 28 – Прототип вкладки «История и аналитика»

Разделение функций делает работу администратора и оператора безопасной и понятной, так как все возможные действия логично распределены и защищены правами доступа.

2.4 Анализ требований к аппаратно-программному обеспечению

2.4.1 Требования к аппаратному обеспечению

Аппаратное обеспечение – это физические ресурсы, включая процессоры, память, устройства и соответствующее микропрограммное обеспечение. Применительно к разрабатываемой системе администрирования и обнаружения событий аппаратное обеспечение включает физические компоненты серверного и клиентского уровней, обеспечивающие выполнение задач потоковой обработки данных [19].

Аппаратное обеспечение системы состоит из следующих ключевых компонентов:

- процессор;
- оперативная память;
- накопитель данных;
- графическая карта;
- сетевой адаптер.

Для развертывания разрабатываемой системы были установлены минимальные и рекомендуемые требования к аппаратному обеспечению серверной и клиентской частей.

Минимальные требования к серверной части (обеспечивают запуск Kafka, ksqlDB и компонентов системы в однопользовательском режиме для разработки и тестирования) отображены в таблице 10.

Таблица 10 – Минимальные требования к аппаратному обеспечению (сервер)

№ п/п	Компонент	Значение
1	Процессор	2 ядра, не менее 2.0 ГГц
2	Оперативная память	8 ГБ
3	Накопитель	10 ГБ свободного места (SSD)
4	Сетевой адаптер	100 Мбит/с

Рекомендуемые требования к серверной части (обеспечивают стабильную работу в многопользовательском режиме с промышленными объёмами потоков событий) отображены в таблице 11.

Таблица 11 – Рекомендуемые требования к аппаратному обеспечению (сервер)

№ п/п	Компонент	Значение
1	Процессор	4 ядра, не менее 2.4 ГГц
2	Оперативная память	16 ГБ
3	Накопитель	50 ГБ свободного места (SSD)
4	Сетевой адаптер	Интегрированная видеокарта с поддержкой OpenGL 3.0

Клиентская часть (рабочее место администратора или оператора данных) не требует специализированного оборудования – доступ к системе осуществляется через веб-интерфейс. Минимальные требования к клиентским приложениям представлены в таблице 12.

Таблица 12 – Минимальные требования к аппаратному обеспечению (клиент)

№ п/п	Компонент	Значение
1	Процессор	2 ядра, не менее 1.6 ГГц
2	Оперативная память	4 ГБ
3	Накопитель	500 МБ свободного места
4	Видеокарта	Интегрированная, с поддержкой разрешения 1366×768

Рекомендуемые требования к клиентским приложениям представлены в таблице 13.

Таблица 13 – Рекомендуемые требования к аппаратному обеспечению (клиент)

№ п/п	Компонент	Значение
1	Процессор	4 ядра, не менее 2.4 ГГц
2	Оперативная память	8 ГБ
3	Накопитель	1 ГБ свободного места
4	Видеокарта	Интегрированная, с поддержкой Full HD (1920×1080)

Такие требования позволяют обеспечить стабильную и надежную работу системы администрирования и обнаружения событий в среде потоковой обработки данных как на этапе разработки и отладки, так и в условиях промышленной эксплуатации.

2.4.2 Требования к программному обеспечению

Программное обеспечение (ПО) – это объект, состоящий из программ, процедур, правил, а также, если предусмотрено, сопутствующих им документации и данных, относящихся к функционированию системы обработки информации [20]. Применительно к разрабатываемой системе администрирования и обнаружения событий программное обеспечение

включает совокупность программных компонентов, обеспечивающих приём, обработку, анализ потоков событий и предоставление результатов пользователю.

Программное обеспечение системы подразделяется на следующие категории:

- системное ПО – комплекс программ, которые управляют аппаратной частью компьютера и создают среду для других программ;
- прикладное ПО – программы для решения конкретных задач пользователя и автоматизации его работы;
- инструментальное ПО – комплекс средств для разработки, отладки, тестирования и сопровождения ПО на всех этапах его жизненного цикла.

Системное программное обеспечение, необходимое для развертывания серверной части системы, представлено в таблице 14.

Таблица 14 – Программные требования для разработки системы

№ п/п	Компонент	Значение
1	Операционная система	Window 11 – обеспечивает стабильную работу контейнеров и сервисов потоковой обработки
2	Docker	Платформа контейнеризации для развёртывания Kafka, ksqlDB и серверной части REST API
3	PostgreSQL	Реляционная СУБД для хранения метаданных (топики, события, пользователи)
4	Apache Kafka	Распределенная платформа потоковой передачи данных, брокер событий
5	ksqlDB	Движок потоковой обработки для обнаружения событий, агрегаций и аналитики
6	.NET 10 Windows Desktop Runtime.	Среда выполнения для серверной части REST API, разработанной на C#
7	Visual Studio 2026	Интегрированная среда разработки (для отладки и сборки)

Для взаимодействия с панелью администратора и панелью мониторинга оператора требуется современный веб-браузер с поддержкой WebSocket.

Перечисленные требования к системному ПО обеспечивают полную функциональность развёртывания и эксплуатации клиентской и серверной частей системы в среде потоковой обработки данных.

2.4.3 Требования к средствам разработки

Средства разработки – это совокупность программных инструментов и приложений, обеспечивающих процессы создания, тестирования, отладки, развёртывания и сопровождения программных продуктов.

К основным категориям средств разработки относятся:

- редакторы кода и интегрированные среды разработки (IDE);
- компиляторы и интерпретаторы;
- отладчики;
- средства визуального проектирования интерфейсов;
- фреймворки и библиотеки;
- инструменты автоматизированного тестирования;
- системы управления версиями.

Применение современных средств разработки позволяет повысить производительность труда разработчика, качество конечного продукта и эффективность организации работы над проектом.

Для реализации программных средств администрирования и обнаружения событий в системе потоковой обработки данных» были выбраны следующие языки программирования:

- C# – основной язык для реализации серверной части (REST API) и клиентского приложения (графический интерфейс на Windows Forms);
- SQL – для работы с метаданными в PostgreSQL;
- ksqlDB (SQL-подобный язык) – для описания логики обнаружения событий, потоковых запросов и агрегаций.

C# выбран благодаря высокой производительности, строгой типизации, развитой экосистеме для создания микросервисов и настольных приложений, а также удобству интеграции с Kafka и REST API.

В качестве основной среды разработки использована Microsoft Visual Studio 2026, предоставляющая мощные средства для написания кода, отладки, профилирования, интеграции с Git и управления пакетами NuGet.

В процессе разработки задействованы следующие фреймворки и библиотеки:

- ASP.NET Core – для создания RESTful API, обеспечивающего администрирование событий, управление топиками, запросы к ksqlDB, аутентификацию и авторизацию;

- Windows Forms – классическая платформа от Microsoft для создания графических интерфейсов (GUI) в среде .NET, предназначенная для разработки настольных приложений под Windows (панель администратора, панель мониторинга оператора);
- Confluent.Kafka – официальный .NET-клиент для взаимодействия с Apache Kafka (публикация событий);
- Npgsql – провайдер для подключения к PostgreSQL.

Выбор указанных средств обоснован сравнительным анализом альтернатив (Java, Spring Boot, Python, FastAPI): C# в связке с Windows Forms и ASP.NET Core обеспечивает наилучшую производительность, единую экосистему для серверной и клиентской частей, а также удобство работы с потоковыми данными через Kafka и ksqlDB.

Применение данных технологий позволяет в полной мере реализовать функциональные требования к администрированию, обнаружению событий, мониторингу в реальном времени и масштабированию разрабатываемой системы.

2.5 Постановка задачи на разработку программных средств

Разрабатываемые программные средства предназначены для администрирования и обнаружения событий в системе потоковой обработки данных на платформе Apache Kafka. Система будет обеспечивать управление потоковой инфраструктурой, конструирование событий и мониторинг потоков данных в реальном времени.

Компоненты системы включают в себя Apache Kafka, который представляет собой распределённый брокер сообщений для приёма, хранения и маршрутизации потоков событий. ksqlDB Server служит движком потоковой обработки, обеспечивающим обнаружение событий на основе непрерывных SQL-подобных запросов, агрегацию потоковых данных и формирование результирующих потоков и таблиц.

REST API на базе ASP.NET Core выступает серверным интерфейсом для администрирования системы, предоставляя управление источниками данных, топиками и событиями, а также выполнение запросов к ksqlDB. База метаданных PostgreSQL используется для хранения информации об источниках, топиках, зарегистрированных событиях, пользователях, ролях, статусах объектов и прочих настройках системы.

Язык C# применяется для разработки серверной части REST API и клиентских приложений, а SQL-подобные запросы ksqlDB используются для описания правил обнаружения и обработки событий.

Apache Kafka позволит осуществлять надёжную передачу потоков событий между источниками и потребителями данных. ksqlDB Server обеспечит непрерывную обработку потоков, обнаружение событий по заданным правилам и предоставление результатов через собственный API. REST API будет выполнять функции аутентификации и авторизации пользователей, просмотра источников, топиков и событий, активации и деактивации источников данных, конструирования событий, развертывания событий, а также отправки запросов к ksqlDB Server.

Панель администратора, реализованная на Windows Forms, предоставит русскоязычный графический интерфейс для конструирования событий, управления источниками данных, работы с топиками, просмотра метаданных и контроля статусов объектов системы. Через данную панель администратор сможет создавать черновики событий, разворачивать события, удалять неразвернутые черновики и удалять активные события при условии, что они не используются в составе других сложных событий.

Операторская панель мониторинга будет реализована на Windows Forms и обеспечит наблюдение в реальном времени за событиями, поступающими через WebSocket. В ее задачи войдёт отслеживание динамики потоков и формирование аналитических отчетов. Оператору будет предоставлена возможность выбирать отслеживаемые события, просматривать поступающие сообщения в структурированном формате, а также приостанавливать мониторинг с

сохранением сформированного отчета. Дополнительно будет доступен анализ истории сообщений за произвольный временной интервал.

Должна быть учтена расширяемость перечня событий: добавление новых типов не должно приводить к снижению производительности основных операций. Интерфейсы обеих панелей должны проектироваться удобными и интуитивно понятными. После внедрения системы будет необходимо обеспечивать ее работоспособность, проводить мониторинг эксплуатационных характеристик и сохранять возможность добавления нового функционала при необходимости.

3 Разработка программных средств

3.1 Разработка архитектуры программных средств

Архитектура разрабатываемых программных средств включает в себя набор различных компонентов. Для их представления была выбрана диаграмма развертывания. Она представляет собой графическое отображение архитектуры системы, иллюстрирующее компоненты программного обеспечения и их распределение по узлам или физическим машинам. Она показывает, какие программные модули разворачиваются на каждом узле сети и как наложено взаимодействие между этими модулями. Диаграмма развертывания для программных средств администрирования, обнаружения и мониторинга событий отображена на рисунке 29.

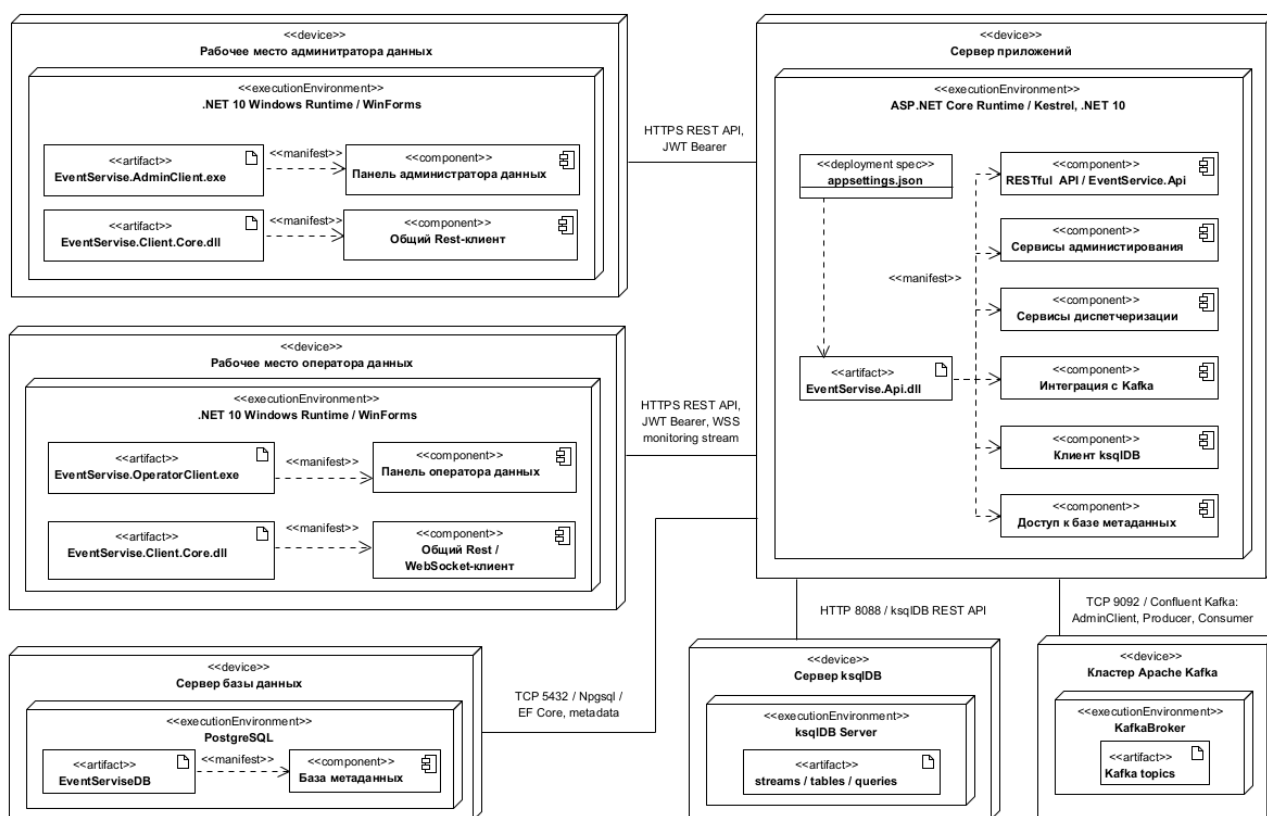


Рисунок 29 – Диаграмма развертывания программных средств

На диаграмме развертывания отображены следующие основные узлы:

- рабочее место администратора данных;
- рабочее место оператора данных;

- сервер приложения;
- сервер базы данных;
- сервер ksqlDB;
- кластер Apache Kafka.

Рабочее место администратора данных представляет собой устройство, на котором выполняется среда исполнения .NET 10 Windows Runtime с WinForms. Артефакт EventService.AdminClient.exe реализует компонент «Панель администратора данных», предназначенный для выполнения функций администратора. Артефакт EventService.Client.Core.dll реализует компонент «Общий REST-клиент», обеспечивающий взаимодействие клиентского приложения с сервером приложений по протоколу HTTPS REST API с использованием JWT Bearer-аутентификации. Через рабочее место администратора выполняется работа с источниками данных, топиками, правилами обнаружения событий.

Рабочее место оператора данных также представляет собой устройство, на котором выполняется среда исполнения .NET 10 Windows Runtime с WinForms. Артефакт OperatorClient.exe реализует компонент «Панель оператора данных», предназначенный для наблюдения за поступающими событиями. Артефакт EventService.Client.Core.dll реализует компонент «Общий REST / WebSocket-клиент», который обеспечивает обмен данными с сервером приложений через HTTPS REST API, а также получение мониторинговой информации в реальном времени через WebSocket.

WebSocket – это сетевой протокол, который устанавливает постоянное, непрерывное двустороннее соединение между браузером (клиентом) и сервером. В отличие от обычного HTTP, где клиент должен каждый раз отправлять новый запрос для получения ответа, WebSocket открывает канал связи один раз и позволяет мгновенно обмениваться данными в обоих направлениях [21]. Передавать сообщения из потоков с помощью стандартных запросов HTTP API является нецелесообразным, поскольку такой подход вынуждает клиента постоянно опрашивать сервер, создавая избыточную нагрузку на сеть и

вычислительные ресурсы. Это увеличило бы задержку доставки сообщений и не обеспечило работы в реальном времени. Каждый HTTP-запрос требует повторной установки соединения и не позволяет серверу инициировать передачу данных самостоятельно, то есть клиент был бы вынужден постоянно запрашивать обновления, даже если новых сообщений в потоке нет. WebSocket решает эту проблему, так как после однократного установления соединения через постоянный двунаправленный канал сервер может мгновенно отправлять клиенту каждое новое сообщение из потока, как только оно появляется. Это сильно снижает расходы, исключает задержки опроса и обеспечивает эффективную и непрерывную передачу данных.

Сервер приложений является центральным узлом программной системы. На нем развернута среда исполнения ASP.NET Core Runtime / Kestrel, .NET 10, в рамках которой размещается основной серверный артефакт EventService.Api.dll. Данный артефакт реализует несколько компонентов: RESTful API / EventService.Api, сервисы администрирования, сервисы диспетчеризации, интеграцию с Kafka, клиент ksqlDB и доступ к базе метаданных. RESTful API обеспечивает прием и обработку запросов от клиентских приложений.

Сервисы администрирования отвечают за выполнение операций по настройке объектов системы, а сервисы диспетчеризации обеспечивают работу функций мониторинга. Компонент интеграции с Kafka используется для взаимодействия с брокерами Apache Kafka, компонент «Клиент ksqlDB» обеспечивает работу с сервером ksqlDB, а компонент «Доступ к базе метаданных» отвечает за обмен данными с PostgreSQL. Файл appsettings.json представлен как спецификация развертывания, содержащая параметры подключения и конфигурации серверного приложения.

Сервер базы данных предназначен для хранения служебной и конфигурационной информации системы. На данном узле функционирует среда исполнения PostgreSQL, внутри которой размещен артефакт EventServiceDB. Он реализует компонент «База метаданных».

Кластер Apache Kafka является узлом потоковой передачи сообщений. В его составе представлена среда исполнения KafkaBroker, внутри которой размещен артефакт Kafka topics. Он соответствует топикам Kafka, используемым для передачи сообщений и обнаружения событий и взаимодействует с кластером Apache Kafka по протоколу TCP 9092 с использованием Confluent.Kafka. Включает в себя функции AdminClient, Producer и Consumer.

Для потоковой обработки поверх Apache Kafka был развёрнут сервер ksqlDB. На нем развернута среда исполнения ksqlDB Server, внутри которой размещен артефакт streams / tables / queries. Данный артефакт отражает используемые потоки, таблицы и запросы ksqlDB, необходимые для обработки данных и обнаружения событий. Серверная часть программных средств взаимодействует с сервером ksqlDB через HTTP 8088 / ksqlDB REST API, а сам ksqlDB обменивается данными с кластером Apache Kafka по Kafka protocol, осуществляя чтение и запись потоков, таблиц и топиков.

Представленная диаграмма демонстрирует распределение программных средств между рабочими местами пользователей и серверными узлами. Архитектура обеспечивает четкое разделение функций администрирования и диспетчеризации. Администратор получает инструменты настройки потоковой инфраструктуры, а оператор получает средства наблюдения за событиями в реальном времени.

Подобная реализация объединения потоковой обработки и обнаружения сложных событий в едином программном продукте дает несколько важных преимуществ. Прежде всего, администратору не требуется вручную работать с низкоуровневыми командами Kafka и ksqlDB. Основные действия выполняются через графический интерфейс: выбор входов, генерация шаблона, сохранение черновика, развертывание и контроль ошибок.

Событие становится не только SQL-запросом, но и управляемым объектом системы. Для него хранятся метаданные, статус, история изменений и дерево состава.

Результирующие события можно повторно использовать как входы для новых событий. Это позволяет строить многоуровневые схемы обнаружения: от простых фактов к агрегированным событиям, затем к сложным составным ситуациям. Такой подход особенно важен для предметных областей, где значимое событие редко выражается одним сообщением. Чаще оно проявляется как последовательность или комбинация нескольких фактов.

Система разделяет роли администратора и оператора. Администратор отвечает за источники, топики, события, SQL-правила и развертывание, а оператор наблюдает за результатами, не имея возможности изменить конфигурацию. Это снижает риск случайного нарушения потоковой обработки и делает эксплуатацию системы безопаснее.

Текущая реализация имеет ряд ограничений. Источники данных считаются заранее зарегистрированными. Через панель администратора можно активировать и деактивировать передачу данных, но не предусмотрено полноценное добавление новых физических источников или настройка внешних коннекторов.

Сложное составное событие в ручном режиме требует выбора ровно двух входных событий. Конструктор Аллена также работает с двумя событиями и ограниченным набором временных отношений. Для корректного объединения событий требуется общий контекст. Кроме того, редактирование доступно только для неразвернутых черновиков. Активное событие нельзя свободно изменить, поскольку оно уже связано с `ksqlDB`-запросом и результирующим Kafka-топиком. Удаление активного события допускается только в том случае, если оно не используется в составе других сложных событий. Это ограничение защищает систему от разрушения зависимостей в дереве событий.

В текущей версии система ориентирована на сообщения в формате JSON и на работу с теми полями, которые используются в `kSQL`-шаблонах. Полноценное управление схемами сообщений и автоматическая проверка совместимости схем не выделены в отдельный механизм. Эти ограничения не нарушают основную логику системы, но определяют границы ее текущего применения.

Несмотря на указанные ограничения, выбранная архитектура является практичной для системы администрирования и обнаружения событий. Kafka обеспечивает надежную передачу потоковых данных, ksqlDB выполняет непрерывную обработку декларативными средствами, REST API связывает инфраструктуру с пользовательскими приложениями, а PostgreSQL хранит метаданные всех объектов наблюдаемой системы. В результате администратор получает инструмент для конструирования и развертывания событий, а оператор – средство наблюдения и анализа результатов потоковой обработки. Система превращает набор разрозненных инфраструктурных компонентов в единый управляемый продукт, предназначенный для эксплуатации в прикладной предметной области.

3.2 Программные средства обнаружения сложных событий в системе потоковой обработки данных Apache Kafka

3.2.1 Основные конструкции языка kSQL для реализации программных средств обнаружения событий

Разрабатываемые программные средства объединяют в себе функции администрирования потоковой инфраструктуры и обнаружения значимых событий в едином продукте. Их основой является взаимосвязанная работа Apache Kafka и ksqlDB сервера. Apache Kafka используется как распределенная платформа передачи и хранения сообщений в потоках, а ksqlDB Server выполняет обработку этих сообщений при помощи декларативных SQL-подобных запросов. Такой подход позволяет описывать правила обнаружения событий не в виде сложного программного кода, а в виде kSQL-скриптов, которые разворачиваются в потоковой среде и выполняются непрерывно.

Язык kSQL представляет собой SQL-подобный декларативный язык, предназначенный для работы с потоковыми данными Apache Kafka. В отличие от классического SQL, который применяется к уже сохраненным данным, kSQL описывает непрерывные операции над событиями, поступающими в Kafka-топики.

В современной документации Confluent чаще используется название ksqlDB. Под ним понимается потоковая SQL-база данных, работающая поверх Apache Kafka. ksqlDB предоставляет возможность описывать потоковые приложения с помощью привычного синтаксиса SQL, не реализуя отдельные приложения на Kafka Streams вручную. ksqlDB не заменяет Kafka, а использует ее как основу для хранения и передачи событий. Топики Kafka выступают источниками и приемниками данных, а ksqlDB задает правила для их непрерывной обработки.

Основными абстракциями языка kSQL являются поток и таблица. Поток отражает последовательность событий, поступающих во времени. Каждое новое сообщение воспринимается как новый факт. Таблица, в свою очередь, представляет текущее состояние данных по ключу. Если в потоке каждое сообщение добавляется как новый элемент, то в таблице новое сообщение с тем же ключом обновляет предыдущее состояние. Потоки удобно использовать для фиксации фактов, а таблицы нужны для хранения последнего состояния объекта или результата агрегации [22].

В таблице 15 представлены основные операции языка kSQL, используемые для обработки данных в потоках.

Таблица 15 – Основные операции языка kSQL

№ п/п	Операция	Назначение
1	CREATE STREAM	Регистрирует поток поверх Kafka-топика. Используется, когда сообщения топика должны рассматриваться как последовательность событий с заданной схемой.
2	CREATE TABLE	Регистрирует таблицу поверх Kafka-топика. Используется для представления текущего состояния данных по ключу.
3	CREATE STREAM AS SELECT	Создает новый поток на основе результата запроса к существующему потоку или таблице. Обычно используется для фильтрации, преобразования и формирования новых событий.
4	CREATE TABLE AS SELECT	Создает новую таблицу на основе результата запроса. Часто применяется для агрегаций, подсчета количества сообщений, вычисления средних, максимумов и других итоговых показателей.
5	SELECT	Позволяет выбрать нужные поля, выполнить преобразования, вычислить выражения и задать структуру результирующего сообщения.
6	WHERE	Фильтрует события по заданному условию. Например, может использоваться для отбора сообщений, в которых температура вышла за допустимые границы.
7	GROUP BY	Группирует сообщения по одному или нескольким полям. Обычно применяется вместе с агрегатными функциями.
8	HAVING	Фильтрует результат после группировки. Используется, когда нужно отобрать только те группы, которые удовлетворяют условию по агрегированному значению.

Продолжение таблицы 15

N п/п	Операция	Назначение
9	WINDOW	Задаёт временное окно для потоковой агрегации. Может использоваться для подсчёта сообщений за минуту, пять минут, час и другие интервалы.
10	JOIN	Объединяет данные из двух потоков, таблиц или потока и таблицы. В системах обнаружения сложных событий применяется для сопоставления нескольких фактов по общему ключу.
11	WITHIN	Задаёт временное ограничение при объединении потоков. Позволяет указать, что два события должны произойти в пределах заданного интервала.
12	PARTITION BY	Изменяет ключ партиционирования результирующего потока. Важен для корректного сопоставления событий по общему идентификатору, например, по correlation_id.
13	EMIT CHANGES	Указывает, что запрос должен непрерывно выдавать изменения по мере поступления новых событий. Используется в постоянных потоковых запросах.
14	EMIT FINAL	Используется для оконных агрегаций, когда результат должен быть выдан после закрытия окна, а не при каждом изменении промежуточного состояния.
15	INSERT INTO	Записывает результат запроса в уже существующий поток. Может применяться, когда несколько правил обработки должны направлять данные в один результирующий поток.
16	DROP STREAM	Удаляет зарегистрированный поток из ksqlDB. При необходимости может также использоваться с удалением связанного Kafka-топика.
17	DROP TABLE	Удаляет зарегистрированную таблицу из ksqlDB.
18	TERMINATE	Останавливает постоянный запрос по его идентификатору. Используется при удалении или остановке ранее развернутой потоковой логики.
19	SHOW STREAMS	Выводит список зарегистрированных потоков.
20	SHOW TABLES	Выводит список зарегистрированных таблиц.
21	DESCRIBE	Показывает схему и сведения о выбранном потоке или таблице.
22	PRINT	Позволяет просматривать сообщения Kafka-топика в ksqlDB, что удобно для проверки входных данных и отладки.

Для того, чтобы развернуть на Kafka агрегированные или составные события чаще всего используются операции CREATE STREAM, CREATE STREAM AS SELECT, JOIN, WHERE, PARTITION BY, WITHIN и EMIT CHANGES. Функционал, предоставляемый языком kSQL хорошо подходит для выбранной архитектуры, так как он позволяет описывать правила обнаружения событий декларативно, а выполнение этих правил переносится на потоковую инфраструктуру Kafka и ksqlDB.

Разработанный объект инфраструктуры ksqlDB Server подключается к Kafka и получает доступ к ее топикам. Основная задача ksqlDB Server в разрабатываемой системе состоит в том, чтобы выполнять непрерывные SQL-подобные запросы над потоками данных. Когда администратор создает событие и нажимает кнопку развертывания, REST API отправляет сформированный kSQL-скрипт на сервер ksqlDB через его HTTP API, транспортный протокол HTTP. После успешного выполнения запроса ksqlDB создает запускает постоянный запрос. Этот запрос продолжает работать после выполнения

команды и обрабатывает все новые сообщения, поступающие во входные топики.

Таким образом, развертывание события в системе потоковой обработки данных Apache Kafka представляет собой не разовую обработку данных или выполненное единожды вычисление, а создание нового постоянно работающего правила для обработки данных. Если обычный SQL-запрос к базе данных возвращает результат только на момент обращения, то kSQL-запрос в ksqlDB задает правило, по которому поступающие сообщения будут обрабатываться непрерывно вплоть до момента, пока событие не будет удалено. Результаты обработки записываются в результирующий Kafka-топик. В дальнейшем этот топик может использоваться оператором для мониторинга или администратором как вход для построения других сложных событий.

3.2.2 Программные средства обнаружения элементарных событий

В текущей версии разработки предусмотрены три основных уровня событий. Первый уровень составляют элементарные события. Они соответствуют потокам исходных данных, поступающих от источников.

Например, в системе фармацевтической компании источник температуры холодильной цепи может публиковать сообщения в созданный для этого топик RAW_PHARMA_COLD_CHAIN_SOURCE_6. Соответствующее элементарное событие может иметь краткий код COLD_TEMP. Такое событие не является результатом сложной обработки. Оно только представляет исходный поток в терминах предметной области системы.

На уровне ksqlDB элементарному событию, как правило, соответствует поток, созданный командой CREATE STREAM поверх указанного топика. Это позволяет системе единообразно ссылаться на событие по его коду во всех последующих запросах. Администратор выполняет такое связывание через графический интерфейс, не прибегая к прямому написанию ksqlDB-команд. Программные средства обеспечивают автоматическое создание потока и топика при первичной активации нового источника. После создания элементарное

событие появляется в общем списке событий и может быть выбрано в качестве входа при конструировании агрегированных и составных событий.

Для элементарного события система автоматически сгенерирует скрипт, основываясь на структуре данных, приходящих от источника.

SQL-скрипт запроса для создания потока значений с датчика температуры холодильной цепи представлен в листинге 1.

Листинг 1 – Пример kSQL-запроса для создания сложного агрегированного события

```
CREATE STREAM RAW_PHARMA_COLD_CHAIN_SOURCE_6 (
  sensor_id    VARCHAR,
  container_id VARCHAR,
  temperature  DOUBLE,
  event_ts     BIGINT
) WITH (
  KAFKA_TOPIC   = 'RAW_PHARMA_COLD_CHAIN_SOURCE_6',
  VALUE_FORMAT = 'JSON',
  TIMESTAMP     = 'event_ts'
);
```

Путем развертывания такого запроса одновременно создается и топик, в который будут записываться приходящие сообщения от датчика температуры, и поток, который можно будет использовать при дальнейшем конструировании сложных событий.

3.2.3 Программные средства обнаружения сложных агрегированных событий

Второй уровень составляют сложные агрегированные события. В текущей реализации под агрегированным событием понимается событие, формируемое на основе одного входного события или одного входного потока. Пользователь выбирает одно входное событие, задает основные метаданные нового события и формирует SQL-шаблон. Этот шаблон может выполнять фильтрацию, преобразование полей, агрегацию или другие вычисления.

Например, на основе поступающих данных о температуре можно создать событие, возникающее при выходе значения температуры за допустимый уровень.

Для такого события kSQL-скрипт может иметь вид, представленный в листинге 2.

Листинг 2 – Пример kSQL-запроса для создания сложного агрегированного события

```
CREATE STREAM AGG_PH_TEMP_OUT_OF_RANGE
WITH (
    KAFKA_TOPIC = 'AGG_PH_TEMP_OUT_OF_RANGE',
    VALUE_FORMAT = 'JSON'
) AS
SELECT
    correlation_id,
    sensor_id,
    temperature,
    event_time
FROM RAW_PHARMA_COLD_CHAIN_SOURCE_6
WHERE temperature < 2 OR temperature > 8
EMIT CHANGES;
```

В этом примере результирующий поток AGG_PH_TEMP_OUT_OF_RANGE будет получать только те сообщения, в которых температура выходит за допустимые границы [23]. Все остальные сообщения будут отфильтрованы. При поступлении нового сообщения во входной Kafka-топик ksqlDB автоматически применяет к нему заданное условие. Если условие истинно, сообщение попадает в результирующий топик события.

3.2.4 Программные средства обнаружения составных событий

Третий уровень составляют сложные составные события. Они строятся на основе двух входных событий. В ручном режиме администратор выбирает ровно два входа и формирует kSQL-запрос, который объединяет их по общему признаку, например, по идентификатору корреляции. Такой подход позволяет выявлять ситуации, которые не могут определены по сырому потоку данных.

Например, если в системе зафиксируется отключение питания, а затем в течение следующих нескольких минут возникнет изменение температуры, такая совокупность условий может рассматриваться как событие риска порчи товара. Для обнаружения такого события необходимо использовать отфильтрованные данные от двух источников – датчика температуры и датчика питания. Каждое

из этих устройств генерирует элементарный поток данных. Далее каждый из них обрабатывается, благодаря чему становится возможным обнаружение двух отдельных событий: «Фиксация отключения питания» и «Значение температуры вышла за допустимый уровень». Объединив эти два события и выставив временное окно в шесть минут, можно обнаружить событие «Риск порчи товара», происходящее в следствии отключения питания и критического изменения температуры.

Схематичное представление входных топиков, потоков элементарных и агрегированных событий, а также их выходных топиков, необходимых для конструирования результирующего составного события, представлено на рисунке 30.



Рисунок 30 – Схема конструирования составного события

В результате создается поток для сложного составного события и соответствующий ему топик, куда приходят сообщения в случае обнаружении риска порчи товара.

Для такого события kSQL-скрипт может иметь вид, представленный в листинге 3.

Листинг 3 – Пример kSQL-запроса для создания сложного составного события

```
CREATE STREAM CS_TEMP_AFTER_POWER_RISK
WITH (
    KAFKA_TOPIC = 'CS_TEMP_AFTER_POWER_RISK',
    VALUE_FORMAT = 'JSON'
) AS
SELECT
    power.correlation_id,
    power.event_time AS power_time,
    temp.event_time AS temp_time,
    temp.temperature,
    'TEMP_AFTER_POWER_RISK' AS event_code
FROM AG_PH_POWER_DISCONNECTED_T power
INNER JOIN AG_PH_TEMP_OUT_OF_RANGE_T temp
    WITHIN 6 MINUTES
    ON power.correlation_id = temp.correlation_id
EMIT CHANGES;
```

Здесь два входных события объединяются по correlation_id.

Дополнительно задается временное ограничение, в пределах которого события должны быть сопоставлены. То есть, если в пределах заданного временного интервала после отключения питания происходит температурное нарушение, то ksqlDB сервер сформирует новое сообщение и запишет его в результирующий топик сложного составного события. Именно так реализуется идея обнаружения сложных событий: система не просто хранит отдельные факты, а сопоставляет их во времени и по общему контексту.

Для построения части сложных составных событий в системе предусмотрен темпоральный конструктор временных отношений на основе интервальной алгебры Аллена. В этом режиме пользователь выбирает первое и второе событие, отношение между ними, минимальный и максимальный интервал. После этого автоматически формируется логическое выражение и SQL-шаблон. Например, отношение BEFORE означает, что первое событие должно произойти раньше второго, причем разница между ними должна попадать в заданный временной диапазон. Такой механизм необходим для описания сценариев вида «событие А произошло до события В в течение N минут».

Поскольку базовые отношения интервальной алгебры Аллена являются бинарными, в прототипе конструктор формирует одно составное событие на основе двух входных событий.

На данном этапе реализации администратору доступно использование четырех из тринадцати отношений из алгебры Аллена. Это BEFORE, AFTER, OVERLAPS и DURING. В таблице 16 представлено описание каждого из этих четырех отношений.

Таблица 16 – Отношения алгебры Аллена

Отношение	Смысл	Условие по интервалам
BEFORE	Событие А завершилось до начала В	$A.end < B.start$
AFTER	Событие А началось после окончания В	$A.start > B.end$
OVERLAPS	А началось раньше В, но закончилось уже после начала В	$A.start < B.start \text{ AND } A.end > B.start \text{ AND } A.end < B.end$
DURING	А полностью произошло внутри В	$B.start < A.start \text{ AND } A.end < B.end$

Для возможности использования событий в качестве входов для нового сложного составного события при помощи темпорального конструктора необходимо, чтобы оба входных события были потоками. Таблицы нельзя использовать в оконных функциях, то есть функциях, где сравниваются временные окна. Временное соединение через WITHIN применяется именно для соединения потоков. Соединения потока с таблицей называется обогащением, в `ksqlDB` являются неоконными и имеют другую семантику. Такого рода событие можно создать, используя режим ручного конструктора.

Если у входных потоков, которые планируется объединить в новое событие будут разные названия полей, это сильно усложнит автогенерацию шаблона запроса и превратит конструктор в сложный редактор схем. Поэтому было принято решение, что каждое событие, которое разрешено использовать в конструкторе алгебры Аллена, должно содержать три стандартных поля:

- `correlation_id` – идентификатор объекта, по которому события связываются;
- `interval_start` – начало события или интервала, BIGINT в миллисекундах;

– `interval_end` – окончание события или интервала, BIGINT в миллисекундах.

Администратор должен заранее привести события к подобному виду, после чего использовать их в конструкторе алгебры Аллена для создания нового события. Таким образом, операция объединения двух потоков JOIN автоматически будет проводиться по `correlation_id`, а сравнение временных отрезков по `interval_start` и `interval_end`. Для событий мгновенных, то есть тех, где нет времени начала и конца события, в качестве обоих значений `interval_start` и `interval_end` должна подставляться его единственная временная метка.

Пример агрегированного события, приведенного к виду, позволяющему использовать его в качестве входа для сложного составного события через темпоральный конструктор, представлен в листинге 4.

Листинг 4 – Пример агрегированного события, совместимого с конструктором Аллена

```
CREATE STREAM AG_PH_TEMP_OUT_OF_RANGE_T
WITH (
  KAFKA_TOPIC = 'ag_ph_temp_out_of_range_t',
  KEY_FORMAT = 'KAFKA',
  VALUE_FORMAT = 'JSON',
  PARTITIONS = 1,
  REPLICAS = 1,
  TIMESTAMP = 'interval_start'
) AS
SELECT
  correlation_id AS correlation_id,
  event_time AS interval_start,
  event_time + 60000 AS interval_end,
  sensor_id AS sensor_id,
  temperature AS temperature,
  event_time AS source_event_time,
  'TEMP_OUT_OF_RANGE' AS event_code
FROM RAW_PHARMA_COLD_CHAIN_SOURCE_6
WHERE temperature < 2 OR temperature > 8
PARTITION BY correlation_id
EMIT CHANGES;
```

Важную роль в такой модели играет контекст события. Контекст определяет, могут ли события быть сопоставлены друг с другом. Например, события, относящиеся к одному устройству, одному помещению или одной категории товара, должны иметь общий контекст. При создании нового события

система автоматически подставляет контекст события, чей результирующий топик используется в этом событии в качестве входного. Если контекст отсутствует, пользователь должен создать его вручную. Это ограничение нужно для корректной работы создания составных событий, так как без одинакового контекста сравнение и объединение потоков становится неоднозначным.

Когда администратор данных инициирует развертывание нового события на Kafka, REST API проверяет корректность переданных метаданных и входных событий. Затем она отправляет kSQL-запрос на ksqlDB Server. Если запрос выполнен успешно, ksqlDB создает потоковый объект и запускает постоянный запрос. После этого событие получает активный статус в базе метаданных, а его результирующий топик становится доступным для мониторинга и для использования в других событиях.

Если при развертывании возникает ошибка, например, из-за некорректного SQL-запроса, система сохраняет событие как черновик и выводит текст ошибки в интерфейсе, но не позволяет развернуть событие на Kafka.

Благодаря данной технологии становится возможным выявление значимого события посредством простого и удобного SQL-подобного языка прямо на непрерывных потоках данных, что на практике реализует идею объединения потоковой обработки и обнаружения сложных событий в едином программном продукте.

3.3 Разработка базы метаданных и ее интеграция с программными средствами администрирования и обнаружения событий

Метаданные – это структурированная информация, которая описывает, поясняет или облегчает использование других данных. Это «данные о данных», которые несут справочную, техническую или описательную функцию.

В разрабатываемых программных средствах база метаданных служит для хранения информации о наблюдаемой системе и объектах потоковой обработки данных. В базе метаданных хранятся сведения о пользователях, ролях, источниках данных, топиках, событиях, правилах обработки и истории статусов.

База метаданных не используется для хранения потоков сообщений Kafka, а выполняет роль хранилища данных, необходимых для работы программных средств.

Физическая модель данных (ER-модель) – это детализированная схема базы данных, которая адаптирует логическую структуру под конкретную СУБД (например, PostgreSQL, MySQL). Она определяет типы данных, размеры полей, ограничения (ключи, индексы) и физическое размещение таблиц на диске. Для разработки базы метаданных используется СУБД PostgreSQL.

Физическая модель базы метаданных отражена на рисунке 31.

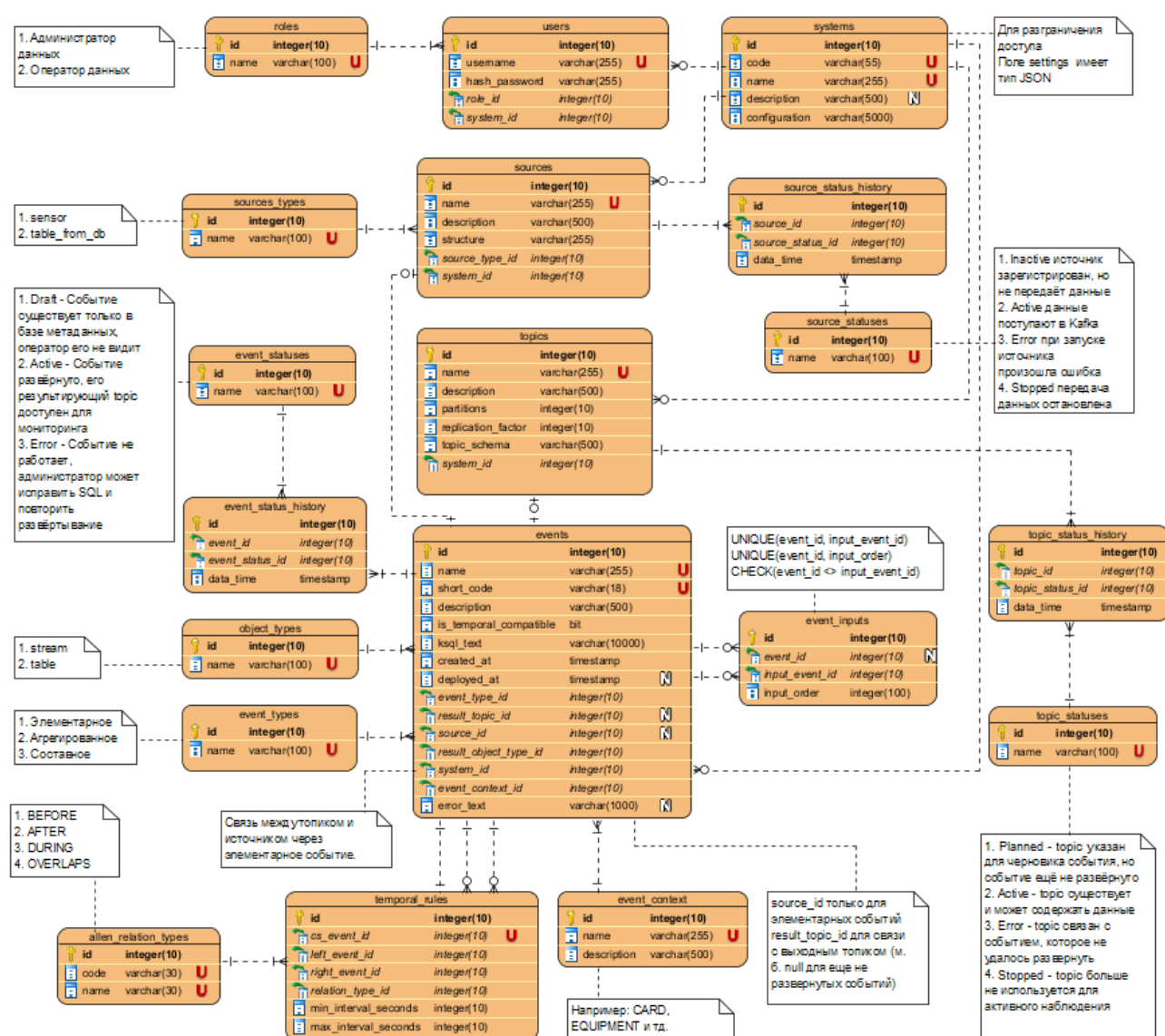


Рисунок 31 – Физическая модель базы метаданных

Спецификация таблиц базы метаданных представлена в таблице 17.

Таблица 17 – Таблицы базы метаданных EventServiceDB

Название таблицы	Назначение
roles	Справочник ролей пользователей. Определяет уровень доступа в зависимости от роли (администратор данных или оператор данных).
users	Хранит учетные записи пользователей, хеши паролей и ссылки на роль и доступную систему.
systems	Содержит сведения о наблюдаемых системах, в рамках которых настраиваются источники, события и топики. Поле configuration используется для хранения дополнительных параметров в формате JSON.
sources_types	Справочник типов источников данных, например, сенсор или таблица базы данных.
sources	Описывает источники входных данных. Содержит название, структуру, тип источника и принадлежность к системе.
source_statuses	Справочник состояний источника: «активен», «неактивен», «ошибка запуска», «остановлен».
source_status_history	Фиксирует историю изменения состояния источников с указанием времени перехода.
topics	Хранит настройки Kafka-топиков: имя, число партиций, коэффициент репликации и схему данных.
topic_statuses	Справочник состояний топика, например, «запланирован», «активен», «ошибка» или «остановлен».
topic_status_history	Содержит журнал изменения состояний Kafka-топиков.
event_types	Справочник типов событий. Используется для разделения элементарных, агрегированных и составных событий.
object_types	Справочник типов объектов, с которыми связано событие, например, поток или таблица.
event_context	Описывает предметный контекст события: устройство, карта, оборудование и другие категории объектов.
events	Центральная таблица метаданных событий. Хранит идентификатор, код, описание, KSQL-запрос, тип события, источник, результирующий топик, контекст, параметры развертывания и сведения об ошибках.
event_inputs	Задаёт входные события для составного или агрегированного события. Поле input_order определяет порядок обработки входов.
event_statuses	Справочник состояний события: «черновик», «активно», «ошибка» или «остановлено».
event_status_history	Хранит историю переходов события между состояниями с отметкой времени.
allen_relation_types	Справочник временных отношений Аллена между событиями, например, BEFORE, AFTER, DURING и OVERLAPS.
temporal_rules	Описывает временные правила между событиями. Содержит левое и правое событие, тип отношения и допустимый интервал времени.

Разработанная база метаданных удобна для сопровождения и допускает дальнейшее расширение системы. Добавление новых статусов, типов объектов или правил не требует существенной переработки существующей модели. Это упрощает развитие системы и снижает связанность её компонентов.

Для каждой таблицы была создана ORM-модель, которая позволяет управлять данными через объекты, автоматически преобразуя их в SQL-запросы и обратно, в виде C#-класса. Поля таблиц были преобразованы в свойства соответствующих типов. Доступ к моделям организован через EventServiceDbContext. В нем сущности были зарегистрированы как коллекции DbSet, после чего Entity Framework Core получил возможность формировать SQL-запросы, отслеживать изменения объектов и сохранять их в базе данных.

Ограничения, отношения «один ко многим» и составные уникальные ключи задавались в конфигурации контекста либо с помощью атрибутов моделей. Такой подход позволил сохранить структуру физической схемы и одновременно с этим работать с данными на уровне типизированных объектов.

Контроллеры вызывают соответствующие сервисы серверной части, а полученные сущности преобразовываются в DTO для передачи клиенту. При создании или изменении объекта сервис формирует экземпляр ORM-модели, добавляет его в контекст и сохраняет транзакцию через `SaveChangesAsync`.

В результате ORM-слой стал связующим звеном между реляционной базой метаданных и логикой серверного приложения, сократив объем ручного SQL-кода и упростив контроль целостности связей.

Для реализации базы метаданных был разработан SQL-скрипт [24], исходный код которого находится в открытом доступе.

3.4 Разработка серверных программных средств администрирования и обнаружения событий в системе потоковой обработки данных

В процессе проектирования было принято решение, что панели администратора и оператора не должны подключаться к Kafka напрямую. Все операции с потоковыми запросами и мониторингом в разрабатываемых программных средствах выполняются через серверную часть REST API. Это позволяет централизовать доступ к потоковой инфраструктуре, скрыть параметры подключения от клиентских приложений и обеспечить единый механизм обработки ошибок.

REST (Representational State Transfer) – это архитектурный стиль для создания веб-сервисов, который определяет правила обмена данными между клиентом и сервером при помощи стандартных HTTP-методов. Основная идея REST заключается в том, чтобы предоставлять доступ к ресурсам через определенные URL-адреса, называемые эндпоинтами (endpoints).

Эндпоинт представляет собой точное местоположение на сервере, куда могут отправляться запросы. Каждый endpoint соответствует определенному ресурсу или функции, что позволяет клиентам взаимодействовать с сервером, используя стандартные методы, такие как GET (получение данных), POST (отправка данных), PUT (замена данных) и DELETE (удаление данных). Это обеспечивает простоту использования и масштабируемость системы [25].

В рамках реализации RESTful API-сервера в качестве центрального узла разрабатываемых программных средств администрирования, обнаружения и мониторинга событий в системе потоковой обработки данных был разработан набор POST-методов, представленный в таблице 18.

Таблица 18 – POST-методы

Название	Входные данные	Описание
/api/auth/login	body: LoginRequest { username, password }	Авторизация пользователя и выдача JWT.
/api/admin/events/aggregated	body: SaveAggregatedEventRequest { shortCode, name, description, resultObjectType, resultTopicName, resultTopicDescription, resultTopicSchema, inputEventId, ksqlText, isTemporalCompatible, eventContextId }	Создать агрегированное событие.
/api/admin/events/aggregated/{eventId}/deploy	path: eventId	Развернуть агрегированное событие.
/api/admin/composite-events	body: SaveCompositeEventRequest { shortCode, name, description, resultObjectType, resultTopicName, resultTopicDescription, resultTopicSchema, inputEventIds, ksqlText, isTemporalCompatible, eventContextId }	Создать составное событие.
/api/admin/composite-events/{eventId}/deploy	path: eventId	Развернуть составное событие.
/api/admin/temporal-events/preview	body: SaveTemporalEventRequest { shortCode, name, description, resultTopicName, resultTopicDescription, leftEventId, rightEventId, relationCode, minIntervalSeconds, maxIntervalSeconds }	Предпросмотр SQL временного события.
/api/admin/temporal-events	body: SaveTemporalEventRequest	Создать временное событие.
/api/admin/temporal-events/{eventId}/deploy	path: eventId	Развернуть временное событие.
/api/admin/sources/{sourceId}/activate	path: sourceId; body: ActivateSourceRequest { elementaryEventShortCode }	Активировать источник.
/api/admin/sources/{sourceId}/stop	path: sourceId	Остановить источник.
/api/operator/analytics/counts	body: OperatorAnalyticsRequest { eventIds, fromUtc, toUtc, bucketMinutes }	Получить агрегированную аналитику по событиям.
/api/operator/monitoring/resolve	body: ResolveMonitoringRequest { eventIds }	Подготовить конфигурацию мониторинга.

В рамках реализации RESTful API-сервера был также разработан набор GET-методов, представленный в таблице 19.

Таблица 19 – GET-методы

Название	Входные данные	Описание
/api/auth/me	нет	Получить данные текущего пользователя.
/api/auth/test/administrator	нет	Проверить доступ администратора.
/api/auth/test/operator	нет	Проверить доступ оператора.
/api/health/database	нет	Проверить подключение к БД.
/api/admin/dashboard/summary	нет	Получить сводку для панели администратора.
/api/admin/dashboard/infrastructure-health	нет	Получить состояние инфраструктуры.
/api/admin/dashboard/topic-availability	нет	Получить доступность Kafka topic.
/api/admin/events	нет	Получить список событий.
/api/admin/events/{eventId}	path: eventId	Получить событие по идентификатору.
/api/admin/events/{eventId}/sql	path: eventId	Получить SQL события.
/api/admin/events/{eventId}/composition	path: eventId	Получить состав события.
/api/admin/events/available-inputs	нет	Получить события, доступные как входные.
/api/admin/events/contexts	нет	Получить контексты событий.
/api/admin/events/{eventId}/actions	path: eventId	Получить доступные действия для события.
/api/admin/events/{eventId}/editor	path: eventId	Получить модель редактирования события.
/api/admin/temporal-events/available-inputs	нет	Получить события для временного анализа.
/api/admin/temporal-events/relations	нет	Получить отношения Аллена.
/api/admin/sources	нет	Получить список источников.
/api/admin/sources/{sourceId}	path: sourceId	Получить источник по идентификатору.
/api/admin/topics	нет	Получить список topic.
/api/admin/topics/{topicId}	path: topicId	Получить topic по идентификатору.
/api/operator/events/active	нет	Получить активные события оператора.
/api/operator/events/active/{eventId}	path: eventId	Получить активное событие по идентификатору.
/api/operator/monitoring/stream	query: eventIds[]; WebSocket	Открыть realtime-поток мониторинга.

В рамках реализации RESTful API-сервера был также разработан набор PUT-методов, представленный в таблице 20.

Таблица 20 – PUT-методы

Название	Входные данные	Описание
/api/admin/events/aggregate/{eventId}	path: eventId; body: SaveAggregatedEventRequest	Обновить агрегированное событие.
/api/admin/composite-events/{eventId}	path: eventId; body: SaveCompositeEventRequest	Обновить составное событие.
/api/admin/temporal-events/{eventId}	path: eventId; body: SaveTemporalEventRequest	Обновить временное событие.
/api/admin/events/{eventId}/draft	path: eventId	Удалить черновик события.
/api/admin/events/{eventId}/deployed	path: eventId	Удалить развернутое событие.

В рамках реализации RESTful API-сервера был также разработан набор DELETE -методов, необходимых для удаления черновика или уже развернутого события. Эти методы представлены в таблице 21.

Таблица 21 – DELETE-методы

Название	Входные данные	Описание
/api/admin/events/{eventId}/draft	path: eventId	Удалить черновик события.
/api/admin/events/{eventId}/deployed	path: eventId	Удалить развернутое событие.

Данные наборы эндпоинтов позволяют пользователям взаимодействовать с потоками данных, администрировать объекты системы и выполнять оперативный мониторинг сообщений Kafka.

Код эндпоинта composite-events из контроллера CompositeEventsController, который необходим для создания нового составного события, отображен в листинге 5.

Листинг 5 – POST /api/admin/composite-events

```
[HttpPost]
public async Task<ActionResult<EventDetailsDto>> Create(
    [FromBody] SaveCompositeEventRequest request,
    CancellationToken cancellationToken)
{
    try
    {
        int eventId = await _compositeEventService.CreateAsync(
            request,
            cancellationToken);

        EventDetailsDto createdEvent =
            await _eventReadService.GetByIdAsync(
                eventId,
                cancellationToken);

        return Created(
            $"/api/admin/events/{eventId}",
            createdEvent);
    }
    catch (ValidationException exception)
    {
        return BadRequest(new { message = exception.Message });
    }
    catch (InvalidOperationException exception)
    {
        return BadRequest(new { message = exception.Message });
    }
}
```

```

        catch (DbUpdateException exception)
        {
            return Conflict(new
            {
                message = "Не удалось сохранить составное событие.",
                detail = exception.InnerException?.Message
            });
        }
    }
}

```

Метод передает параметры нового составного события в `CompositeEventService`, после сохранения получает полную информацию о созданной сущности и возвращает ответ 201 Created с адресом нового ресурса; ошибки валидации приводят к 400 Bad Request, а конфликт при сохранении в базе данных к 409 Conflict.

Фрагмент кода метода `CompositeEventService.CreateAsync`, который используется контроллером для создания нового сложного составного события, представлен ниже, в листинге 6.

Листинг 6 – Фрагмент метода `CompositeEventService.CreateAsync`

```

List<InputEventSnapshot> inputEvents =
    await GetActiveInputEventsAsync(
        request.InputEventIds,
        cancellationToken);

await ValidateEventContextAsync(
    request,
    cancellationToken);

await ValidateUniquenessAsync(
    request,
    existingEventId: null,
    existingTopicId: null,
    cancellationToken);

ValidateManualSql(request, inputEvents);

```

На этом этапе сервис проверяет, что выбранные входные события существуют и активны, контекст события допустим, названия события и `topic` не конфликтуют с уже существующими, и введенный администратором `ksqlDB`-запрос действительно использует указанные входные события.

После создания составного события сервис сохраняет связи с каждым входным событием и фиксирует их порядок, который затем используется при анализе состава события и его развертывании в `ksqlDB`. Фрагмент метода

CompositeEventService.CreateAsync для выполнения этой задачи представлен в листинге 7.

Листинг 7 – Сохранение связей с входными событиями

```
for (int index = 0; index < inputEvents.Count; index++)
{
    _dbContext.EventInputs.Add(
        new EventInput
        {
            EventId = eventItem.Id,
            InputEventId = inputEvents[index].Id,
            InputOrder = index + 1
        });
}
```

Благодаря этой части метода сервер создает записи в таблице EventInput базы метаданных, связывающие новое составное событие с выбранными входными событиями, и сохраняет их порядок через поле InputOrder. В последствии это применяется для вывода дерева состава события и позволяет наглядно показать его структуру.

В ходе реализации программных средств EventService.Api был выделен следующий набор входящих в них пакетов:

- пакет Controllers;
- пакет Application Services;
- пакет Infrastructure Services;
- пакет Data and Contracts;
- пакет Configuration;
- пакет Simulation.

Для визуализации взаимосвязей между данными пакетами, а также для представления их внутренних компонентов была построена диаграмма компонентов.

Диаграмма компонентов – это один из видов структурных диаграмм, используемых в языке моделирования UML (Unified Modeling Language) [26]. Она показывает разбиение системы на компоненты и их зависимости.

Данная диаграмма представлена на рисунке 32.

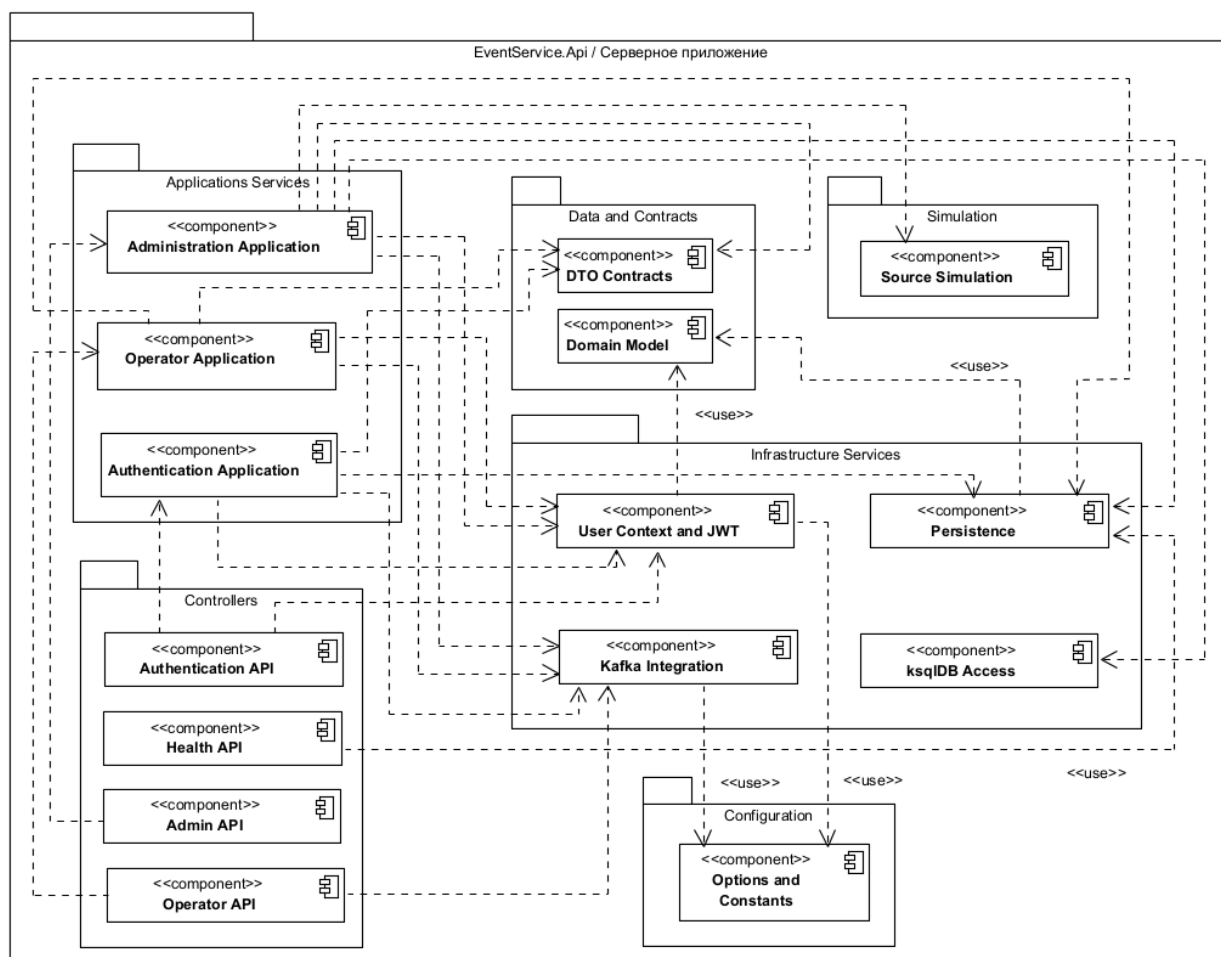


Рисунок 32 – Диаграмма компонентов программных средств

Пакет Controllers содержит компоненты, принимающие HTTP-запросы и формирующие ответы API. В пакет входит следующий набор компонентов:

- Authentication API, отвечающий за операции аутентификации и получение сведений о текущем пользователе;
- Health API, предоставляющий endpoints проверки работоспособности приложения;
- Administration API, который объединяет административные операции управления объектами системы;
- Operator API, обслуживающий запросы мониторинга, просмотра событий и аналитических данных.

Контроллеры не реализуют бизнес-правила. Их задача ограничена проверкой входных данных, вызовом нужного сервиса и подготовкой результата.

В пакете Application Services сосредоточена прикладная логика серверного приложения. В него входят следующие компоненты:

- Administration Application, выполняющий сценарии администратора;
- Operator Application, необходимый для обработки сценариев мониторинга и аналитики;
- AuthService, выполняющий проверку учетных данных и формирует результат аутентификации.

Данные сервисы связывают API с инфраструктурой и не зависят от конкретного способа представления HTTP-ответа.

Компоненты Infrastructure Services пакета обеспечивают доступ к техническим механизмам приложения. К ним относятся:

- User Context and JWT, отвечающий за чтение данных текущего пользователя (CurrentUserService) и создание JWT-токенов (JwtTokenService);
- Kafka Integration, необходимый для инкапсуляции административных операций Kafka (KafkaAdminService), чтения сообщений для аналитики (KafkaAnalyticsReader) и потокового мониторинга (KafkaMonitoringStreamService);
- ksqlDB Access, служащий компонентом доступа, выполняющий запросы к ksqlDB и преобразующий полученные данные в объекты приложения (KsqlDbClient);
- Persistence обеспечивает работу с постоянными данными через Entity Framework Core, его центральным элементом является EventServiceDbContext.

Инфраструктурные компоненты используются прикладными сервисами и отдельными контроллерами.

Пакет Data and Contracts содержит структуры, которыми обмениваются остальные части приложения. К его компонентам относятся:

- Domain Models, описывающие внутренние сущности и их состояние, которые применяются в бизнес-логике и при работе с Persistence;

– DTO Contracts, включающие объекты запросов, ответов и промежуточных представлений данных, отделяющие внешний контракт API от внутренних моделей.

Разделение моделей и DTO снижает связанность между HTTP-интерфейсом и структурой базы данных.

Пакет Configuration содержит параметры, необходимые для запуска и настройки компонентов. Компонент объединяет классы настроек и общие константы приложения. Через него передаются параметры JWT, Kafka, ksqlDB и других инфраструктурных механизмов. Этот компонент не выполняет прикладных операций, а только предоставляет настройки тем частям системы, которым они нужны. Пакет Simulation отвечает за генерацию тестовых событий внутри серверного приложения. Компонент Source Simulation создает искусственный поток данных и передает сообщения в Kafka Integration. Компонент используется для проверки мониторинга и аналитических сценариев без подключения реального источника данных. Внутри пакета находится сервис симуляции и связанные с ним структуры генерации данных.

Таким образом, серверная часть программных средств, работающая на основе RESTful API, состоит из взаимосвязанных компонентов, разделенных по логическим группам и выполняющих конкретные задачи системы. Фрагменты листинга серверной части представлены в приложении В. Полный программный код находится в открытом доступе [25].

3.5 Разработка клиентских приложений администрирования и диспетчеризации событий

3.5.1 Разработка интерфейса

Оба клиентских приложения имеют проработанные и стильные интерфейсы, реализация которых была отдельной задачей при разработке программных средств. Благодаря интерфейсу пользователь получает доступ ко всем возможностям системы, будучи полностью изолированным от сложных

конструкций реализации программных средств, в том числе и от их серверной части.

Интерфейсы десктопных приложений разрабатывались на языке C# средствами Windows Forms. Windows Forms – это технология пользовательского интерфейса для .NET, представляющая собой набор управляемых библиотек. С помощью среды разработки, такой как Visual Studio, можно она позволяет создавать интеллектуальные клиентские приложения Windows Forms, которые отображают информацию, запрашивают ввод пользователя и выполняют пользовательские функции.

Для разработки клиентских приложений использовались различные элементы управления, предоставляемые технологией Windows Forms. Каждое окно представляет из себя форму Form. Окно авторизации администратора предоставляет пользователю увидеть адрес сервера, к которому будет производиться подключение, ввести свой логин и пароль, а также использовать функции просмотра пароля и запоминания данных для повторного входа. После заполнения полей администратор может нажать кнопку «Войти в систему». При необходимости кнопкой «Заккрыть» можно выйти из панели. На окне присутствуют элементы для отображения текста label, текстовые поля ввода/вывода textbox, элементы выбора параметров checkbox и кнопки button. Внешний вид окна авторизации панели администратора представлен на рисунке 33.

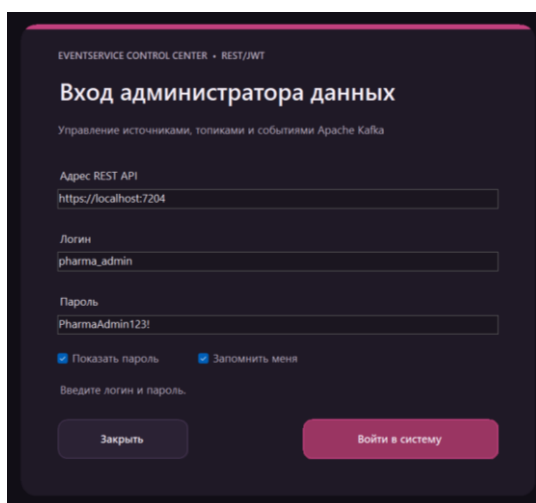


Рисунок 33 – Окно авторизации панели администратора данных

После нажатия кнопки «Войти в систему» авторизованный пользователь попадает на главное окно панели администратора. Там он может просматривать информацию по текущему состоянию системы и контролировать доступность компонентов ее инфраструктуры. В левой части окна располагается навигационное меню, позволяющее администратору перемещаться между вкладками. Интерфейс содержит в себе элементы для отображения текста label, контейнеры для группировки элементов panel, таблица dataGridView и кнопки button. Внешний вид главного окна панели администратора представлен на рисунке 34.

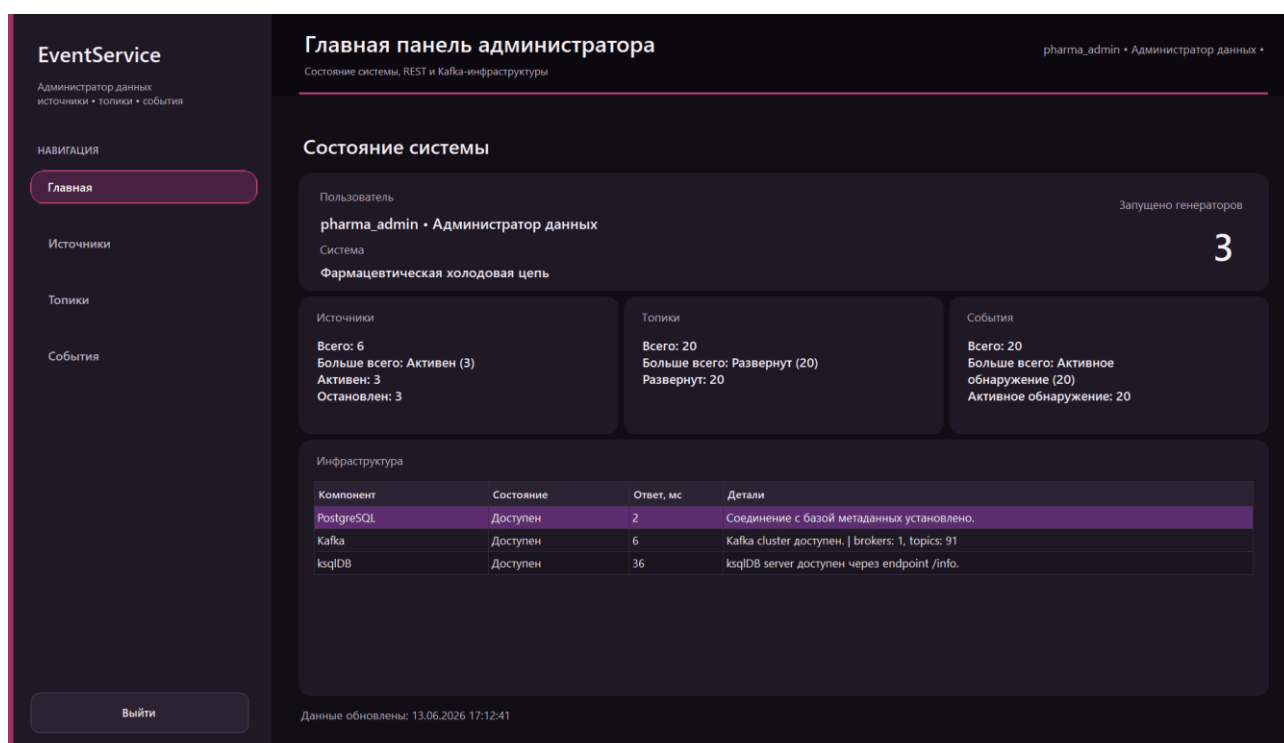


Рисунок 34 – Главная панель администратора данных

Далее пользователь может перемещаться по вкладкам навигационного меню: «Главная», «Источники», «Топики» и «События». В нижней части меню располагается кнопка «Выйти», позволяющая вернуться на окно авторизации, не прерывая текущую сессию.

Окно «Источники» позволяет администратору отслеживать текущее состояние всех зарегистрированных в системе источников данных с помощью удобного табличного отображения. Активные источники отображаются зеленым

цветом, отключенные – белым. Над таблицей присутствует поисковая строка, благодаря которой пользователь может найти в таблице нужный источник по его названию или типу. В верхней части окна находится кнопка «Обновить», которая позволяет обновить список источников, не перезагружая окно целиком. В правой части окна отображается карточка источника, в которой выводится подробная информация о выбранном в таблице источнике, включая его описание и структуру. В нижней части окна находятся поле для ввода краткого кода элементарного события, кнопки для активации отключенного источника и деактивации включенного. На окне присутствуют элементы для отображения текста label, текстовые поля ввода/вывода textbox, контейнеры для группировки элементов panel, таблица dataGridView и кнопки button. Вкладка источников отображена на рисунке 35.

EventService
Администратор данных
источники • топики • события

НАВИГАЦИЯ
Главная
Источники
Топики
События

Источники
pharma_admin • Администратор данных •
Просмотр источников, активация элементарных событий и управление генераторами

Источники данных
Просмотр источников системы, создание элементарного события, запуск и остановка генераторов

Обновить

СПИСОК ИСТОЧНИКОВ

Поиск:

ID	Источник	Тип	Статус	Генерация
8	pharma_reefer_door_sensor	DOOR_STATE	Активен	Да
7	pharma_reefer_humidity_sensor	HUMIDITY	Остановлен	Нет
11	pharma_reefer_power_sensor	POWER_STATUS	Остановлен	Нет
6	pharma_reefer_temperature_sensor	COLD_TEMPERAT...	Активен	Да
9	pharma_reefer_vibration_sensor	VIBRATION	Остановлен	Нет
10	pharma_vehicle_telemetry_sensor	VEHICLE_TELEME...	Активен	Да

Карточка источника

Название: pharma_reefer_humidity_sensor

Статус: Остановлен

Генерация: Нет

Тип имитации: HUMIDITY

Интервал: 3000 мс

Связанных событий: 1 событие

Статус события: Активное обнаружение

Kafka topic: raw_pharma_cold_chain_source_7

Описание: Датчик относительной влажности внутри фармацевтического контейнера.

Структура: sensor_id VARCHAR, container_id VARCHAR, humidity DOUBLE, event_ts BIGINT

Краткий код события: HUMIDITY

Элементарное событие уже создано. Повторная активация перезапустит генератор источника.

Источники загружены: 6. Обновлено 13.06.2026 17:13:09.

Рисунок 35 – Вкладка «Источники»

Окно «Топики» предоставляет пользователю возможность просматривать информацию о доступных в системе топиках. Отображение списка топиков происходит с помощью таблицы в левой части окна. Развернутые топики отображаются зеленым цветом, сохраненные, но не развернутые на Kafka –

белым. Над таблицей присутствует поисковая строка, благодаря которой пользователь может найти в таблице нужный топик по его названию. В верхней части окна находится кнопка «Обновить», которая позволяет обновить список топиков, не перезагружая окно целиком. В правой части окна отображается карточка топика, в которой выводится подробная информация о выбранном в таблице топике, включая его описание и список событий, которые используют этот топик в качестве источника входных данных. Интерфейс состоит из элементов для отображения текста label, текстовых полей ввода/вывода textbox, контейнеров для группировки элементов panel, таблиц dataGridVeiw и кнопок button. Вкладка топиков представлена на рисунке 36.

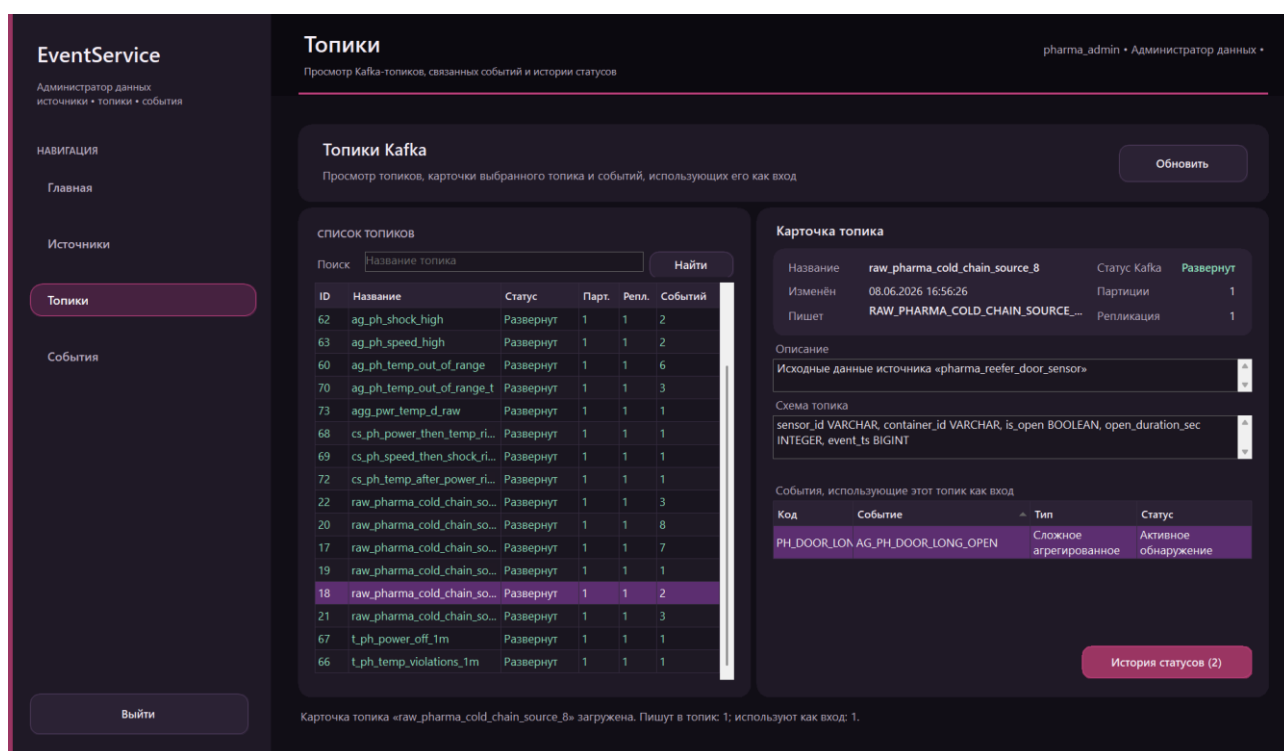


Рисунок 36 – Вкладка «Топики»

В нижней части окна топиков находится кнопка «История статусов», позволяющая просмотреть список состояний выбранного топика с точными датой и временем, когда был приобретён тот или иной статус. Интерфейс окна состоит из элементов для отображения текста label, контейнеров для группировки элементов panel, таблицы dataGridVeiw и кнопок button. Всплывающее окно «История статусов топика» показано на рисунке 37.

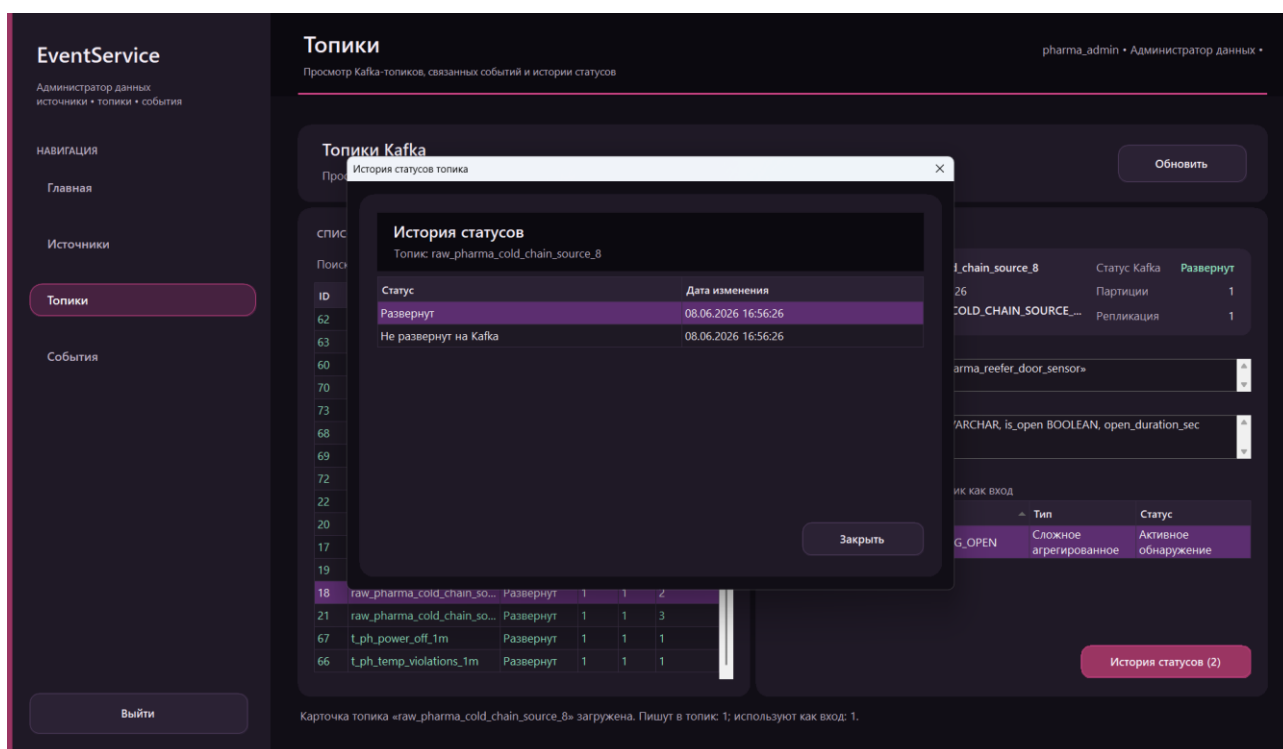


Рисунок 37 – Окно истории статусов топика

Окно «События» позволяет администратору просматривать информацию о имеющихся в наблюдаемой системе событиях. Список событий представляет из себя таблицу, где события, находящиеся в статусе активного обнаружения, отображаются зеленым цветом, события, сконструированные, но еще не развернутые на Kafka, – белым, а те, в которых возникла ошибка, – красным. Над таблицей присутствует поисковая строка, благодаря которой пользователь может найти в таблице нужное событие по его названию или краткому коду. В верхней части окна находятся кнопка «Обновить», которая позволяет обновить список источников, не перезагружая окно целиком, и кнопка «Создать», необходимая для перехода на окно создания нового события. В правой части окна отображается карточка события, в которой выводится подробная информация о выбранном в таблице событии, включая его описание. Под карточкой товара находятся четыре кнопки: «Редактировать» – кнопка, доступная только для неразвернутых событий и позволяющая изменить параметры или код запроса события, «Удалить активное» – кнопка удаления активного события, «Удалить черновик» – кнопка удаления неразвернутого

события, «История» – кнопка для перехода на окно с историей состояний конкретного события. В нижней части окна находится таблица для отображения дерева состава события, где администратор может просмотреть полный состав события и выявить место ошибки, при ее возникновении. На окне присутствуют элементы для отображения текста label, текстовые поля ввода/вывода textbox, контейнеры для группировки элементов panel, таблица dataGridView и кнопки button. Вкладка событий представлена на рисунке 38.

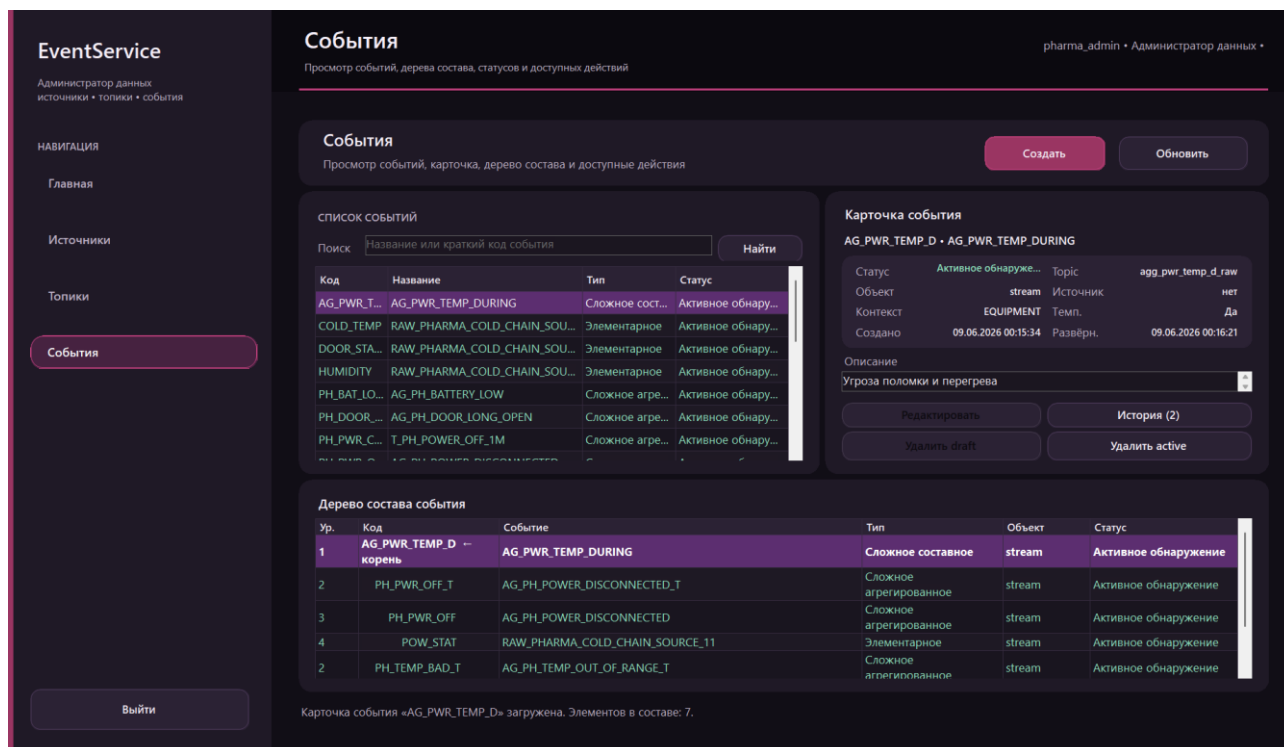


Рисунок 38 – Окно событий

При нажатии на кнопку «Создать» на окне событий, открывается всплывающее окно «Создание события».

В левой части этого окна администратор может ввести техническое название нового события, его краткий код, выбрать в раскрывающемся списке объект результата (topic или stream), ввести текстовое описание события, отметить возможность его использования в темпоральном конструкторе, выбрать контекст события из предложенных или ввести новый. В правой части окна пользователь с помощью раскрывающегося списка выбирает тип события: агрегированное или составное.

При выборе агрегированного события ему становится доступна таблица с возможностью поиска по названию или коду. События, выводящиеся в таблице, соответствуют выбранному ранее контексту. Благодаря полю выбора в первом столбце таблицы администратор может выбрать нужное событие в качестве входа. Предусмотрена возможность просмотра кода запроса входного события по кнопке «SQL-код», которая находится справа от строки поиска. Все вводимые параметры автоматически переносятся в шаблон в нижней части окна создания события, что сокращает время администратора на написание кода и предотвращает синтаксические ошибки. Интерфейс окна состоит из элементов для отображения текста label, раскрывающихся списков comboBox, контейнеров для группировки элементов panel, элементов выбора параметров checkbox, таблицы dataGridView и кнопок button. На рисунке 39 отображен внешний вид интерфейса окна создания нового агрегированного события.

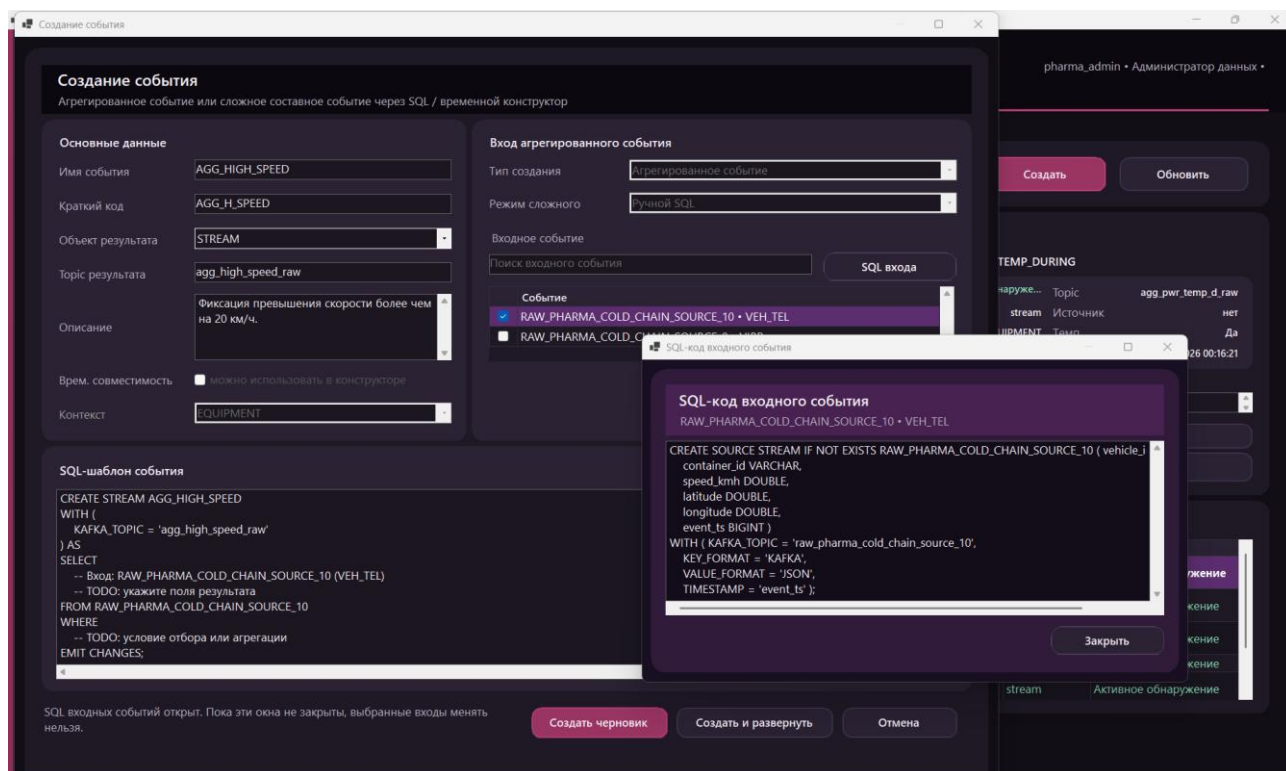


Рисунок 39 – Окно создания нового агрегированного события

При выборе составного события администратор так же сможет выбрать один из двух режимов конструирования: «Ручной SQL» или «Темпоральный конструктор Аллена». В ручном режиме пользователь получит возможность

выбрать два события в качестве входа и просмотреть код запросов для обоих событий. В режиме темпорального конструктора помимо выбора двух входных событий администратору нужно будет отметить минимальное и максимальное значения интервала и выбрать в раскрывающемся списке один из четырех предоставленных видов отношений. Ниже, в поле «Выражение» автоматически конструируется логическое выражение в синтаксисе алгебры Аллена, а шаблон SQL-запроса для создания нового составного события сгенерируется автоматически в нижнем поле. Окно состоит из элементов для отображения текста label, раскрывающихся списков comboBox, контейнеров для группировки элементов panel, элементов для выбора чисел из диапазона с фиксированным шагом изменения numericUpDown, элементов выбора параметров checkbox, таблицы dataGridView и кнопок button. Интерфейс окна создания нового составного события показан на рисунке 40.

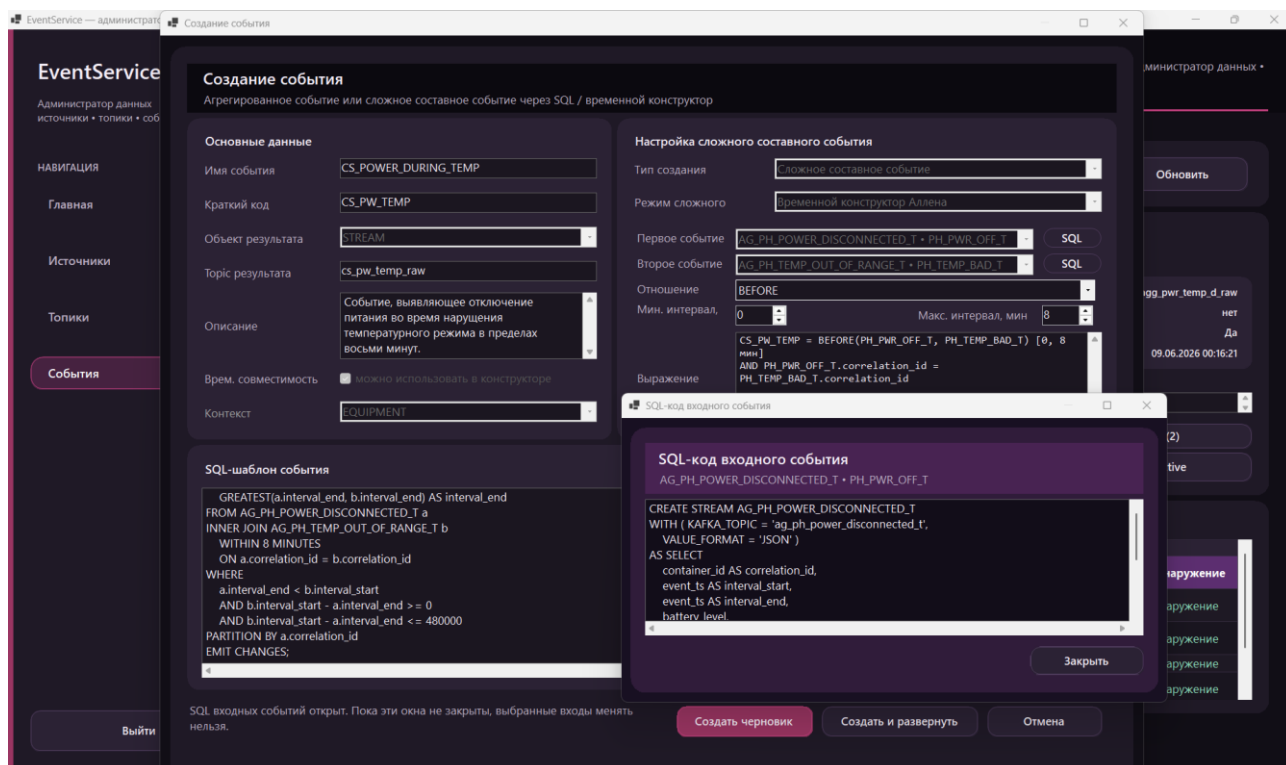


Рисунок 40 – Окно создания нового составного события в режиме темпорального конструктора

В нижней части окна создания события находятся поле для логов и кнопки «Сохранить черновик», «Сохранить и развернуть» и «Отмена». Первая позволяет

сохранить событие неразвернутым с последующей возможностью редактирования и удаления, вторая нужна для немедленного запуска обнаружения нового события, а третья позволяет выйти из окна без сохранения.

После нажатия одной из этих кнопок при отсутствии ошибок пользователь попадет обратно на окно событий и сможет продолжить работу. При возникновении ошибок на любом из этапов его работы, система выведет текст ошибки или не даст завершить некорректное действие. Благодаря этому панель администратора становится удобным и продуманным инструментом, позволяющим комфортно работать как новому пользователю, так и опытному специалисту.

Окно авторизации оператора по функционалу не отличается от окна авторизации администратора. При нажатии на кнопку «Войти в систему» авторизованной пользователь попадет на главное окно панели оператора.

На окне присутствуют элементы для отображения текста label, текстовые поля ввода/вывода textbox, элементы выбора параметров checkbox и кнопки button. Интерфейс окна авторизации оператора представлен на рисунке 41.

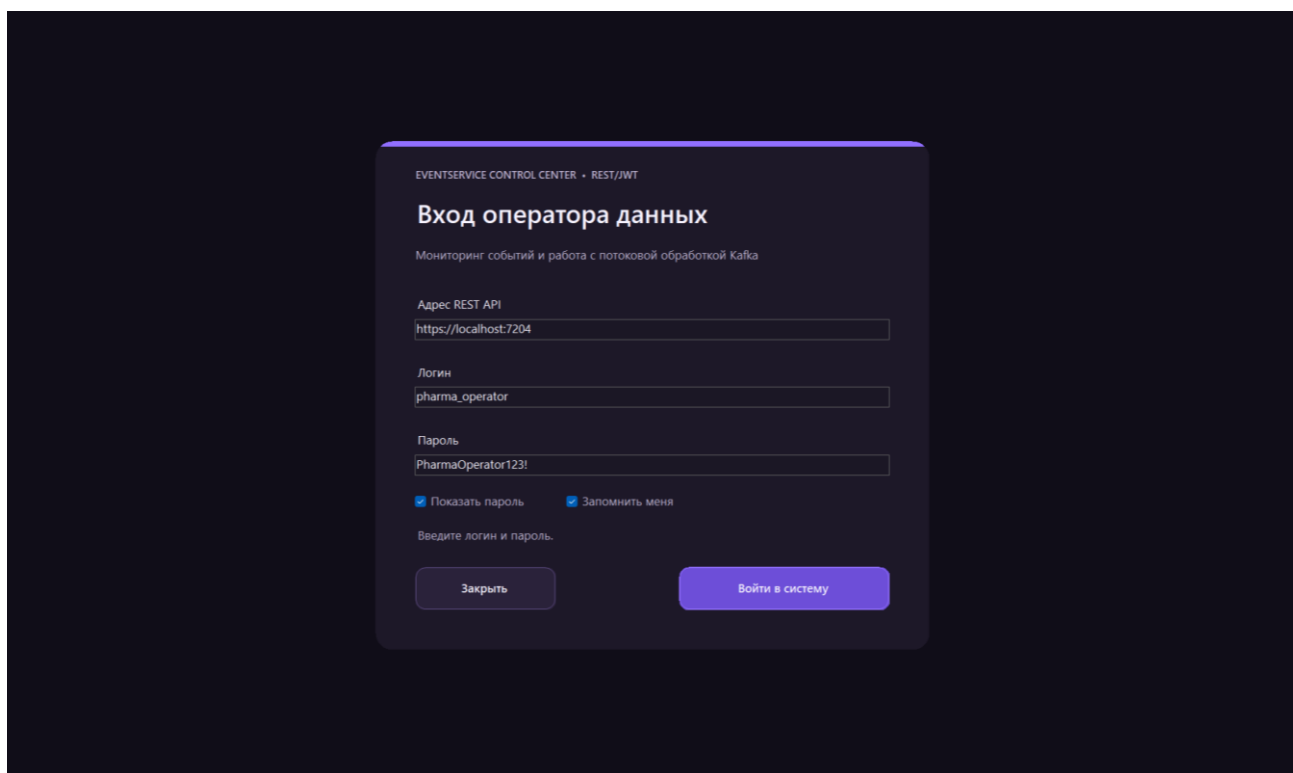


Рисунок 41 – Окно авторизации панели оператора

На главном окне панели оператора пользователь получает доступ к информации о системе и списку событий, находящихся в статусе активного обнаружения. Таблица в нижней панели позволяет выбирать события для дальнейшего мониторинга с помощью поля выбора в первом столбце. Над таблицей находится строка поиска, благодаря которой оператор может найти событие по коду, названию или имени результирующего топика.

Так же оператор может воспользоваться дополнительной фильтрацией событий, выбрав временной отрезок, в котором было развернуто событие, или тип события. Кнопки справа позволяют применить или отменить фильтрацию к списку событий. Кнопка «Перейти к мониторингу» в нижней части окна позволит сразу начать отслеживание выбранных в таблице событий.

Интерфейс окна состоит из элементов для отображения текста label, раскрывающихся списков comboBox, элементов для выбора даты и времени datePicker, текстовых полей ввода/вывода textbox, контейнеров для группировки элементов panel, элементов выбора параметров checkbox, таблицы dataGridView и кнопок button. На рисунке 42 показан внешний вид главного окна панели оператора.

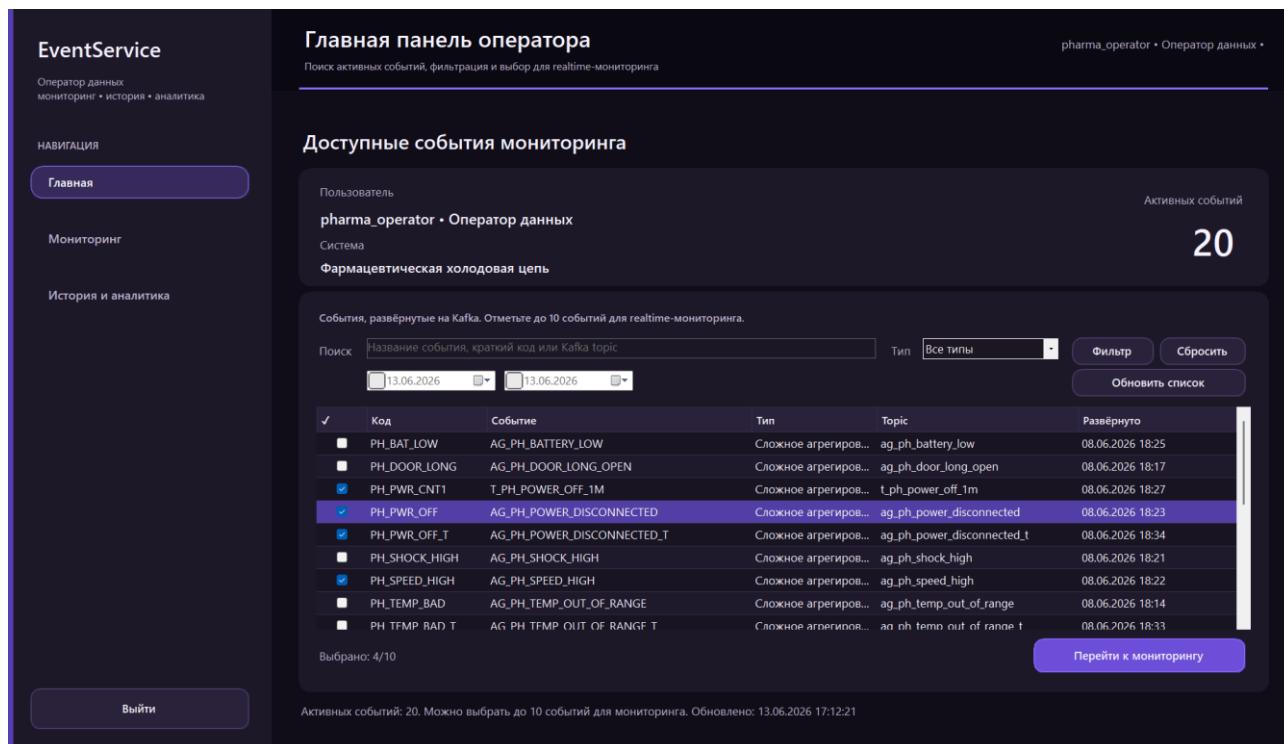


Рисунок 42 – Главное окно панели оператора

На окне мониторинга оператор может в реальном времени просматривать поступающие сообщения об обнаруженных событиях. В верхней панели выводятся названия выбранных активных событий. В центре окна расположена динамическая таблица, куда в реальном времени попадают структурированные данные из результирующих топиков, выбранных на главном окне событий. Ниже отображается отформатированный текст сообщения с выделением полей верхнего уровня. В нижней части окна расположена панель с динамически обновляемыми полями. Там отображается число сообщений, пришедших за время мониторинга, число ошибок, а также время и топик последнего на данный момент обнаруженного события. В верхней части окна находятся кнопки запуска, остановки мониторинга и кнопка очистки потока, которая очищает таблицу, но не останавливает мониторинг. При нажатии на кнопку «Стоп» в локальном хранилище автоматически сохраняется отчет по проведенному мониторингу. Окно включает в себя элементы для отображения текста `label`, текстовые поля ввода/вывода `textbox`, контейнеры для группировки элементов `panel`, таблицу `dataGridView` и кнопки `button`. На рисунке 43 отображен внешний вид интерфейса вкладки мониторинга событий.

EventService
Оператор данных
мониторинг • история • аналитика

НАВИГАЦИЯ
Главная
Мониторинг
История и аналитика

Мониторинг событий
pharma_operator • Оператор данных •
WebSocket-подписка на результирующие Kafka topics выбранных событий

Realtime-мониторинг Kafka
События: PH_PWR_OFF • AG_PH_POWER_DISCONNECTED, PH_SPEED_HIGH • AG_PH_SPEED_HIGH, PH_SPEED_SHOCK •
Topics: ag_ph_power_disconnected, ag_ph_speed_high, cs_ph_speed_then_shock_risk, raw_pharma_cold_chain_source_7

Старт Стоп
Очистить поток

Входящие сообщения

Получено	Событие	Topic	Данные	Offset	Ошибка
17:23:12	VIBR • RAW_PHARMA_COLD_C...	raw_pharma_cold_chain...	sensor_id: VIB_SENSOR_02 • container_id: PHARMA_BOX_02 • ...	2650	нет
17:23:12	VEH_TEL • RAW_PHARMA_COL...	raw_pharma_cold_chain...	vehicle_id: PHARMA_TRUCK_02 • container_id: PHARMA_BOX...	2806	нет
17:23:12	HUMIDITY • RAW_PHARMA_C...	raw_pharma_cold_chain...	sensor_id: HUM_SENSOR_01 • container_id: PHARMA_BOX_0...	3107	нет
17:23:09	VEH_TEL • RAW_PHARMA_COL...	raw_pharma_cold_chain...	vehicle_id: PHARMA_TRUCK_01 • container_id: PHARMA_BOX...	2805	нет
17:23:09	HUMIDITY • RAW_PHARMA_C...	raw_pharma_cold_chain...	sensor_id: HUM_SENSOR_03 • container_id: PHARMA_BOX_0...	3106	нет

Форматированное сообщение

```

Получено: 13.06.2026 17:23:12
Kafka timestamp: 13.06.2026 17:23:12
Событие: VIBR • RAW_PHARMA_COLD_CHAIN_SOURCE_9
Тип события: элементарное
Topic: raw_pharma_cold_chain_source_9
Partition: 0
Offset: 2650
Key: PHARMA_BOX_02
JSON: да

```

Всего сообщений: **25** Ошибок: **0** Последнее сообщение: 17:23:12 • raw_pharma_cold_chain_source_9

Получено сообщение из raw_pharma_cold_chain_source_9. Видимых строк: 25.

Выйти

Рисунок 43 – Вкладка «Мониторинг событий»

Окно истории и аналитики предоставляет оператору возможность получения аналитических данных о обнаруженных в системе событиях за выбранный промежуток времени. В верхней панели окна расположены поля выбора даты и времени для начала и окончания периода, за который нужно просчитать аналитику. Слева в поле «Интервал» оператор может ввести количество минут, которому будет равен интервал для вывода аналитики. В этой же панели слева находятся две кнопки. Кнопка «Обновить события» позволяет обновить список доступных активных событий без перезапуска всего окна. Кнопка «Построить аналитику» позволяет пользователю получить результат аналитики по выбранным параметрам.

В таблице в левой части окна оператор должен предварительно выбрать те события, аналитику по которым он хочет получить. Это возможно благодаря специальному полю выбора в первом столбце таблицы. После нажатия кнопки «Построить аналитику» оператор в правой части окна получает графическое представление в виде столбчатых диаграмм и графику общего потока по времени.

В нижней части окна расположена панель, где отображается число обнаруженных за выбранный промежуток времени с указанным интервалом событий. Так же там отображается информация о количестве выбранных для аналитики событий и пиковое значение за интервал.

Переключаясь между окнами своей панели, оператор не теряет текущие данные, так как состояние окон сохраняется до непосредственного обновления параметров самим пользователем.

Интерфейс вкладки «История и аналитика» состоит из элементов для отображения текста `label`, раскрывающихся списков `comboBox`, элементов для выбора даты и времени `dateTimePicker`, элементов для выбора чисел из диапазона с фиксированным шагом изменения `numericUpDown`, текстовых полей ввода/вывода `textBox`, контейнеров для группировки элементов `panel`, элементов выбора параметров `checkbox`, таблицы `dataGridVeiw` и кнопок `button`. Вкладка истории и аналитики представлена на рисунке 44.

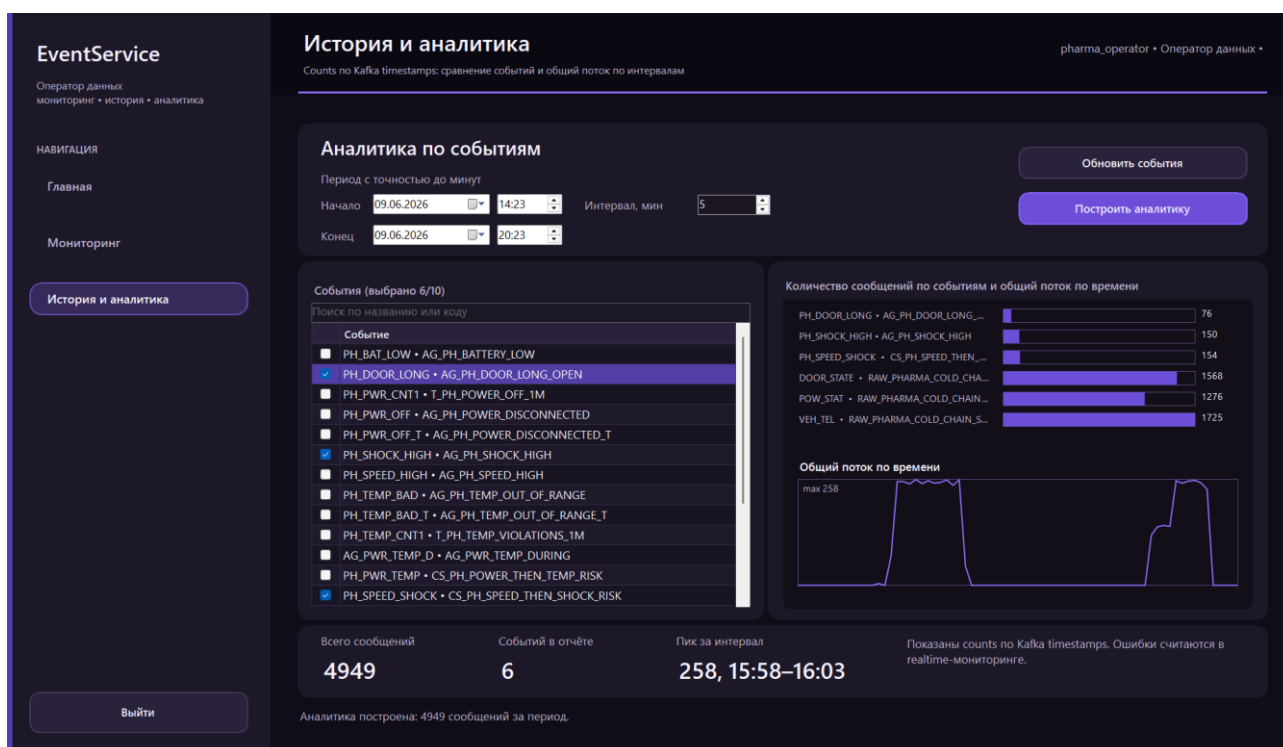


Рисунок 44 – Вкладка «История и аналитика»

Интерфейсы обоих клиентских приложений обладают единым стилем на основе темной темы с цветовыми акцентами. Использование общей дизайн-схемы гарантирует узнаваемое и удобное отображение данных.

Таким образом интерфейсы клиентских панелей администрирования и диспетчеризации являются структурированными, легкими для восприятия, простыми для использования как для новых пользователей, так и для опытных специалистов.

3.5.2 Разработка программных средств клиентских приложений

В составе разрабатываемых программных средств администрирования и мониторинга событий было разработано два отдельных клиентских приложения для двух разных классов пользователей. Оба этих приложения взаимодействуют с централизованной клиентской библиотекой.

Для клиентских приложений была разработана диаграмма компонентов, которая демонстрирует структуру системы и взаимодействие ключевых компонентов. Диаграмма компонентов представлена на рисунке 45.

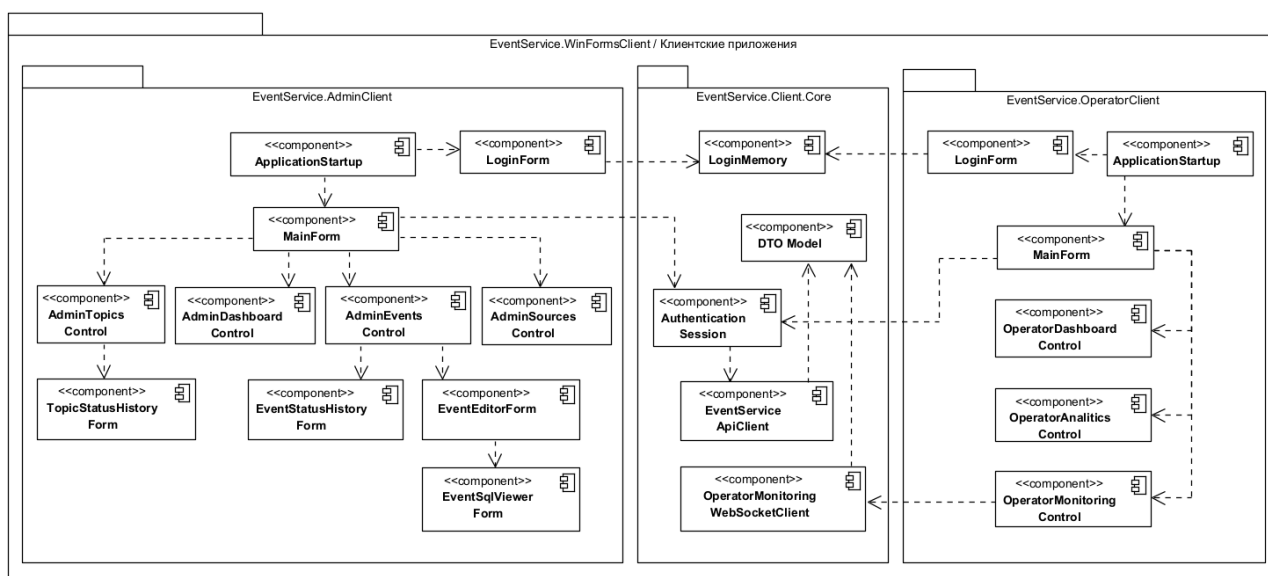


Рисунок 45 – Диаграмма компонентов для клиентских приложений

На диаграмме изображены основные модули в виде пакетов. В ходе разработки было выполнено разделение решения на два независимых исполняемых WinForms-приложения и общую библиотеку EventService.Client.Core. Клиентские панели администратора и оператора имеют разные наборы окон, но общие для них модели данных и состояние авторизации не дублируются. Такое распределение позволило отделить сценарии пользовательского интерфейса от общей логики взаимодействия с серверной частью.

Подробное описание компонентов представлено в таблицах 22 – 24.

Таблица 22 – Спецификации диаграммы компонентов пакета «AdminClient»

Компонент	Описание
ApplicationStartup	Точка входа для панели администратора. Создает сессию, последовательно запускает форму авторизации и главное окно, а также обрабатывает повторный вход после выхода из системы.
LoginForm	Форма авторизации администратора. Поддерживает временное запоминание параметров входа.
MainForm	Главное окно администратора. Содержит навигацию и размещает в рабочей области элементы управления для главной панели, источников, топиков и событий.
AdminDashboardControl	Форма главного окна панели администратора, отображает информацию о системе и ее состоянии.
AdminSourcesControl	Форма окна источников данных. Обеспечивает просмотр и поиск источников, отображение их параметров, активацию элементарных событий и остановку генерации.
AdminTopicsControl	Форма, отображающая список Kafka-топиков, их конфигурацию, схему, текущее состояние и связанные события.

Продолжение таблицы 22

Компонент	Описание
AdminEventsControl	Форма окна событий. Реализует просмотр, поиск и выбор событий, загрузку состава события и доступных операций. Запускает создание, редактирование, развёртывание и удаление событий.
TopicStatusHistoryForm	Форма окна истории статусов топика.
EventStatusHistoryForm	Форма окна истории статусов событий.
EventEditorForm	Форма окна создания и редактирования редактирования и событий. Выполняет валидацию параметров, формирует SQL-шаблоны и позволяет сохранить либо сразу развернуть событие.
EventSqlViewerForm	Форма окна для отображения SQL-кода входного события.

Таблица 23 – Спецификации диаграммы компонентов пакета «Client.Core»

Компонент	Описание
LoginMemory	Объект, хранящий адрес API, логин и пароль отдельно для каждой роли в памяти текущего процесса.
DTO Model	Содержит модели запросов и ответов для авторизации, административных операций, мониторинга и аналитики. Эти модели используются как единый контракт между интерфейсом и REST API.
Authentication Session	Компонент, хранящий требуемую роль, данные авторизованного пользователя и настроенный API-клиент. Управляет завершением входа, проверкой состояния авторизации и освобождением ресурсов при выходе.
EventServiceApiClient	Единый центр выполнения всех HTTP-запросов GET, POST, PUT и DELETE, сериализации JSON и передачи JWT-токена. Централизованно обрабатывает ошибки API, перенаправления и пустые ответы.
OperatorMonitoringWebSocketClient	Объект, устанавливающий WebSocket-соединение с потоком мониторинга.

Таблица 24 – Спецификации диаграммы компонентов пакета «OperatorClient»

Компонент	Описание
ApplicationStartup	Точка входа для панели оператора. Создает сессию, запускает авторизацию и главное окно, после чего обрабатывает завершение работы или повторный вход.
LoginForm	Форма авторизации оператора. По структуре использует тот же сценарий входа, что и административный клиент.
MainForm	Главное окно оператора. Управляет переходами между панелью событий, мониторингом и аналитикой, сохраняет выбранные для наблюдения события.
OperatorDashboardControl	Форма главного окна оператора. Загружает активные события, позволяет выбрать события для последующего мониторинга.
OperatorMonitoringControl	Форма мониторинга, которая управляет запуском и остановкой мониторинга и выводит поток сообщений.
OperatorAnalyticsControl	Форма получения истории и аналитики, отображающая результаты аналитики и ее графическое представление.

При разработке было сохранено стандартное для WinForms разделение классов. Разметка элементов и параметры, формируемые дизайнером Visual Studio, находятся в файлах .Designer.cs. Обработчики, запросы и прикладная логика размещены в основных файлах классов. Поскольку WinForms не использует таблицы стилей, визуальное оформление было вынесено в отдельные файлы UiTheme.cs. В них определены стили, цвета, шрифты, оформление кнопок, таблиц, панелей и анимация появления форм.

Сетевое взаимодействие было организовано асинхронно. REST-запросы проходят через единый EventServiceApiClient, а передаваемые данные описаны DTO-моделями. Поточковый мониторинг был выделен в WebSocket-клиент. В интерфейсе предусмотрены отмена операций, безопасное закрытие соединения и перенос полученных данных в UI-поток.

Благодаря такой структуре компонентов и их реализации, клиентские приложения используют всю предоставляемую серверной частью бизнес-логику и обеспечивают удобство пользователя. Фрагменты листинга клиентских приложений представлены в приложении Г. Полный код находится в открытом доступе [27].

4 Контроль качества программных средств

4.1 Планирование процесса контроля качества программных средств

4.1.1 Обзор мероприятий по контролю качества программных средств

Контроль качества программных средств представляет собой систематическую деятельность, направленную на обеспечение соответствия разрабатываемого продукта установленным требованиям и стандартам. В рамках этой деятельности выполняется планирование проверок, непосредственное измерение характеристик программного обеспечения, а также анализ полученных результатов с целью выявления и устранения дефектов на всех этапах жизненного цикла. Основная задача контроля качества – убедиться, что программа корректно функционирует и удовлетворяет всем поставленным требованиям.

Показатели качества программных средств подразделяются на единичные, то есть относящиеся к одному свойству, и комплексные, охватывающие несколько характеристик.

К числу ключевых показателей относят функциональность, надежность, удобство применения, эффективность, сопровождаемость и мобильность. Каждый из этих показателей детализируется через набор атрибутов – например, надежность включает в себя устойчивость к дефектам, восстанавливаемость и завершенность [28].

Методы контроля качества можно классифицировать по нескольким направлениям. К числу методов контроля качества относят:

- методы, основанные на выявлении фактического поведения программы, то есть тестирование, профилирование, измерение количественных метрик;
- методы, нацеленные на обнаружение нарушений стандартов кодирования, проектных моделей и документации;

- методы формализованного анализа исходного кода и архитектуры, включая статический анализ и формальные верификации;
- методы оценки качества с использованием симуляции работы программного обеспечения, что особенно полезно для сложных распределённых систем [29].

В рамках данной выпускной квалификационной работы в качестве основных мероприятий по контролю качества выбраны функциональное тестирование методом черного ящика и оценка сложности программного кода с применением метрики Холстеда. Такой набор позволяет охватить как внешнее поведение системы, так и внутренние структурные характеристики.

4.1.2 Обоснование выбора модели тестирования

Тестирование программного обеспечения – это процесс, в ходе которого выполняется проверка соответствия реального поведения программы ожидаемым результатам [30]. Тестирование включает в себя планирование, подготовку тестовых данных, разработку и выполнение тестовых сценариев, анализ результатов и формирование отчетности. Процесс тестирования необходим для обнаружения дефектов, повышения уверенности в качестве продукта и минимизации рисков при его эксплуатации [31]. Виды тестирования описаны в таблице 25.

Таблица 25 – Виды тестирования

Название	Описание	Пример
Модульные тесты	Метод тестирования программного обеспечения, при котором проверяются отдельные модули или компоненты программы (функции, методы, классы).	Тестирование функции, которая вычисляет сумму двух чисел. Проверяется, что функция возвращает правильный результат для различных входных данных.
Интеграционные тесты	Уровень тестирования, который направлен на проверку взаимодействия нескольких модулей или компонентов системы	Тестирование взаимодействия между двумя модулями, например, проверка того, что модуль базы данных корректно взаимодействует с модулем пользовательского интерфейса.
Функциональные тесты	Тип проверки программного обеспечения, который проверяет, соответствует ли функциональность приложения его требованиям и спецификациям.	Тестирование функции регистрации пользователя. Проверяется, что пользователь может успешно зарегистрироваться с корректными данными и получить ошибку при вводе некорректных данных.

Продолжение таблицы 25

Название	Описание	Пример
Нагрузочные тесты	Процесс проверки производительности программного обеспечения или системы при увеличивающейся нагрузке	Тестирование веб-приложения под нагрузкой в 1000 одновременных пользователей. Проверяется, как система справляется с таким количеством запросов и не падает ли она.
Приемочное тестирование	Этап в процессе разработки программного обеспечения, на котором проверяется, соответствует ли продукт заранее определенным требованиям и спецификациям.	Тестирование приложения на соответствие требованиям заказчика. Проверяется, что все функции работают так, как было оговорено в техническом задании.
Тестирование производительности	Процесс оценки скорости, устойчивости и масштабируемости системы под различными нагрузками.	Измерение времени отклика системы при выполнении различных операций, таких как поиск по базе данных или загрузка страницы.
Регрессионные тесты	Проверка работоспособности приложения после внесения модификаций и доработок.	Повторное тестирование функций приложения после внесения изменений в код. Проверяется, что новые изменения не сломали ранее работавшие функции.

Выбор используемого вида тестирования зависит от специфики проекта, доступных ресурсов и стадии жизненного цикла. Для разрабатываемых программных средств администрирования и диспетчеризаций событий критически важны корректная обработка входных данных, соблюдение бизнес-логики и отказоустойчивость.

В таких условиях целесообразно применять функциональное тестирование, которое проверяет систему с точки зрения внешних требований, не углубляясь во внутреннюю реализацию. Исходя из этого был выбран подход «чёрного ящика». Данный подход не требует знания внутренней структуры кода и позволяет проверить, корректно ли система реагирует на различные комбинации входных данных и выдает ожидаемые выходные результаты [31]. Это в полной мере соответствует задачам проверки бизнес-логики и пользовательских сценариев.

4.1.3 Обоснование выбора метрик программного кода

Метрики программного кода представляют собой количественные показатели, позволяющие оценить сложность, надежность и сопровождаемость программного продукта [28]. Они могут применяться как ко всему коду, так и к отдельным модулям, классам или методам. Использование метрик дает возможность прогнозировать объемы и трудоемкость тестирования, выявлять

потенциально опасные участки кода и контролировать качество на ранних стадиях разработки.

В настоящее время существует множество различных метрик для оценки программ. Их можно разделить на несколько категорий:

- измерение сложности программ: оценка сложности кода, которая может включать в себя количество строк, циклов и других структурных элементов;
- оценка функциональной надежности: проверка того, насколько надежно работает программное обеспечение, например, как часто возникают ошибки или сбои;
- измерение производительности ПО: оценка того, насколько быстро и эффективно работает программа, например, время выполнения операций или использование ресурсов;
- измерение уровня языковых средств: оценка использования различных языковых конструкций и их влияния на качество кода;
- измерение восприятия и понимания программных текстов: оценка того, насколько легко программисты могут читать и понимать код;
- измерение производительности труда программистов: оценка эффективности работы программистов, например, количество кода, написанного за определенное время.

Для оценки качества программного кода были выбраны метрики сложности программ, которые представлены в таблице 26.

Таблица 26 – Метрики сложности программ

Название	Описание
Метрики Холстеда	Эта метрика помогает оценить сложность программы, анализируя количество уникальных слов и общее количество слов в коде. Она также использует словарь программы, чтобы определить, сколько слов нужно для написания программы.
Метрики МакКейба	Эта метрика оценивает сложность программы, подсчитывая количество базовых путей в управляющем графе. Базисные пути – это пути, которые можно пройти через программу, и маккейб предлагает использовать цикломатическую сложность как основную метрику, которая показывает, насколько сложно тестировать программу.
Метрики Спена	Эта метрика основана на анализе местоположения ссылок на данные в каждом разделе программы. Спен показывает, как часто в коде упоминается конкретный идентификатор, и если он упоминается много раз, это может затруднить процесс отладки и тестирования.
Метрики Чепина	Эта метрика направлена на оценку информационной прозрачности конкретного программного модуля. Она анализирует, как часто используются переменные из списка ввода–вывода, чтобы понять, насколько легко понять и тестировать этот модуль.

Из приведенного перечня метрик для оценки качества программного кода разрабатываемых программных средств была выбрана метрика Холстеда.

4.2 Функциональное тестирование программных средств методом черного ящика

Функциональное тестирование методом черного ящика – это техника проверки программного обеспечения, при которой тестирущик не имеет доступа к внутренней структуре кода и ориентируется исключительно на спецификации требований и входные-выходные данные. Такой подход позволяет смоделировать поведение реального пользователя и убедиться, что система обрабатывает запросы в соответствии с ожиданиями [32].

Каждый тест-кейс включает в себя конкретные действия, входные данные и ожидаемый результат. Результаты тестирования фиксируются в виде протоколов.

По окончании выполнения тестов анализируются расхождения между ожидаемыми и фактическими результатами, оформляются отчеты о дефектах. Успешным считается тест, в котором фактическое поведение полностью совпадает с ожидаемым. В таблице 27 приведены тестовые сценарии, разработанные для проверки логики работы приложений.

Таблица 27 – Функциональное тестирование программных средств методом черного ящика

№	Название теста	Ожидаемый результат	Фактические результат
1	2	3	4
1	Авторизация администратора с корректными данными	После ввода корректного логина и пароля открывается главное окно административного клиента.	Авторизация выполнена. Главное окно административного клиента открыто. Тест пройден. Рисунок Д.1.
2	Авторизация администратора с некорректными данными	При вводе неверного логина или пароля отображается сообщение «Неверный логин или пароль». Главное окно приложения не открывается.	Отображено сообщение «Неверный логин или пароль». Доступ к приложению не предоставлен. Тест пройден. Рисунок Д.2.
3	Поиск источника по названию	В списке выделяется источник, соответствующий поисковому запросу.	В списке был выделен источник, соответствующий введенному названию. Тест пройден. Рисунок Д.3.
4	Активация неактивного источника	Генерация данных запускается, источник получает статус «Активен».	Источник активирован. Элементарное событие и Kafka topic созданы, генерация данных запущена. Тест пройден. Рисунок Д.4.

Продолжение таблицы 27

1	2	3	4
5	Остановка активного источника	Генерация данных источника прекращается, а источник получает статус «Остановлен».	Генерация данных прекращена. Источнику присвоен статус «Остановлен». Тест пройден. Рисунки Д.5 и Д.6.
6	Просмотр истории состояний Kafka topic	Для выбранного Kafka topic открывается окно, содержащее зарегистрированные изменения его статуса.	Окно истории открыто. В таблице отображены изменения статуса выбранного Kafka topic. Тест пройден. Рисунок Д.7.
7	Поиск события по краткому коду	Отображается событие, которому принадлежит указанный код.	Событие с указанным кратким кодом найдено и отображено в списке. Тест пройден. Рисунок Д.8.
8	Создание агрегированного события как черновика	Создается новое агрегированное событие со статусом «Черновик».	Агрегированное событие создано и отображено в списке со статусом «Черновик». Тест пройден. Рисунки Д.9 и Д.10.
9	Создание события с уже существующим кратким кодом	Отображается сообщение о невозможности создания события. Новое событие не сохраняется.	Отображено сообщение о том, что указанный код уже используется. Новое событие не создано. Тест пройден. Рисунок Д.11.
10	Создание события с ошибкой в коде SQL-запроса	Отображается сообщение о невозможности создания события. Новое событие не сохраняется.	Отображено сообщение о ошибке в коде запроса. Новое событие не создано. Тест пройден. Рисунок Д.12.
11	Развертывание события со статусом «Черновик»	Для события создаются необходимые объекты Kafka и ksqlDB. Статус события изменяется с «Черновик» на «Активен».	Событие развернуто. Необходимые объекты созданы, событию присвоен статус «Активен». Тест пройден. Рисунки Д.13 и Д.14.
12	Создание составного события с одним входным событием	Отображается сообщение о необходимости выбрать ровно два входных события. Составное событие не создается.	Отображено сообщение о необходимости выбора двух входных событий. Новое событие не создано. Тест пройден. Рисунок Д.15.
13	Создание и развертывание составного события в темпоральном конструкторе	Создается новое составное событие со статусом «Активное обнаружение».	Составное событие создано и отображено в списке со статусом «Активное обнаружение». Тест пройден. Рисунки Д.16 и Д.17.
14	Удаление неразвернутого события со статусом «Черновик»	После подтверждения удаления событие удаляется из системы и перестает отображаться в списке.	Событие удалено и больше не отображается в списке событий. Тест пройден. Рисунки Д.18 и Д.19.
15	Удаление развернутого события со статусом «Активное обнаружение»	Активное событие удаляется из системы и перестает отображаться в списке.	Активное событие, не являющееся входным для других событий, удалено и больше не отображается в списке событий. Тест пройден. Рисунки Д.20 и Д.21.
16	Авторизация оператора с корректными данными	Открывается главное окно операторского клиента.	Авторизация выполнена. Главное окно операторского клиента открыто. Тест пройден. Рисунок Д.22.
17	Проведение мониторинга одного активного события	Устанавливается WebSocket-соединение. Поступающие сообщения выбранного события отображаются в таблице мониторинга до момента остановки.	Мониторинг запущен. Сообщения выбранного события поступают и отображаются в таблице. Отчет сохранен. Тест пройден. Рисунки Д.23 и Д.24.
18	Запуск мониторинга без выбранных событий	Отображается сообщение о необходимости выбрать хотя бы одно событие. WebSocket-соединение не устанавливается.	Отображено сообщение о необходимости выбора события. Мониторинг не запущен. Тест пройден. Рисунок Д.25.

Окончание таблицы 27

1	2	3	4
19	Построение аналитики для периода с корректными датами	Отображается корректно построенная аналитика по выбранным событиям.	Отображена корректно построенная аналитика по выбранным событиям. Тест пройден. Рисунок Д.26.
20	Построение аналитики для периода с некорректными датами (дата окончания периода раньше даты начала)	Отображается сообщение об ошибке. Аналитический запрос не выполняется.	Отображено сообщение о некорректно заданном периоде. Аналитика не построена. Тест пройден. Рисунок Д.27.

Некоторые результаты тестирования представлены на рисунках 46 – 48.

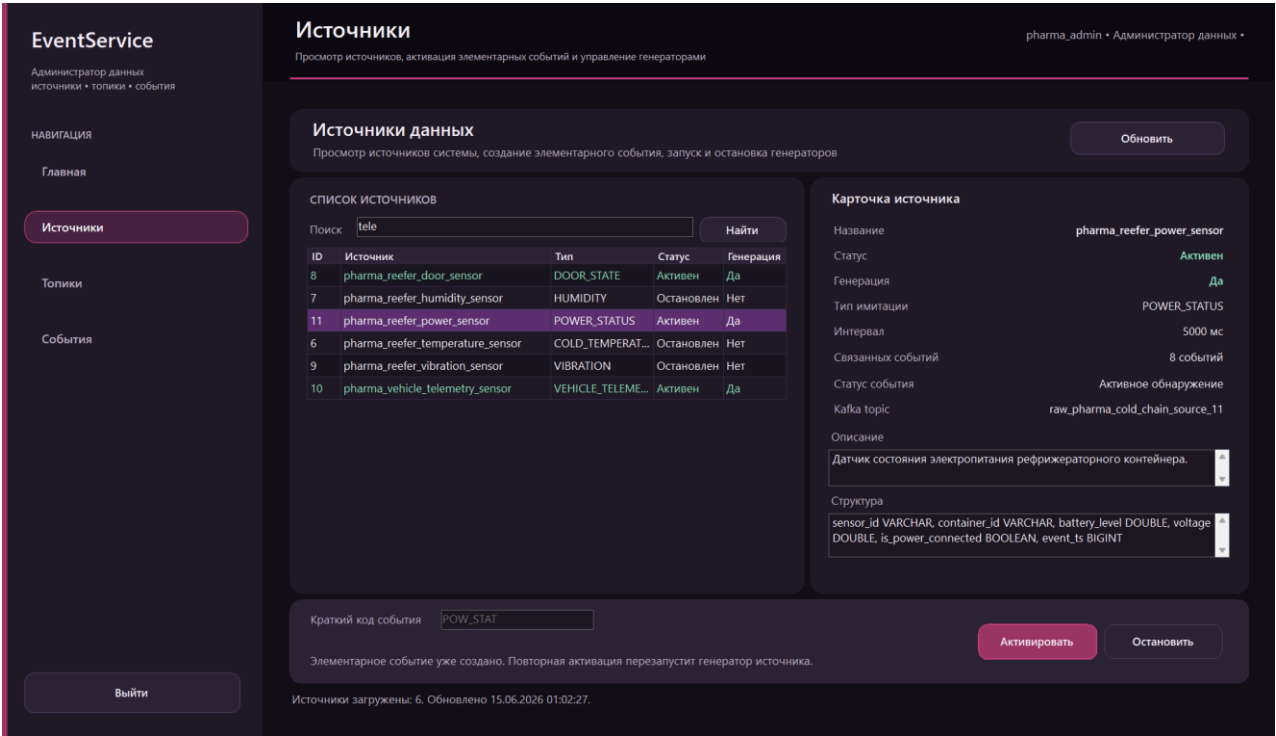


Рисунок 46 – Тест «Активация неактивного источника»

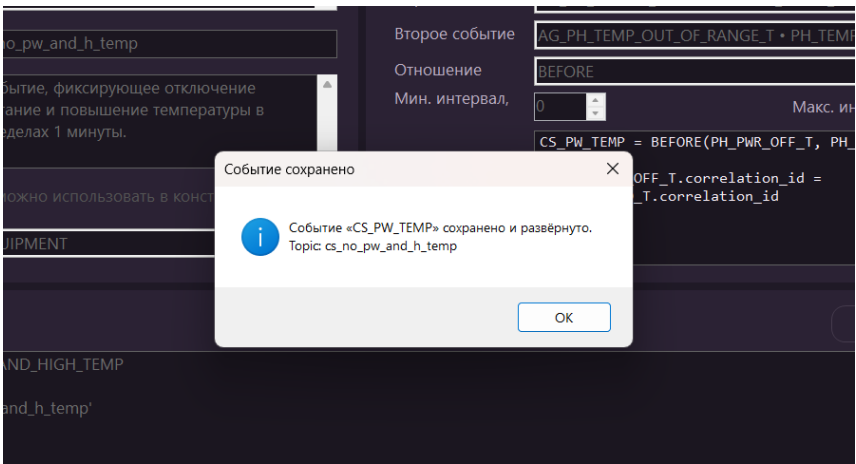


Рисунок 47 – Тест «Создание и развертывание составного события с помощью темпорального конструктора»

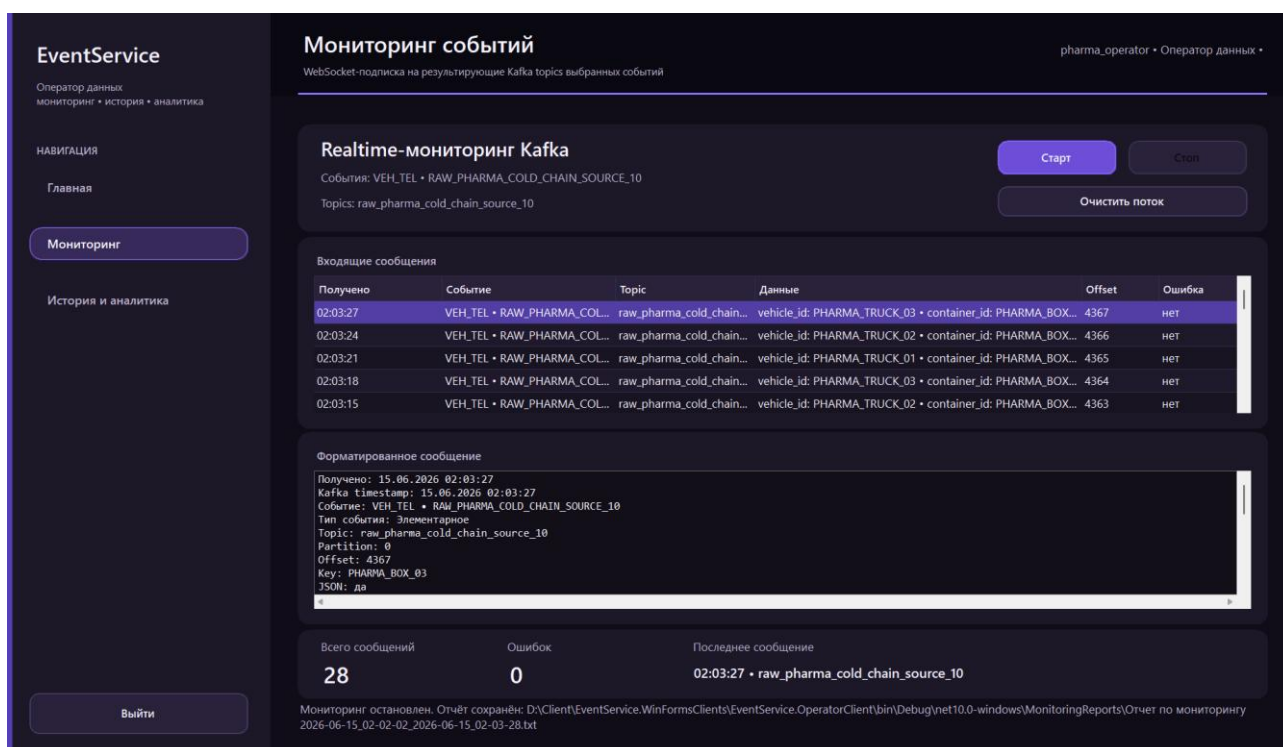


Рисунок 48 – Тест «Проведение мониторинга одного активного события»

В ходе тестирования было установлено, что расхождений между ожидаемым и фактическим результатом нет, что подтверждает корректность работы программных средств. Скриншоты результатов функционального тестирования отображены в приложении Д.

4.3 Оценка качества программного кода

Для количественной оценки сложности и качества программного кода была применена метрика Холстеда. Данная метрика основывается на подсчете количества операторов и операндов в исходном коде. Операторами считаются ключевые слова языка, знаки операций, скобки, точки, точки с запятой и другие синтаксические разделители. Операндами – имена переменных, константы и литералы [29].

Вычисление значений данной метрики позволяет разработчику определить, насколько хорошо структурирован код, и выявить потенциально проблемные модули, требующие рефакторинга.

Для тестирования был использован метод `ValidateBasicRequest` класса `TemporalEventService`. Метод предназначен для проверки основных параметров

запроса на создание или изменение временного составного события. Код представлен в листинге 8.

Листинг 8 – Код метода

```
private static void ValidateBasicRequest(SaveTemporalEventRequest
request)
{
    if (!ShortCodeRegex.IsMatch(request.ShortCode))
    {
        throw new ValidationException( "Краткий код должен
содержать только заглавные латинские буквы, цифры и символ
подчёркивания.");
    }

    if (!KsqlObjectNameRegex.IsMatch(request.Name))
    {
        throw new ValidationException( "Техническое имя события
имеет некорректный формат.");
    }

    if (!KafkaTopicNameRegex.IsMatch(request.ResultTopicName))
    {
        throw new ValidationException( "Имя результирующего topic
имеет некорректный формат.");
    }

    if (request.LeftEventId <= 0 || request.RightEventId <= 0)
    {
        throw new ValidationException( "Выберите два входных
события.");
    }
}
```

Для оценки сложности необходимо подсчитать общее и уникальное количество операторов и операндов рассматриваемого метода.

К операторам были отнесены ключевые слова языка C#, операции сравнения, логические операции и синтаксические конструкции. К операндам были отнесены имена метода, типов, параметров, полей, свойств, вызываемых методов и литералов.

Парные конструкции () и {} учитывались как единые операторы. Каждый символ кавычек строкового литерала также учитывался как оператор, а текст внутри кавычек рассматривался как отдельный операнд. Такой способ соответствует подходу, использованному в исходном примере расчёта.

Операторы метода ValidateBasicRequest представлены в таблице 28.

Таблица 28 – Операторы метода ValidateBasicRequest

Операторы	Число вхождений
public	1
static	1
void	1
()	12
{}	5
if	4
!	3
.	8
throw	4
new	4
;	4
<=	2
	1
Итого	50

Общее количество операндов составляет 50, а число уникальных операторов – 14. Операторы данного метода представлены в таблице 29.

Таблица 29 – Операнды метода ValidateBasicRequest

Операнды	Число вхождений
ValidateBasicRequest	1
SaveTemporalEventRequest	1
request	6
ShortCodeRegex	1
IsMatch	3
ShortCode	1
ValidationException	4
"Краткий код должен содержать только заглавные латинские буквы, цифры и символ подчёркивания."	1
KsqlObjectNameRegex	1
Name	1
"Техническое имя события имеет некорректный формат."	1
KafkaTopicNameRegex	1
ResultTopicName	1
"Имя результирующего topic имеет некорректный формат."	1
LeftEventId	1
0	2
RightEventId	1
"Выберите два входных события."	1
Итого	29

Длина программы рассчитывается по формуле:

$$N = N1 + N2 = 50 + 29 = 79,$$

где $N1$ – общее число операторов;

$N2$ – общее число операндов.

Словарь программы вычисляется по формуле:

$$n = n1 + n2 = 13 + 18 = 31,$$

где $n1$ – число уникальных операторов;

n_2 – число уникальных операндов.

Объем программы рассчитывается по формуле:

$$HV = N * \log_2(n) = 79 * \log_2(31) = 391.38,$$

где N – длина программы;

n – словарь программы.

Сложность программы определяется по формуле:

$$HD = (n_1/2) * (N_2/2) = (13/2) * (29/18) = 10.47,$$

где n_1 – число уникальных операторов;

N_2 – общее число операндов;

n_2 – число уникальных операндов.

Сложность процедуры приблизительно равна 10,47. На основании проведенной оценки можно сделать вывод, что метод `ValidateBasicRequest` обладает приемлемыми характеристиками сложности. Он выполняет важную для системы обнаружения событий функцию, содержит разнообразные программные конструкции и при этом остается компактным и понятным. Значение сложности 10,47 свидетельствует о том, что метод удобен для тестирования, сопровождения и дальнейшего расширения. Дополнительная декомпозиция рассматриваемого метода не требуется.

Заключение

Целью данного дипломного проекта была разработка программных средств для администрирования, мониторинга и обнаружения событий в системе потоковой обработки данных на платформе Apache Kafka.

На текущей стадии разработки созданы программные средства:

- модуль администрирования, обеспечивающий конфигурирование потоковой инфраструктуры, а именно: просмотр доступных в системе топиков, активацию передачи данных от источников данных в Kafka, конструирование сложных событий, настройку их параметров и активацию их обнаружения;
- модуль диспетчеризации, работающий на основе Kafka API и предоставляющий оператору возможность в реальном времени наблюдать за потоками событий, отслеживать динамику поступающих сообщений без внесения изменений в структуру системы;
- сервер RESTful API, необходимый для централизованного управления системой и осуществления бизнес-логики;
- сервер ksqlDB, позволяющий осуществлять потоковую обработку данных.

В рамках дипломного проекта для создания программных средств были разработаны два клиентских приложения, позволяющих полностью использовать функционал программных средств.

Все задачи, поставленные в рамках выполнения дипломного проекта, были выполнены в соответствии с установленными требованиями. Цель была достигнута.

Разработанные программные средства являются прототипом и могут расширяться в направлении предоставляемой ими функциональности. Планируется расширить набор доступных операций темпорального конструктора, добавить полноценный low-code конструктор для агрегированных и составных событий, предоставить возможность графического представления дерева состава события.

Список использованных источников

1. Архитектурные решения для обработки потоковых данных // OTUS [Электронный ресурс]. URL: <https://habr.com/ru/companies/otus/articles/757098/> (дата обращения: 21.04.2026).
2. Клеппман, М. Высоконагруженные приложения: программирование, масштабирование, поддержка / М. Клеппман ; пер. с англ. И. Пальти, А. Тумаркина. – СПб. : Питер, 2022. – 640 с.
3. Марц, Н. Большие данные: принципы и практика построения масштабируемых систем обработки данных в реальном времени / Н. Марц, Дж. Уоррен ; пер. с англ. – М. : Вильямс, 2016. – 368 с.
4. Самарев, Р. С. Обзор состояния области потоковой обработки данных / Р. С. Самарев // Труды ИСП РАН, том 29, вып. 1, 2017 г., С. 231-260.
5. Chakravarthy S., Krishnaprasad V., Anwar E., Kim S.-K. Anatomy of a Composite Event Detector // Tech. Report UF-CIS-TR-93-039. – Gainesville: University of Florida, Computer and Information Sciences Department, 1993. – 43 p.
6. Обработка сложных событий. Архитектура и другие практические аспекты [Электронный ресурс]. URL: <https://www.redpanda.com/guides/event-stream-processing-complex-event-processing> (дата обращения: 23.04.2026).
7. Обнаружение событий с помощью ksqlDB [Электронный ресурс]. URL: <https://www.confluent.io/blog/build-a-intrusion-detection-using-ksqldb/> (дата обращения: 28.04.2026).
8. Нгуен, Т. М. В. Спецификация потока задач в расширенной интервальной логике Аллена / Т. М. В. Нгуен, Г. С. Плесневич // Открытые семантические технологии проектирования интеллектуальных систем. – 2016. – С. 295–300.
9. AKHQ (KafkaHQ) // GitHub Repository [Электронный ресурс]. URL: <https://github.com/tchiotludo/akhq> (дата обращения: 03.05.2026).
10. Redpanda Console Documentation [Электронный ресурс]. URL: <https://docs.redpanda.com/redpanda-connect/redpanda-console/> (дата обращения: 03.05.2026).

11. Nussknacker Documentation [Электронный ресурс]. URL: <https://docs.nussknacker.io/scenariosAuthoring/introduction/> (дата обращения: 03.05.2026).
12. IBM Event Automation Documentation [Электронный ресурс]. URL: <https://www.ibm.com/docs/en/event-automation> (дата обращения: 04.05.2026).
13. IBM Event Automation Pricing on AWS Marketplace [Электронный ресурс]. URL: <https://aws.amazon.com/marketplace/seller-profile?id=ibm> (дата обращения: 04.05.2026).
14. Скопин, И. Н. Модели жизненного цикла программного обеспечения [Электронный ресурс] / И. Н. Скопин. – URL: <https://goo.su/oGNM1> (дата обращения: 05.05.2026).
15. Вигерс, К. Разработка требований к программному обеспечению / К. Вигерс, Дж. Битти ; пер. с англ. – 3-е изд. – М. : Русская редакция, 2019. – 736 с.
16. Балашова, И. Ю. Современные информационные технологии в проектировании программных систем и комплексов : учеб. пособие / И. Ю. Балашова, Д. В. Такташкин. – Пенза : Изд-во ПГУ, 2019. – 106 с.
17. Ричардс, М. Фундаментальный подход к программной архитектуре: паттерны, свойства, проверенные методы / М. Ричардс, Н. Форд. – СПб. : Питер, 2023. – 448 с.
18. Иванова, Л. В. Разработка пользовательских интерфейсов / Л. В. Иванова. – Витебск : ВГУ имени П. М. Машерова, 2024. – 29 с.
19. ГОСТ Р 59163-2020. Информационные технологии. Тестирование программного обеспечения. Термины и определения. – М. : Стандартинформ, 2020. – 12 с.
20. ГОСТ 28806-90. Качество программных средств. Термины и определения. – М. : Изд-во стандартов, 1991. – 14 с.
21. WebSocket. Принципы работы и использования [Электронный ресурс]. URL: <https://habr.com/ru/sandbox/171066/> (дата обращения: 07.05.2026).

22. CREATE TABLE statement in ksqlDB for Confluent Platform [Электронный ресурс]. URL: <https://goo.su/vnaCRcp> (дата обращения: 08.05.2026).

23. CREATE STREAM AS SELECT statement in ksqlDB for Confluent Platform [Электронный ресурс]. URL: <https://goo.su/qSsO> (дата обращения: 08.05.2026).

24. Лоре, А. Проектирование веб-API / А. Лоре ; пер. с англ. Д. А. Беликова. – М. : ДМК Пресс, 2020. – 440 с.

25. Асташкина, Т. И. EventService.Backend : репозиторий полного исходного кода серверного приложения [Электронный ресурс]. URL: <https://github.com/grenader77/EventService.Backend> (дата обращения: 25.05.2026).

26. Буч, Г. Язык UML : руководство пользователя / Г. Буч, Д. Рамбо, И. Якобсон. – 2-е изд. – М. : ДМК Пресс, 2017. – 496 с.

27. Асташкина, Т. И. EventService.Clients : репозиторий полного исходного кода клиентских приложений [Электронный ресурс]. URL: <https://github.com/grenader77/EventService.Clients> (дата обращения: 30.05.2026).

28. Белик, А. Г. Качество и надежность программных систем : учеб. пособие / А. Г. Белик, В. Н. Цыганенко. – Омск : Изд-во ОмГТУ, 2018. – 80 с.

29. Коцюба, И. Ю. Методы оценки и измерения характеристик информационных систем : учеб. пособие / И. Ю. Коцюба, А. В. Чунаев, А. Н. Шиков. – СПб. : Университет ИТМО, 2015. – 264 с.

30. Куликов, С. С. Тестирование программного обеспечения : базовый курс / С. С. Куликов. – 3-е изд. – Минск : Четыре четверти, 2020. – 312 с.

31. Пероцкая, В. Н. Основы тестирования программного обеспечения : учеб. пособие / В. Н. Пероцкая, Д. А. Градусов. – Владимир : Изд-во ВлГУ, 2017. – 100 с.

32. Бейзер, Б. Тестирование черного ящика: технологии функционального тестирования программного обеспечения / Б. Бейзер ; пер. с англ. – М. : Лори, 2004. – 320 с.

Приложение А
(справочное)
Глоссарий

Apache Kafka – распределенная система обмена сообщениями между серверными приложениями в режиме реального времени.

ksqlDB – потоковая SQL-база данных, работающая поверх Apache Kafka, которая позволяет описывать потоковые приложения с помощью привычного синтаксиса SQL.

RESTful API – архитектурный стиль для создания веб-сервисов, который определяет правила обмена данными между клиентом и сервером при помощи стандартных HTTP-методов (GET, POST, PUT, DELETE).

Агрегированное событие – подвид сложного события, которое формируется на основе агрегирующего запроса к входному потоку элементарных событий и является результатом применения вычислительной функции.

Интервальная алгебра Аллена – исчисление, определяющее тринадцать бинарных отношений, которые могут существовать между двумя произвольными временными интервалами.

Комплексная обработка событий (Complex Event Processing, CEP) – технология, которая позволяет анализировать и обрабатывать множество потоков информации в режиме реального времени, выявляя закономерности и значимые цепочки событий, распределенных во времени.

Сложное событие (complex event) – событие, которое фиксируется в результате выполнения операций над элементарными и/или другими сложными событиями; отражает значимые ситуации или закономерности, выявленные путем анализа совокупности входящих данных.

Составное событие – подвид сложного события, которое представляет собой комбинацию элементарных и/или сложных событий с возможным использованием временных ограничений и логических операций.

Элементарное событие – базовое, атомарное событие, фиксирующее единичный факт или изменение состояния объекта мониторинга в конкретный момент времени; поступает непосредственно от внешних наблюдаемых систем и не подлежит дальнейшей декомпозиции в рамках обрабатывающей системы.

Приложение Б
(рекомендуемое)
Сценарии вариантов использования

Таблица Б.1 – Сценарий варианта использования «Получить данные о состоянии объектов системы»

Прецедент	Получить данные о состоянии объектов системы
Краткое описание	Пользователь просматривает общее состояние объектов системы и доступность инфраструктурных компонентов
Главные актеры	Администратор данных
Второстепенные актеры	Нет
Предусловие	Пользователь авторизован в панели администратора
Основной поток	1) пользователь переходит на вкладку «Главная»; 2) система отображает сведения о текущем пользователе; 3) система отображает количество активных источников данных; 4) система отображает показатели по источникам, топикам и событиям; 5) пользователь просматривает количество активных объектов и объектов с ошибками; 6) пользователь просматривает таблицу состояния внешних компонентов; 7) система отображает доступность базы метаданных, Kafka и ksqlDB, а также время отклика и сообщение о состоянии.
Альтернативные потоки	7.а если один из компонентов недоступен, система отображает соответствующий статус и текст ошибки; 7.б если REST API не может получить часть данных, система выводит сообщение об ошибке в соответствующей области интерфейса.
Постусловие	Пользователь получил информацию о текущем состоянии системы и может перейти к дальнейшей работе

Таблица Б.2 – Сценарий варианта использования «Работать с источником данных»

Прецедент	Работать с источником данных
Краткое описание	Пользователь просматривает источники данных и управляет их активацией или деактивацией
Главные актеры	Администратор данных
Второстепенные актеры	Нет
Предусловие	Пользователь авторизован в панели администратора
Основной поток	1) пользователь переходит на вкладку «Источники»; 2) система отображает таблицу зарегистрированных источников данных; 3) пользователь выбирает необходимый источник; 4) система отображает метаданные выбранного источника; 5) пользователь нажимает кнопку активации или деактивации источника; 6) система выполняет действие через REST API; 7) система обновляет состояние источника в таблице; 8) система выводит результат операции в области логов.
Альтернативные потоки	5.а если источник активируется впервые, система запрашивает краткий код для автоматически создаваемого элементарного события; 6.а если не удалось выполнить активацию или деактивацию, система выводит сообщение об ошибке в логах; 6.б если Kafka недоступна, система сообщает о невозможности запуска или остановки передачи данных.
Постусловие	Состояние выбранного источника изменено или пользователь получил сообщение о причине невозможности выполнения действия

Таблица Б.3 – Сценарий варианта использования «Работать с топиком Kafka»

Прецедент	Работать с топиком Kafka
Краткое описание	Пользователь просматривает список топиков Kafka, их метаданные и/или историю статусов
Главные актеры	Администратор данных
Второстепенные актеры	Нет
Предусловие	Пользователь авторизован в панели администратора
Основной поток	1) пользователь переходит на вкладку «Топики»; 2) система отображает таблицу топиков Kafka; 3) пользователь выбирает необходимый топик; 4) система отображает карточку выбранного топика; 5) пользователь просматривает метаданные топика; 6) пользователь нажимает кнопку «История статусов»; 7) система открывает всплывающее окно с историей изменений статуса выбранного топика.
Альтернативные потоки	3.а если топик не выбран, кнопка просмотра истории статусов недоступна; 7.а если история статусов отсутствует, система отображает пустую таблицу; 7.б если при получении данных возникла ошибка, система выводит сообщение об ошибке.
Постусловие	Пользователь получил сведения о выбранном топике и/или его истории статусов

Таблица Б.4 – Сценарий варианта использования «Работать с событием»

Прецедент	Работать с событием
Краткое описание	Пользователь просматривает события, их метаданные, историю статусов и дерево состава
Главные актеры	Администратор данных
Второстепенные актеры	Нет
Предусловие	Пользователь авторизован в панели администратора
Основной поток	1) пользователь переходит на вкладку «События»; 2) система отображает таблицу созданных и развернутых событий; 3) пользователь выбирает нужное событие; 4) система отображает карточку события; 5) система отображает дерево состава события в нижней таблице; 6) пользователь просматривает метаданные события, его статус и состав; 7) при необходимости пользователь нажимает кнопку «История»; 8) система открывает окно истории статусов выбранного события.
Альтернативные потоки	6.а если выбранное событие является черновиком, пользователю доступны кнопки «Редактировать» и «Удалить draft»; 6.б если выбранное событие является активным, пользователю доступна кнопка «Удалить active» при условии, что событие не используется в составе других событий; 6.в если действие над событием запрещено установленными ограничениями, соответствующая кнопка блокируется; 8.а если история статусов отсутствует, система отображает пустую таблицу.
Постусловие	Пользователь получил сведения о событии, его составе и истории либо выполнил доступное действие над событием

Таблица Б.5 – Сценарий варианта использования «Осуществить анализ данных»

Прецедент	Осуществить анализ данных
Краткое описание	Пользователь получает статистику по выбранным событиям за заданный временной период
Главные актеры	Оператор данных
Второстепенные актеры	Нет
Предусловие	Пользователь авторизован в панели оператора; в системе существуют активные события, по которым можно получить статистику
Основной поток	1) пользователь переходит на вкладку «История и аналитика»; 2) система отображает список активных событий; 3) пользователь выбирает одно или несколько событий для анализа; 4) пользователь задает дату и время начала периода; 5) пользователь задает дату и время окончания периода; 6) пользователь указывает интервал группировки в минутах; 7) пользователь запускает построение аналитики; 8) система запрашивает данные через REST API; 9) система отображает количество сообщений по событиям в виде столбчатой диаграммы; 10) система отображает общий поток сообщений по временным интервалам; 11) система выводит итоговые показатели: всего сообщений, событий в отчете и пик за интервал.
Альтернативные потоки	3.а если пользователь выбирает больше допустимого количества событий, система запрещает выбор лишних событий; 7.а если события не выбраны, система выводит сообщение о необходимости выбора события; 7.б если период указан некорректно, система выводит сообщение об ошибке; 8.а если сервер не смог получить данные, система отображает сообщение об ошибке; 11.а если за выбранный период сообщения отсутствуют, система отображает нулевые показатели.
Постусловие	Пользователь получил аналитическую сводку по выбранным событиям за заданный период.

Приложение В

(обязательное)

Листинг серверной части программных средств

Листинг В.1 – Файл Program.cs

```

using EventService.Api.Constants;
using EventService.Api.Data;
using EventService.Api.Infrastructure;
using EventService.Api.Options;
using EventService.Api.Services;
using EventService.Api.Services.Admin;
using EventService.Api.Services.Operator;
using EventService.Api.Simulation;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Hosting;
using Microsoft.IdentityModel.Tokens;
using Microsoft.OpenApi;
using System.Security.Claims;
using System.Text;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();
// OpenAPI configuration is omitted.

string connectionString = builder.Configuration
    .GetConnectionString("EventServiceDb")
    ?? throw new InvalidOperationException(
        "Строка подключения EventServiceDb не найдена.");

builder.Services.AddDbContext<EventServiceDbContext>(options =>
    options.UseNpgsql(connectionString));

builder.Services.Configure<JwtOptions>(
    builder.Configuration.GetSection(JwtOptions.SectionName));

JwtOptions jwtOptions = builder.Configuration
    .GetSection(JwtOptions.SectionName)
    .Get<JwtOptions>()
    ?? throw new InvalidOperationException("Настройки JWT не
найденны.");

// Registration of application and infrastructure services is
omitted.

builder.Services
    .AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.MapInboundClaims = false;
        options.TokenValidationParameters = new
TokenValidationParameters
        {
            ValidateIssuer = true,

```

```

        ValidIssuer = jwtOptions.Issuer,
        ValidateAudience = true,
        ValidAudience = jwtOptions.Audience,
        ValidateLifetime = true,
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(
            Encoding.UTF8.GetBytes(jwtOptions.Key)),
        NameClaimType = ClaimTypes.Name,
        RoleClaimType = ClaimTypes.Role,
        ClockSkew = TimeSpan.Zero
    });
});

builder.Services.AddAuthorization(options =>
{
    options.AddPolicy(PolicyNames.AdministratorOnly,
        policy => policy.RequireRole(RoleNames.Administrator));
    options.AddPolicy(PolicyNames.OperatorOnly,
        policy => policy.RequireRole(RoleNames.Operator));
});

var app = builder.Build();
app.UseHttpsRedirection();
app.UseWebSockets();
app.UseAuthentication();
app.UseAuthorization();
app.MapControllers();
app.Run();

```

Листинг В.2 – Сервис TemporalEventService.cs

```

using System.ComponentModel.DataAnnotations;
using System.Text.RegularExpressions;
using EventService.Api.Constants;
using EventService.Api.Data;
using EventService.Api.DTOs.Admin;
using EventService.Api.Infrastructure;
using EventService.Api.Models;
using EventService.Api.Options;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Options;
using DbEvent = global::EventService.Api.Models.Event;

namespace EventService.Api.Services.Admin;

public sealed class TemporalEventService
{
    private const string TemporalTopicSchema =
        "correlation_id VARCHAR KEY, interval_start BIGINT, " +
        "interval_end BIGINT, left_interval_start BIGINT, " +
        "left_interval_end BIGINT, right_interval_start BIGINT, "
+
        "right_interval_end BIGINT";

```

```

private static readonly Regex ShortCodeRegex =
    new(@"^[A-Z][A-Z0-9_]{0,17}$", RegexOptions.Compiled);

private readonly EventServiceDbContext _dbContext;
private readonly CurrentUserService _currentUserService;
private readonly KsqlDbClient _ksqlDbClient;
private readonly KafkaOptions _kafkaOptions;

public TemporalEventService(
    EventServiceDbContext dbContext,
    CurrentUserService currentUserService,
    KsqlDbClient ksqlDbClient,
    IOptions<KafkaOptions> kafkaOptions)
{
    _dbContext = dbContext;
    _currentUserService = currentUserService;
    _ksqlDbClient = ksqlDbClient;
    _kafkaOptions = kafkaOptions.Value;
}

public async Task<int> CreateAsync(
    SaveTemporalEventRequest request,
    CancellationToken cancellationToken = default)
{
    PreparedTemporalEvent prepared = await PrepareAsync(
        request, true, null, null, cancellationToken);

    await using var transaction =
        await _dbContext.Database.BeginTransactionAsync(
            cancellationToken);

    try
    {
        var topic = new Topic
        {
            Name = request.ResultTopicName,
            TopicSchema = TemporalTopicSchema,
            Partitions = _kafkaOptions.DefaultPartitions,
            ReplicationFactor =
                _kafkaOptions.DefaultReplicationFactor,
            SystemId = _currentUserService.SystemId
        };
        _dbContext.Topics.Add(topic);
        await _dbContext.SaveChangesAsync(cancellationToken);

        var eventItem = new DbEvent
        {
            Name = request.Name,
            ShortCode = request.ShortCode,
            KsqlText = prepared.GeneratedSql,
            ResultTopicId = topic.Id,
            SystemId = _currentUserService.SystemId,
            EventContextId = prepared.ContextId
        }
    }
}

```

```

        };
        _dbContext.Events.Add(eventItem);

        // Input links, temporal rule and status history are
added here.
        await _dbContext.SaveChangesAsync(cancellationToken);
        await transaction.CommitAsync(cancellationToken);
        return eventItem.Id;
    }
    catch
    {
        await transaction.RollbackAsync(cancellationToken);
        throw;
    }
}

// Метода валидации, генерации SQL -скриптов, обновления и
развертывания события.
}

```

Листинг В.2 – Сервис KafkaMonitoringStreamService.cs

```

using System.Net.WebSockets;
using System.Text.Json;
using Confluent.Kafka;
using EventService.Api.DTOS.Operator;
using EventService.Api.Options;
using Microsoft.Extensions.Options;

namespace EventService.Api.Infrastructure;

public sealed class KafkaMonitoringStreamService
{
    private static readonly JsonSerializerOptions
WebSocketJsonOptions =
        new(JsonSerializerDefaults.Web) { WriteIndented = false };

    private readonly KafkaOptions _kafkaOptions;
    private readonly ILogger<KafkaMonitoringStreamService>
_logger;

    public KafkaMonitoringStreamService(
        IOptions<KafkaOptions> kafkaOptions,
        ILogger<KafkaMonitoringStreamService> logger)
    {
        _kafkaOptions = kafkaOptions.Value;
        _logger = logger;
    }

    public async Task StreamAsync(
        WebSocket webSocket,
        MonitoringConfigurationDto configuration,
        CancellationToken cancellationToken = default)
    {

```

```

if (configuration.DistinctTopicNames.Count == 0)
{
    await SendServiceMessageAsync(
        websocket, "error", "Kafka topics не найдены.",
        cancellationToken);
    await CloseIfOpenAsync(websocket,
        WebSocketCloseStatus.InvalidPayloadData,
        "topics not found");
    return;
}

var consumerConfig = new ConsumerConfig
{
    BootstrapServers = _kafkaOptions.BootstrapServers,
    GroupId = configuration.ConsumerGroupId,
    AutoOffsetReset = AutoOffsetReset.Latest,
    EnableAutoCommit = true,
    AllowAutoCreateTopics = false
};

using IConsumer<string, string> consumer =
    new ConsumerBuilder<string,
string>(consumerConfig).Build();

try
{
    consumer.Subscribe(configuration.DistinctTopicNames);

    while (websocket.State == WebSocketState.Open
        && !cancellationToken.IsCancellationRequested)
    {
        ConsumeResult<string, string>? result =
            await ConsumeNextAsync(consumer,
cancellationToken);

        if (result?.Message is null)
        {
            continue;
        }

        MonitoringStreamMessageDto message = BuildMessage(
            result, null, configuration.Events);

        await SendMessageAsync(
            websocket, message, cancellationToken);
    }
}
catch (OperationCanceledException)
{
    // Normal monitoring shutdown.
}
finally
{

```

```
        consumer.Close();  
        await CloseIfOpenAsync(webSocket,  
                                WebSocketCloseStatus.NormalClosure,  
                                "monitoring stopped");  
    }  
}  
  
    // Конвертация сообщений и вспомогательные метода для  
WebSocket.  
}
```

Приложение Г
(обязательное)

Листинг клиентской части программных средств

Листинг Г.1 – EventServiceApiClient.cs

```

using System.Net.Http.Headers;
using System.Text;
using System.Text.Json;
using EventService.Client.Core.Dto;
using System.Diagnostics;

namespace EventService.Client.Core;

public sealed class EventServiceApiClient : IDisposable
{
    private static readonly JsonSerializerOptions JsonOptions =
        new(JsonSerializerDefaults.Web)
        {
            PropertyNameCaseInsensitive = true,
            PropertyNamingPolicy = JsonNamingPolicy.CamelCase
        };

    private readonly HttpClient _httpClient;

    public EventServiceApiClient(string baseUrl)
    {
        if (string.IsNullOrEmpty(baseUrl))
        {
            throw new ArgumentException(
                "Адрес REST API не указан.", nameof(baseUrl));
        }

        BaseUrl = NormalizeBaseUrl(baseUrl);
        _httpClient = new HttpClient(
            new HttpClientHandler { AllowAutoRedirect = false })
        {
            BaseAddress = new Uri(BaseUrl),
            Timeout = TimeSpan.FromSeconds(30)
        };
    }

    public string BaseUrl { get; }
    public string? AccessToken { get; private set; }

    public void SetBearerToken(string? accessToken)
    {
        AccessToken = string.IsNullOrEmpty(accessToken)
            ? null
            : accessToken;

        _httpClient.DefaultRequestHeaders.Authorization =
            AccessToken is null
                ? null
                : new AuthenticationHeaderValue("Bearer",
AccessTokен);
    }

```

```

public Task<TResponse> GetAsync<TResponse>(
    string relativeUrl,
    CancellationToken cancellationToken = default)
{
    return SendAsync<TResponse>(
        HttpMethod.Get, relativeUrl, null, cancellationToken);
}

public Task<TResponse> PostAsync<TRequest, TResponse>(
    string relativeUrl,
    TRequest request,
    CancellationToken cancellationToken = default)
{
    return SendAsync<TResponse>(
        HttpMethod.Post, relativeUrl, request,
cancellationToken);
}

private async Task<TResponse> SendAsync<TResponse>(
    HttpMethod method,
    string relativeUrl,
    object? requestBody,
    CancellationToken cancellationToken)
{
    using var request = new HttpRequestMessage(method,
relativeUrl);
    if (requestBody is not null)
    {
        string json = JsonSerializer.Serialize(requestBody,
JsonOptions);
        request.Content = new StringContent(
            json, Encoding.UTF8, "application/json");
    }

    using HttpResponseMessage response =
        await _httpClient.SendAsync(request,
cancellationToken);
    string text = await response.Content.ReadAsStringAsync(
        cancellationToken);

    if (!response.IsSuccessStatusCode)
    {
        throw new ApiException(response.StatusCode,
            ExtractApiErrorMessage(text) ??
response.ReasonPhrase,
            text);
    }

    return JsonSerializer.Deserialize<TResponse>(text,
JsonOptions)
        ?? throw new InvalidOperationException(
            "Не удалось прочитать JSON-ответ REST API.");
}

```

```

    }

    // PUT, DELETE

    public void Dispose()
    {
        _httpClient.Dispose();
    }
}

```

Листинг Г.2 – OperatorMonitoringWebSocketClient.cs

```

using System.Net.WebSockets;
using System.Text;
using System.Text.Json;
using EventService.Client.Core.Dto;

namespace EventService.Client.Core;

public sealed class OperatorMonitoringWebSocketClient :
    IDisposable
{
    private static readonly JsonSerializerOptions JsonOptions =
        new(JsonSerializerDefaults.Web)
        {
            PropertyNameCaseInsensitive = true
        };

    private readonly string _baseUrl;
    private readonly string? _accessToken;
    private ClientWebSocket? _webSocket;

    public OperatorMonitoringWebSocketClient(
        string baseUrl,
        string? accessToken)
    {
        if (string.IsNullOrEmpty(baseUrl))
        {
            throw new ArgumentException(
                "Адрес REST API не указан.", nameof(baseUrl));
        }

        _baseUrl = baseUrl;
        _accessToken = accessToken;
    }

    public async Task RunAsync(
        IReadOnlyCollection<int> eventIds,
        Func<MonitoringStreamMessageDto, Task> onMessageAsync,
        CancellationToken cancellationToken = default)
    {
        _webSocket?.Dispose();
        _webSocket = new ClientWebSocket();
    }
}

```

```

if (!string.IsNullOrEmpty(_accessToken))
{
    _webSocket.Options.SetRequestHeader(
        "Authorization", "Bearer " + _accessToken);
}

await _webSocket.ConnectAsync(
    BuildMonitoringUri(eventIds), cancellationToken);

var buffer = new byte[16 * 1024];

while (_webSocket.State == WebSocketState.Open
    && !cancellationToken.IsCancellationRequested)
{
    using var messageBuffer = new MemoryStream();
    WebSocketReceiveResult result;

    do
    {
        result = await _webSocket.ReceiveAsync(
            buffer, cancellationToken);
        if (result.MessageType ==
WebSocketMessageType.Close)
        {
            await CloseAsync();
            return;
        }
        messageBuffer.Write(buffer, 0, result.Count);
    }
    while (!result.EndOfMessage);

    if (result.MessageType == WebSocketMessageType.Text)
    {
        string json = Encoding.UTF8.GetString(
            messageBuffer.ToArray());
        MonitoringStreamMessageDto? message =
JsonSerializer.Deserialize<MonitoringStreamMessageDto>(
            json, JsonSerializerOptions);
        if (message is not null)
        {
            await onMessageAsync(message);
        }
    }
}

public async Task CloseAsync()
{
    if (_webSocket?.State == WebSocketState.Open)
    {
        await _webSocket.CloseAsync(
            WebSocketCloseStatus.NormalClosure,

```

```

        "monitoring stopped by operator",
        CancellationToken.None);
    }
}

public void Dispose()
{
    _webSocket?.Dispose();
}
}

```

Листинг Г.3 – Форма EventEditorForm.cs

```

using System.Text.RegularExpressions;
using EventService.Client.Core;
using EventService.Client.Core.Dto;

namespace EventService.AdminClient.Forms;

public partial class EventEditorForm : Form
{
    private const string AggregatedScenario =
        "Агрегированное событие";
    private const string CompositeScenario =
        "Сложное составное событие";
    private const string CompositeManualMode = "Ручной SQL";
    private const string CompositeTemporalMode =
        "Временной конструктор Аллена";

    private readonly AuthSession _session;
    private int? _eventId;
    private readonly List<InputEventListItem> _availableInputs =
new();
    private readonly List<ContextListItem> _contexts = new();
    private readonly HashSet<int> _selectedCompositeInputIds =
new();

    private bool _isBusy;
    private bool _isLoadingLookups;
    private bool _isUpdatingSqlTemplate;
    private bool _sqlTemplateManaged = true;

    public EventEditorForm(AuthSession session)
    {
        _session = session;

        InitializeComponent();
        ConfigureTemporalConstructorLayout();
        ApplyVisualStyle();
        WireEvents();
        InitializeStaticValues();
        UpdateMode();
        UpdateTemporalState();
        RefreshSqlTemplate(force: true);
    }
}

```

```

    }

    public EventEditorForm(AuthSession session, int eventId)
        : this(session)
    {
        _editEventId = eventId;
    }

    public int? CreatedEventId { get; private set; }
    public int? SavedEventId { get; private set; }
    public bool IsEditMode => _editEventId.HasValue;

    private void RefreshSqlTemplate(bool force)
    {
        if (!force && !_sqlTemplateManaged)
        {
            return;
        }

        _isUpdatingSqlTemplate = true;
        try
        {
            txtSql.Text =
NormalizeSqlLineEndings(BuildSqlTemplate());
            _sqlTemplateManaged = true;
        }
        finally
        {
            _isUpdatingSqlTemplate = false;
        }
    }

    private string BuildSqlTemplate()
    {
        if (IsAggregatedScenario())
        {
            return BuildAggregatedSqlTemplate();
        }

        if (IsCompositeTemporalMode())
        {
            RefreshTemporalExpression();
            return BuildTemporalSqlTemplate();
        }

        return BuildCompositeManualSqlTemplate();
    }

    // Загрузка, валидация, сохранение и вспомогательные методы
    для работы с SQL
}

```

Приложение Д
(рекомендуемое)
Результаты функционального тестирования

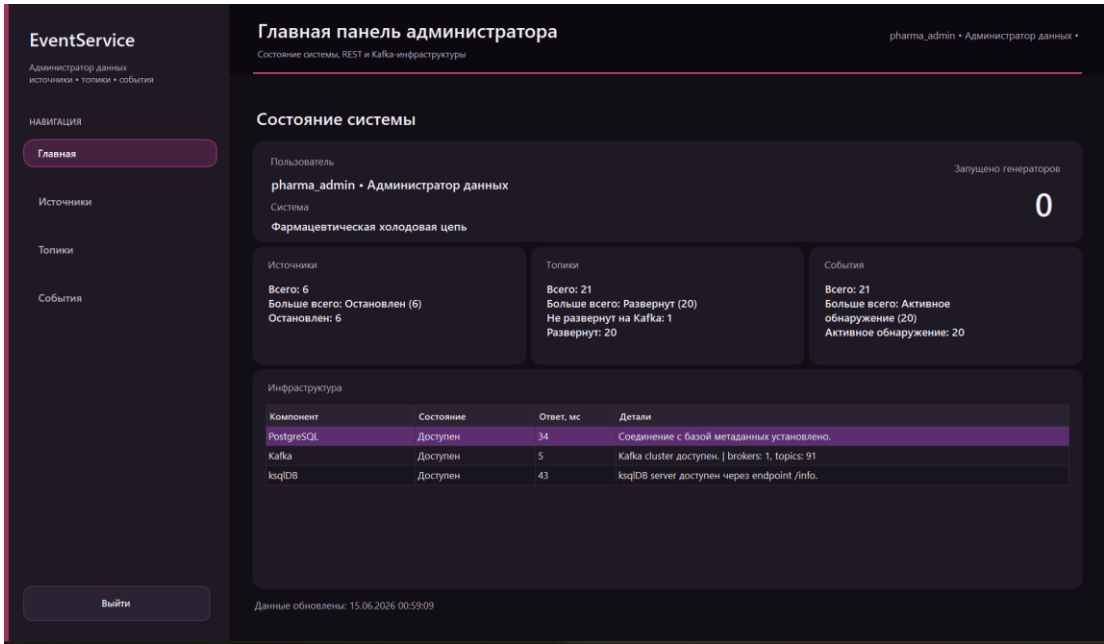


Рисунок Д.1 – Успешная авторизация администратора

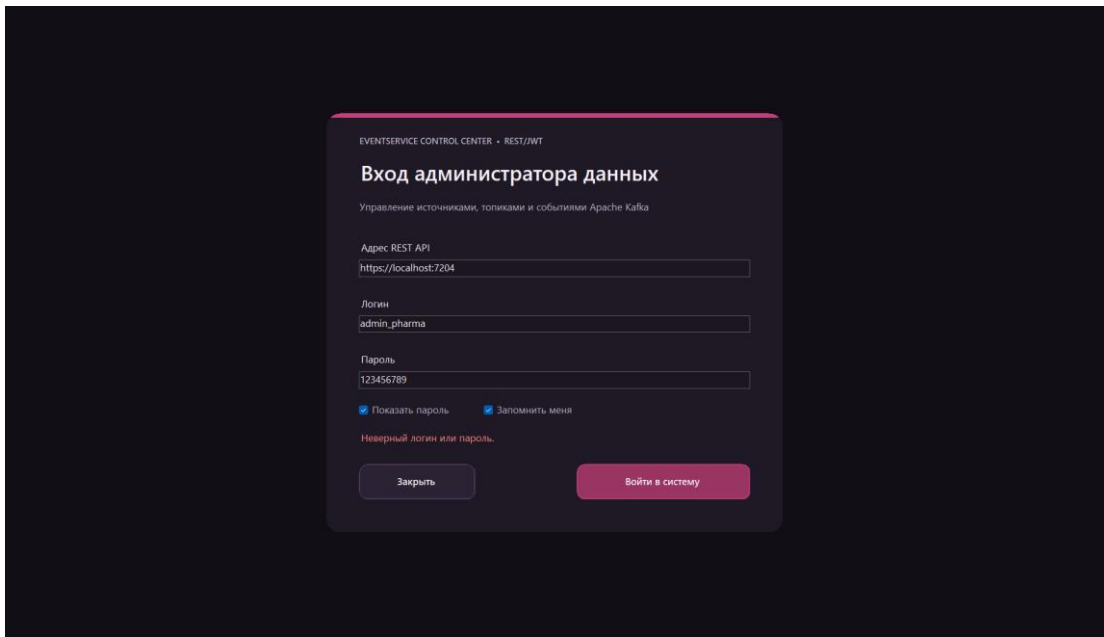


Рисунок Д.2 – Ошибка авторизации администратора

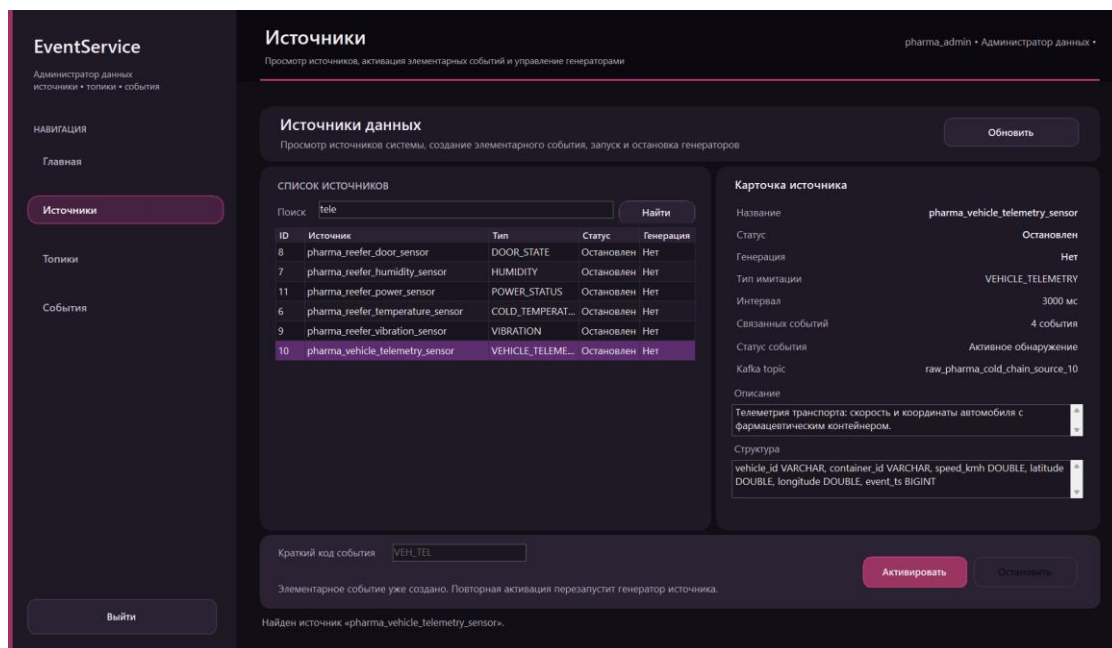


Рисунок Д.3 – Успешный поиск источника по названию

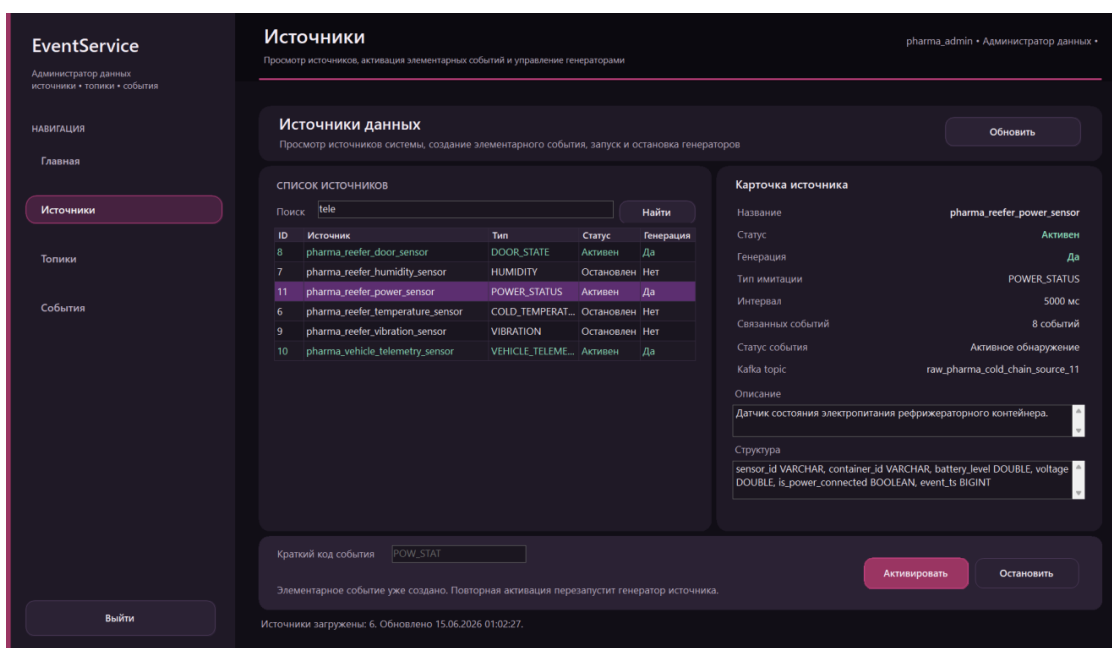


Рисунок Д.4 – Успешная активация источника данных

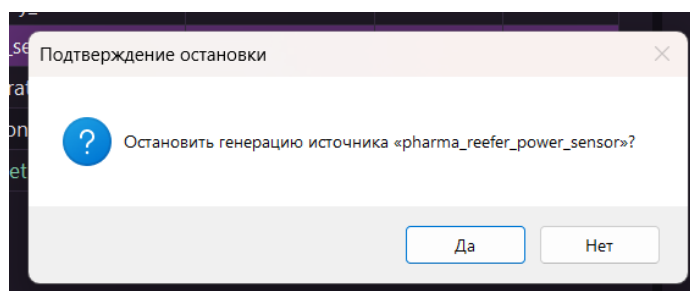


Рисунок Д.5 – Подтверждение остановки источника

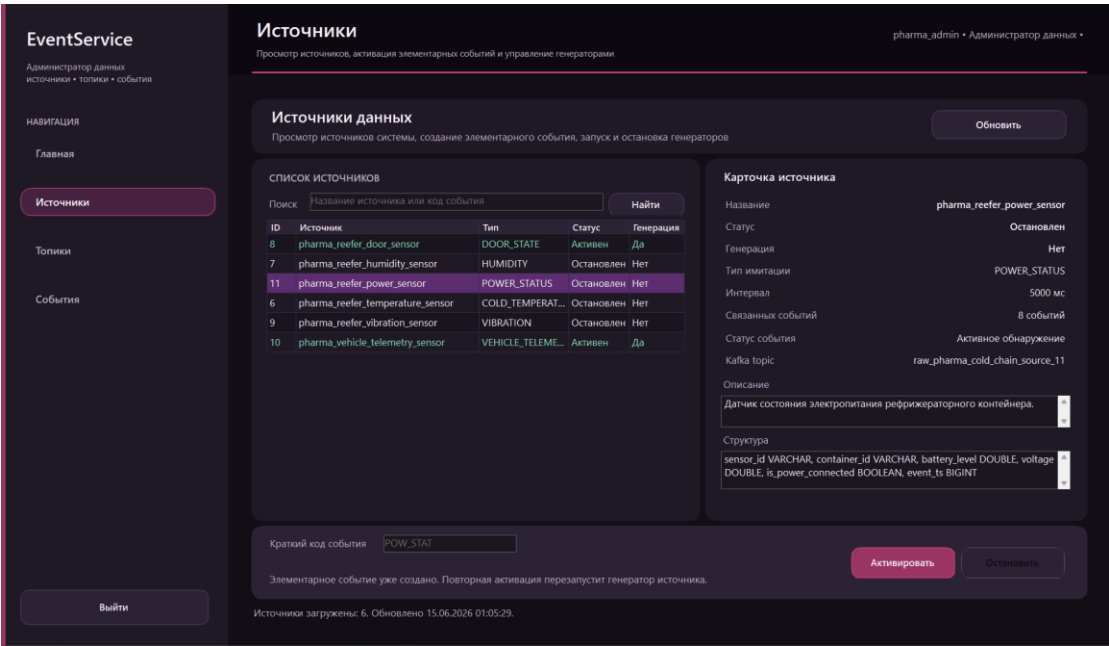


Рисунок Д.6 – Успешная остановка выбранного источника

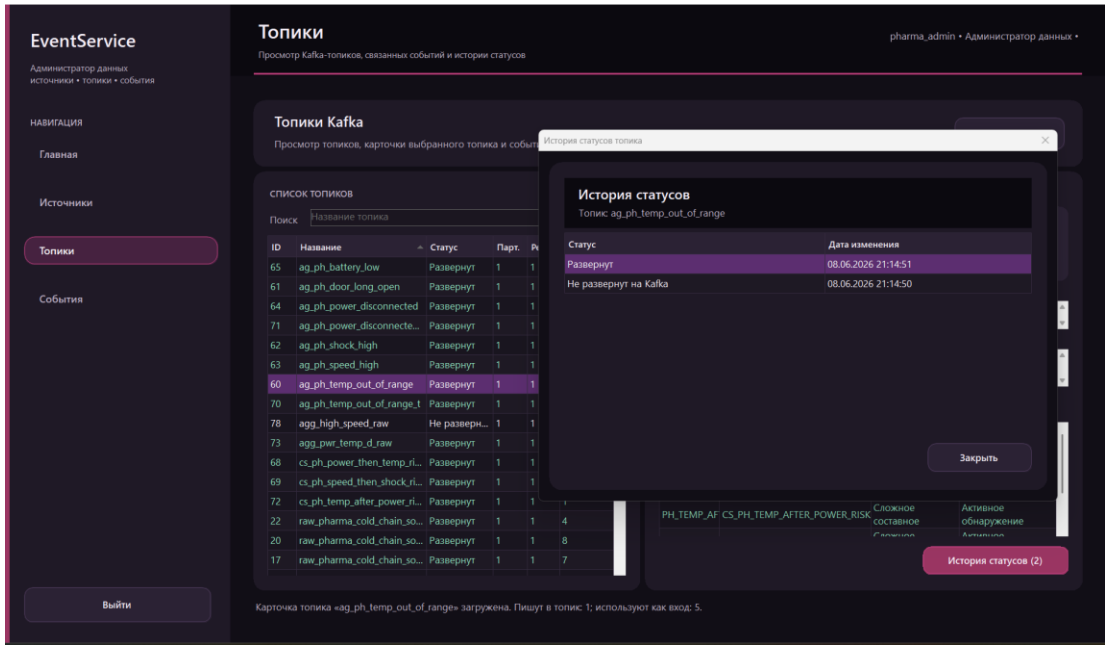


Рисунок Д.7 – Просмотр истории состояний Kafka топика

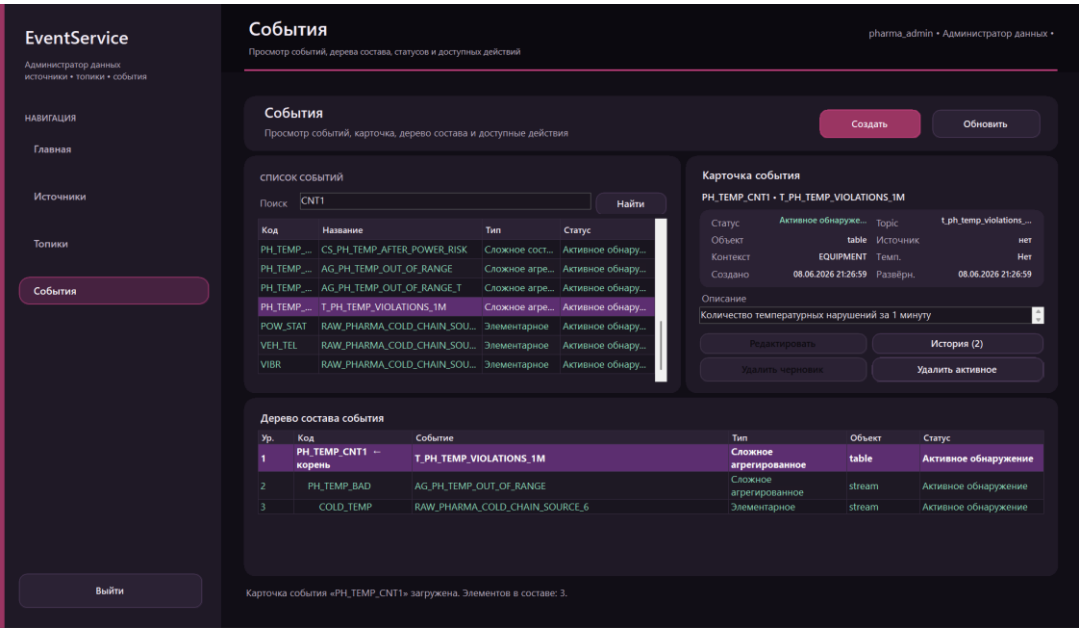


Рисунок Д.8 – Успешный поиск события по краткому коду

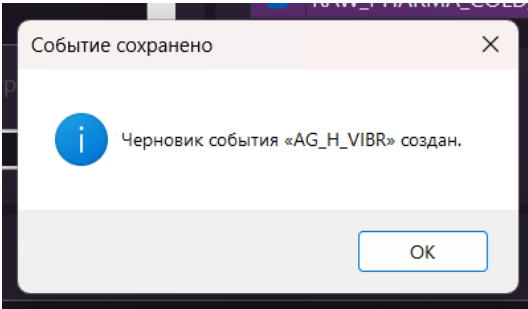


Рисунок Д.9 – Уведомление о сохранении черновика

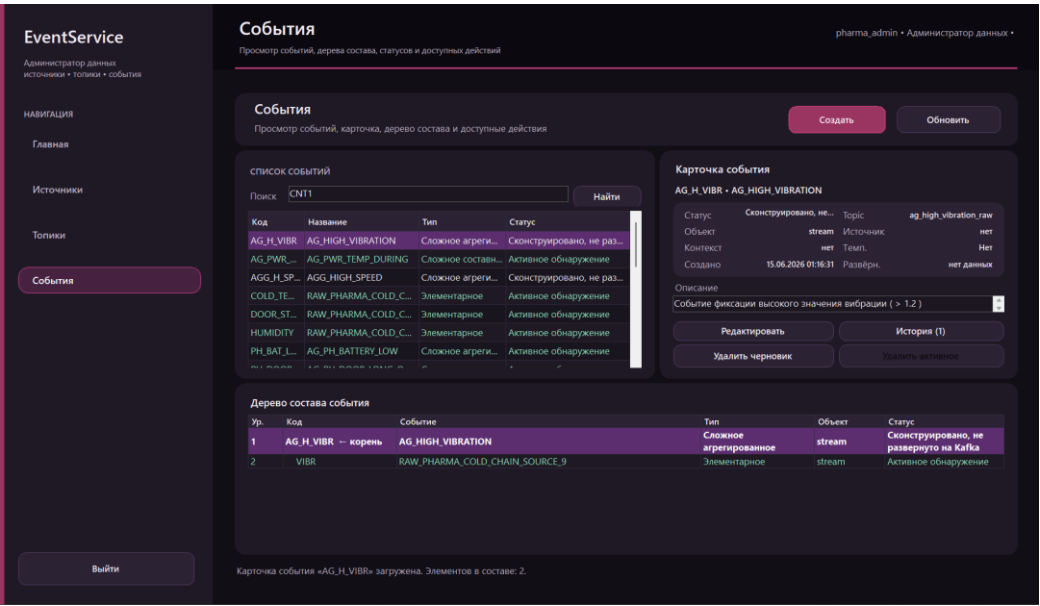


Рисунок Д.10 – Успешное сохранение черновика события

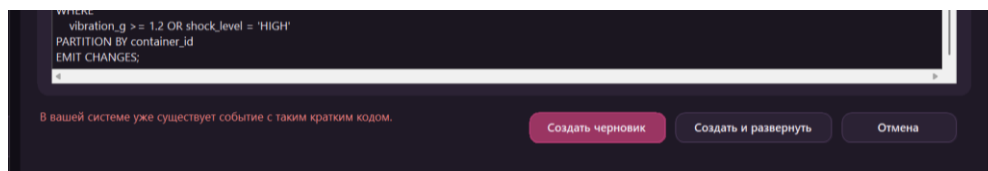


Рисунок Д.11 – Ошибка при повторном использовании краткого кода события

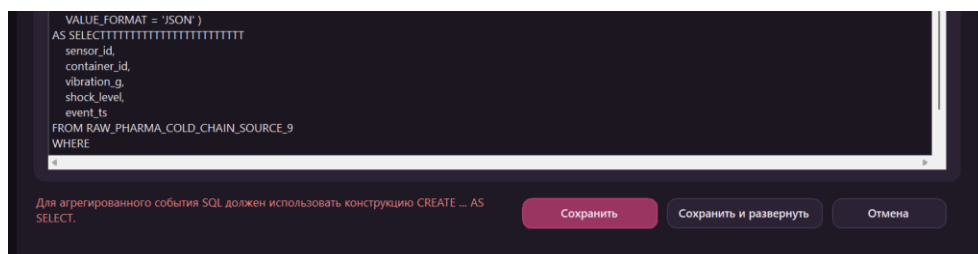


Рисунок Д.12 – Ошибка в коде SQL-запроса

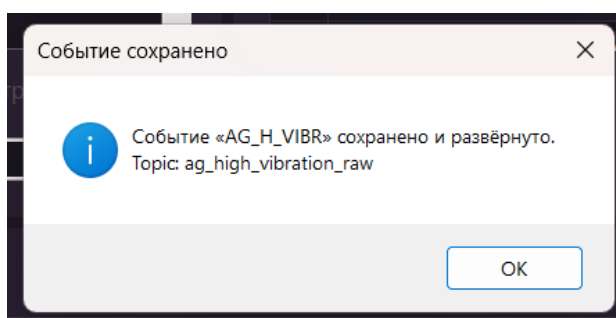


Рисунок Д.13 – Уведомление о успешно развернутом агрегированном событии

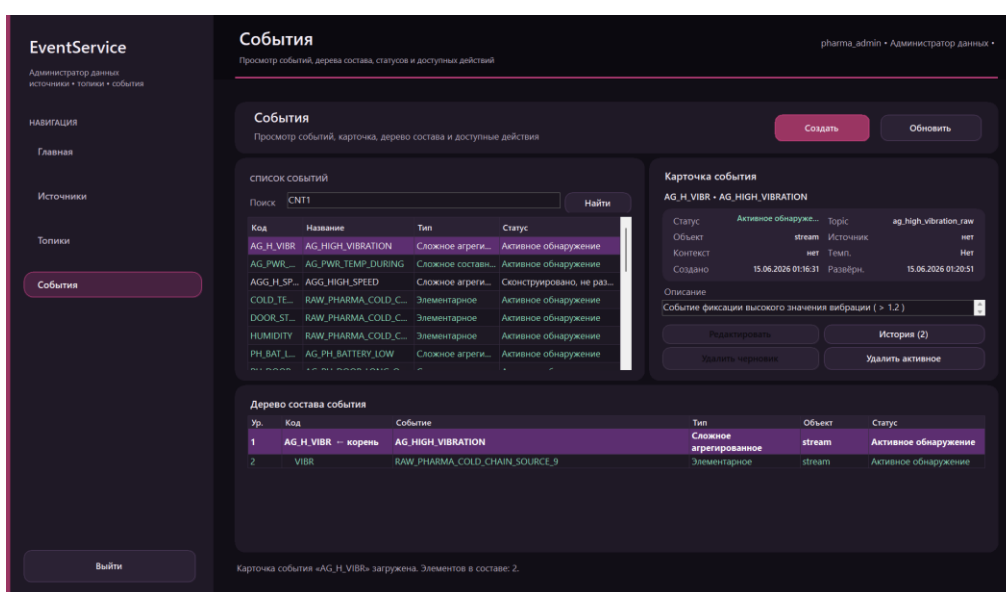


Рисунок Д.14 – Успешное сохранение и развертывание агрегированного события

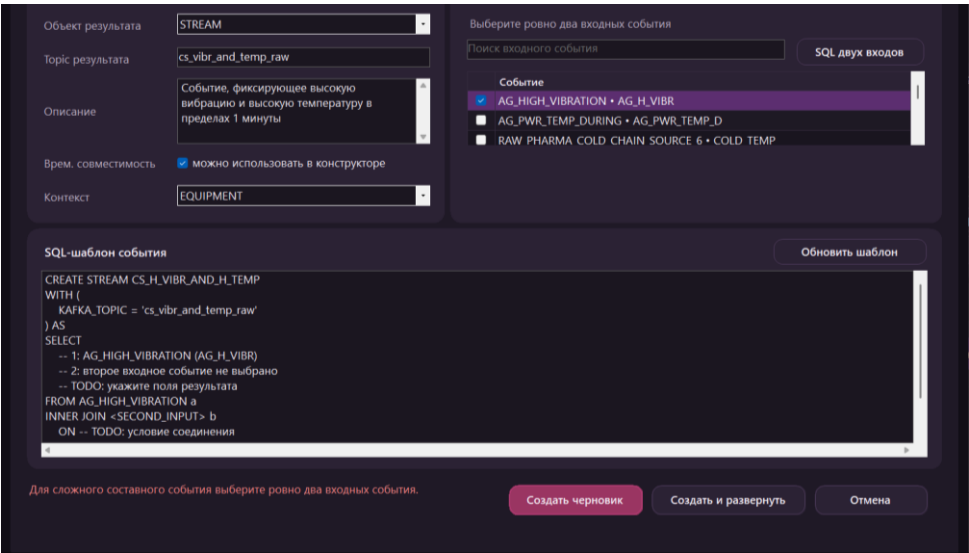


Рисунок Д.15 – Ошибка использования одного входа вместо двух для составного события

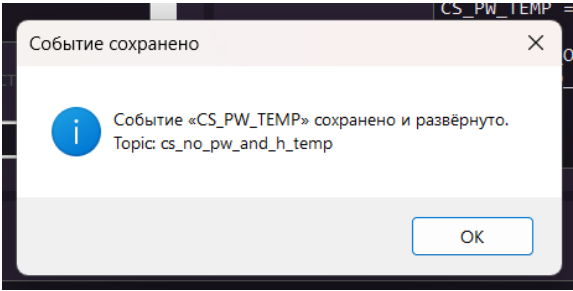


Рисунок Д.16 – Уведомление о успешно развернутом темпоральном составном событии

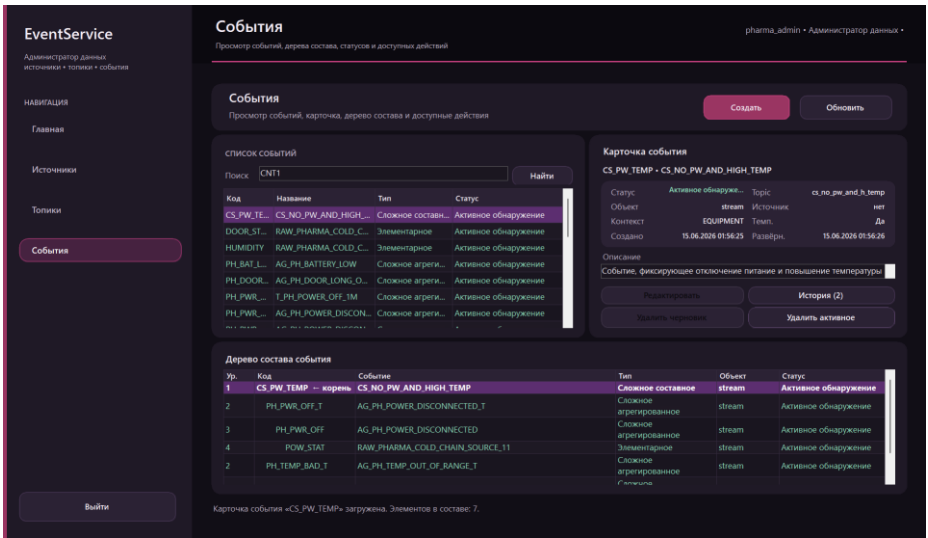


Рисунок Д.17 – Успешное сохранение и развертывание темпорального составного события

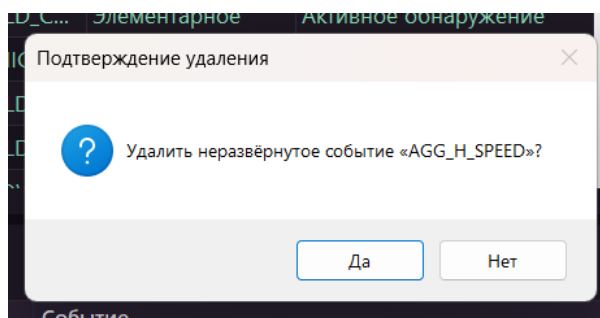


Рисунок Д.18 – Подтверждение удаления черновика события

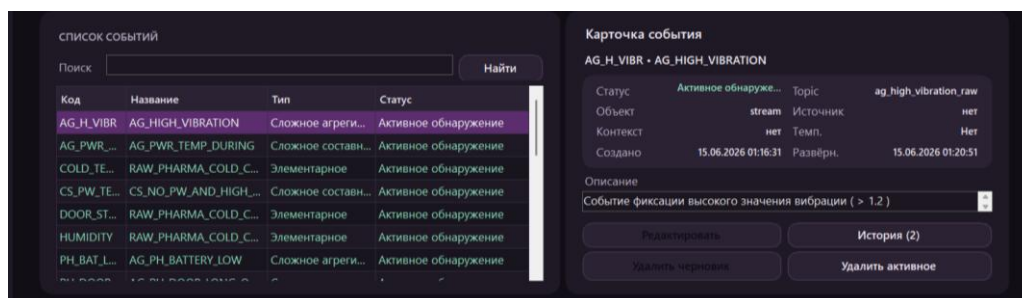


Рисунок Д.19 – Успешное удаление черновика события

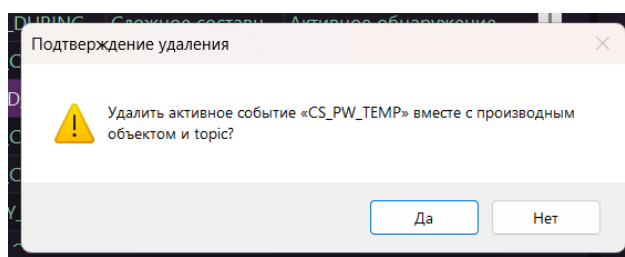


Рисунок Д.20– Подтверждение удаления активного события

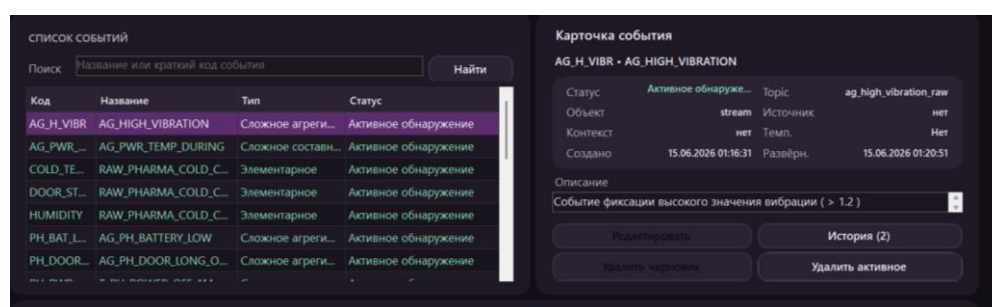


Рисунок Д.21– Успешное удаление активного события

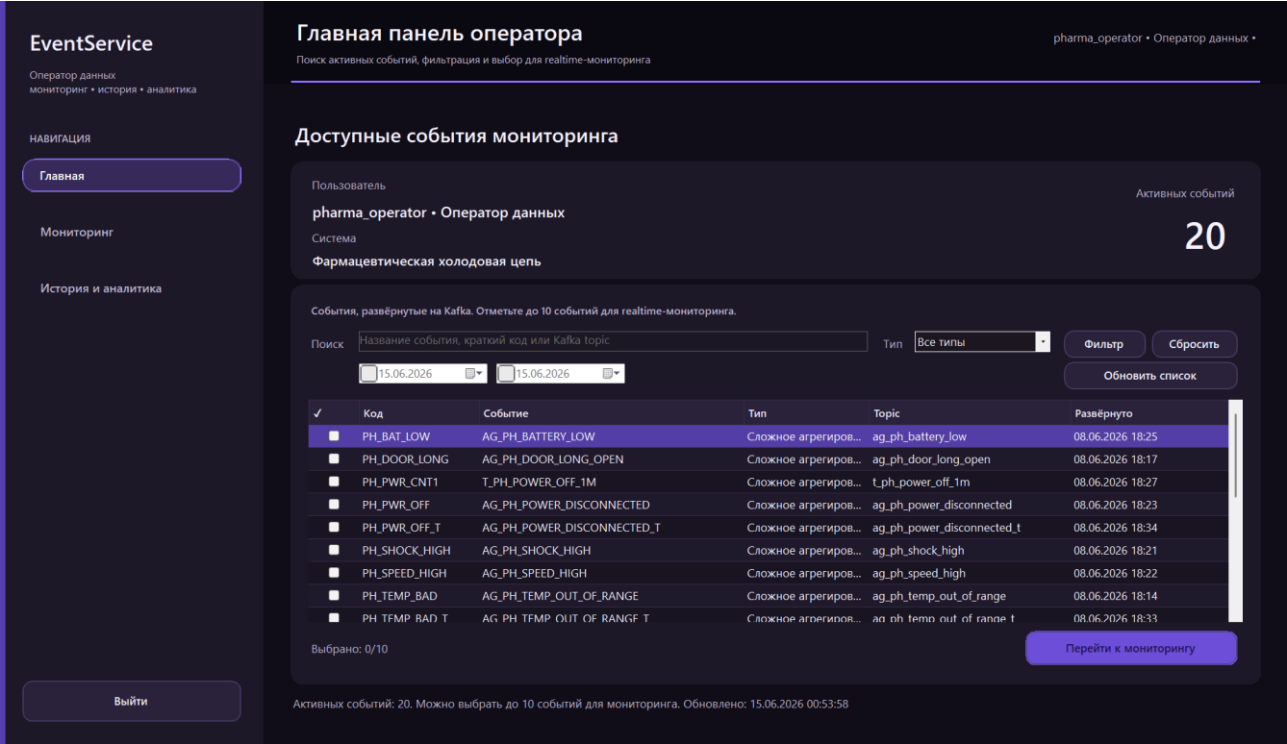


Рисунок Д.22 – Успешная авторизация оператора

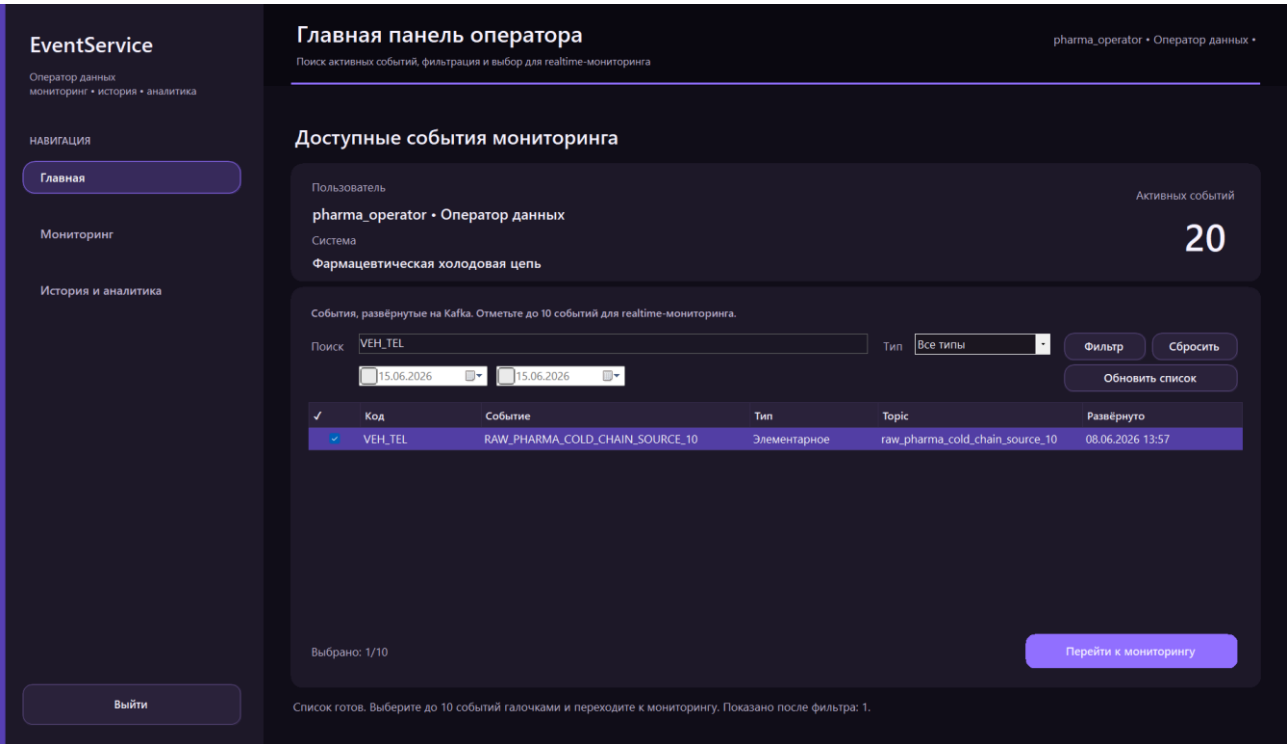


Рисунок Д.23 – Выбор одного события для мониторинга на главной панели оператора

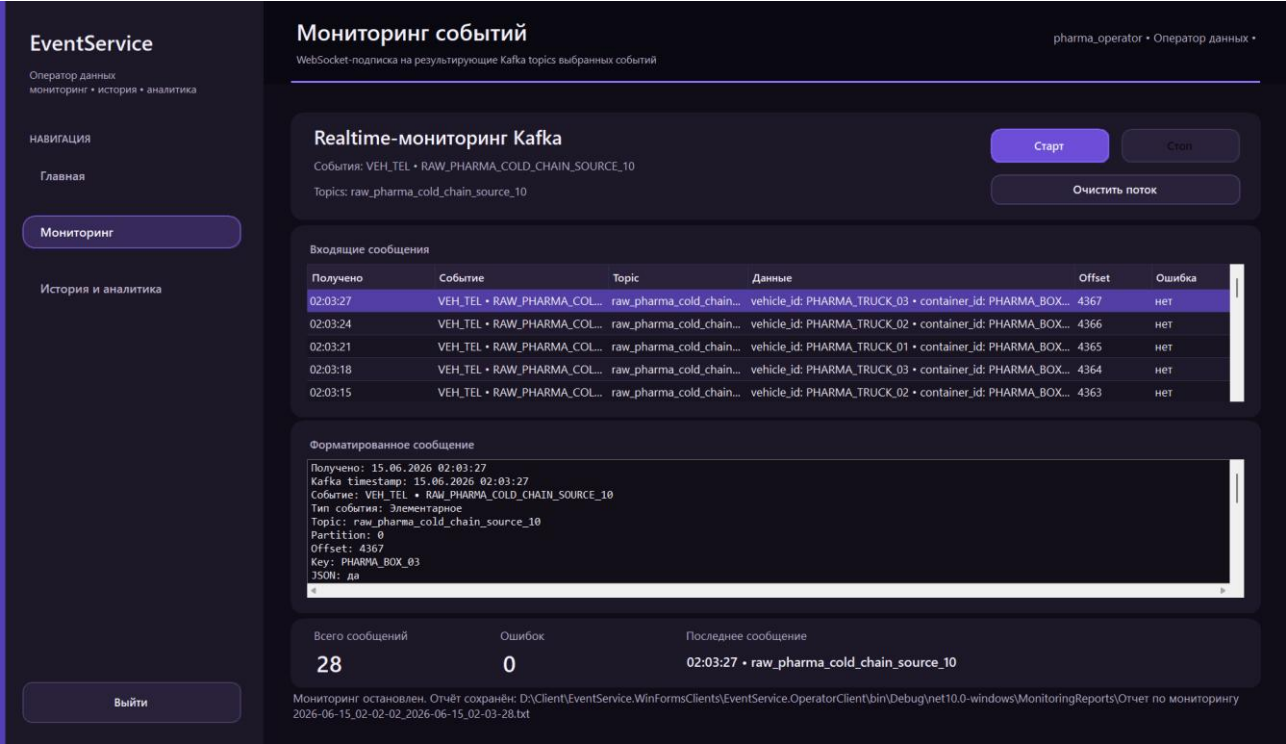


Рисунок Д.24 – Успешное проведение мониторинга с сохранением отчета

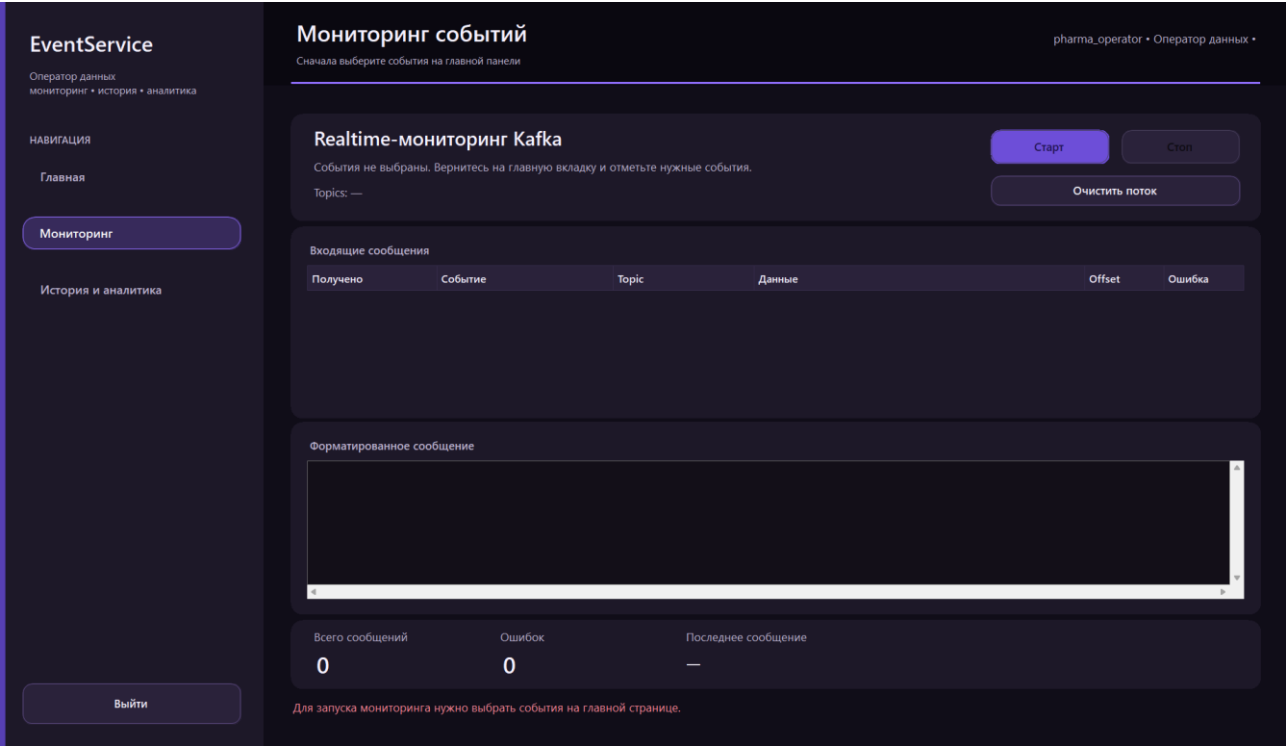


Рисунок Д.25 – Ошибка проведения мониторинга при отсутствии выбранных событий

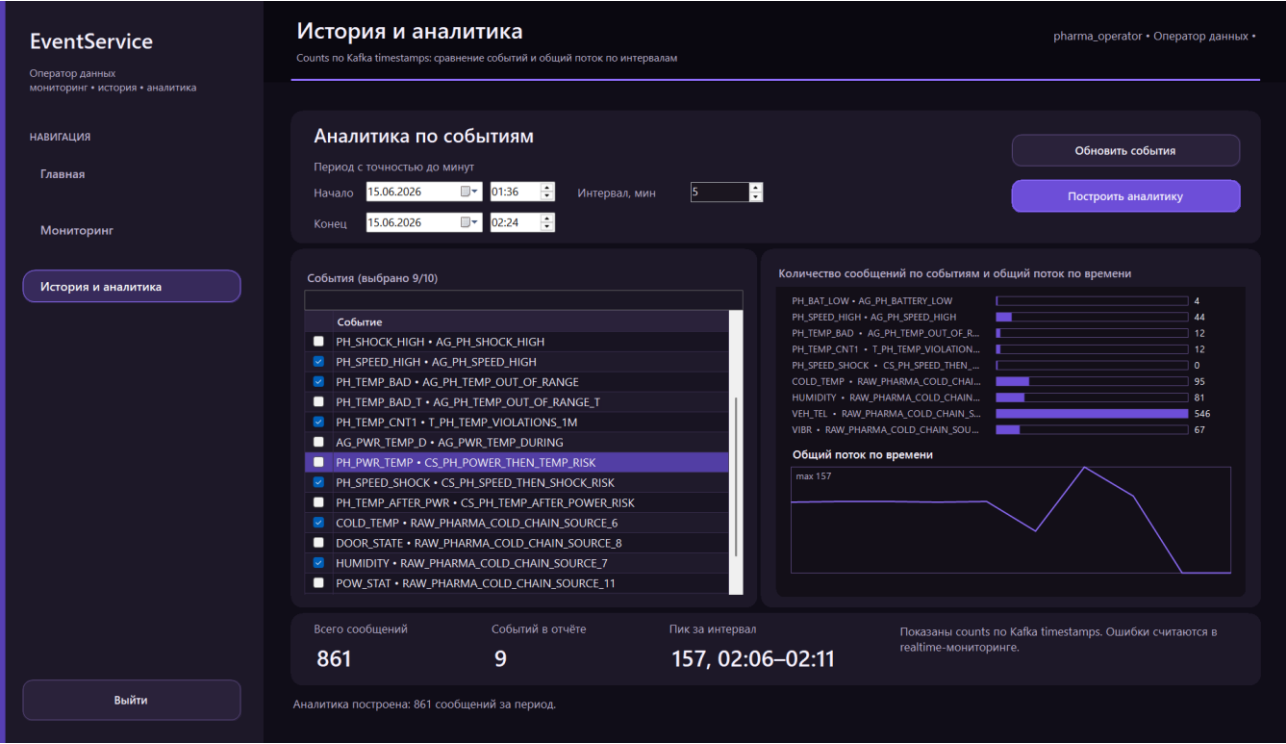


Рисунок Д.26 – Успешное построение аналитики

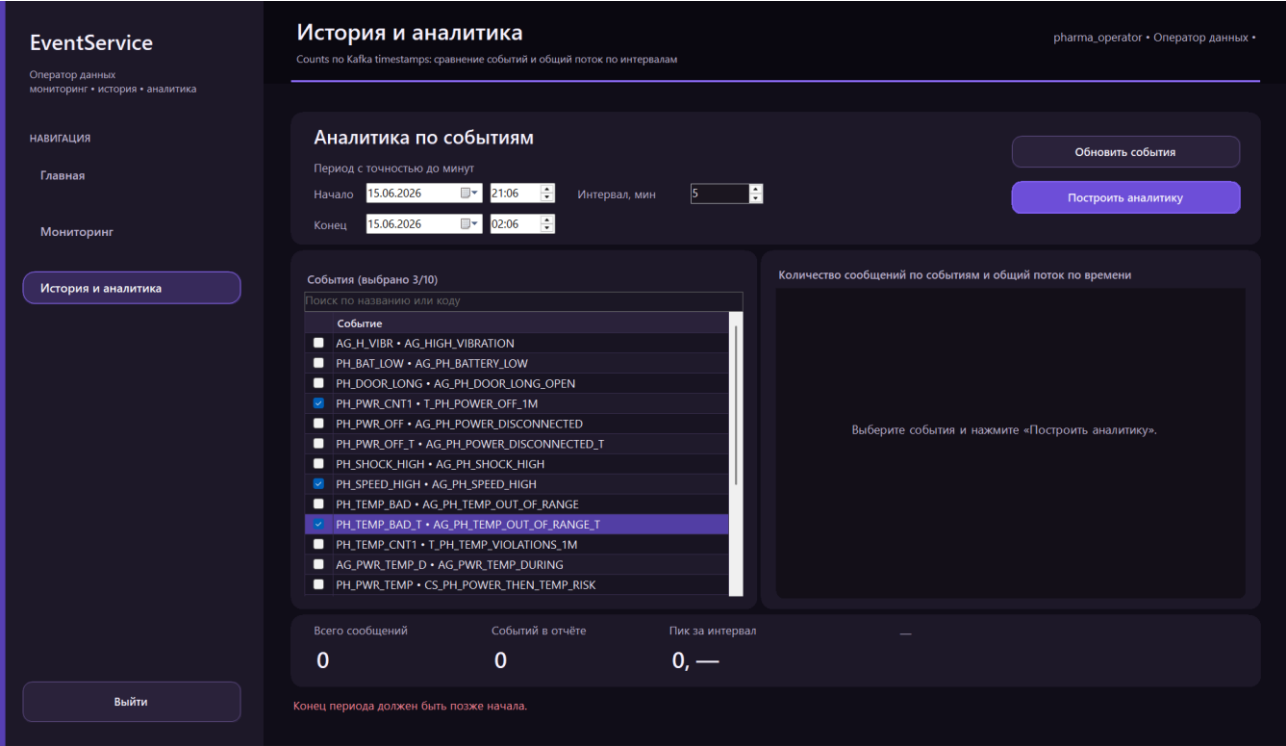


Рисунок Д.27 – Ошибка построения аналитики