

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

Факультет
вычислительной техники

Кафедра
МО и ПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

**Программные средства поддержки графowego
представления метаданных для сервиса потоковой
обработки событий**

Студент	_____	<u>Боброва Александра Васильевна</u>
	(подпись, дата)	(ФИО полностью)
Руководитель	_____	<u>Шибанов С.В.</u>
	(подпись, дата)	(фамилия, инициалы)
Нормоконтролер	_____	<u>Такташкин Д.В.</u>
	(подпись, дата)	(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

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, 2023

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
«__» _____ 2023 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Боброва Александра Васильевна гр. 19ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Программные средства поддержки графowego представления метаданных
для сервиса потоковой обработки событий

Тема утверждена приказом ПГУ № _____ от " __ " _____ 2023 г.

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

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

1) *Назначение: программные средства поддержки графowego представления метаданных
для сервиса потоковой обработки событий*

2) *Среда выполнения: Windows*

Языковые средства разработки: JavaScript

Инструментальные средства: WebStorm, Google Chrome, Docker.

Технологии разработки ООП, клиент-ориентированный подход.

3) *Основные функции разрабатываемого программного обеспечения:*

а) отображение активных правил системы в виде графowego представления;

б) создание элементов активного правила в графowego представлении;

в) управление элементами активного правила в графowego представлении;

г) анализ корректности отображаемых элементов активного правила;

д) взаимодействие с серверным приложением;

е) управление визуализацией активного правила в графowego представлении.

5. Перечень подлежащих разработке вопросов:

1) Анализ предметной области и постановка задачи;
2) План разработки бюджета;
3) Разработка требований и проектирование;
4) Реализация;
5) Контроль качества программного обеспечения;
6) Оформление пояснительной записки.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	12.04.2023 – 16.04.2023	
2	Проектирование программных средств	20%	17.04.2023 – 26.04.2023	
3	Реализация программных средств	30%	27.04.2023 – 11.05.2023	
4	Тестирование программных средств	20%	12.05.2023 – 21.05.2023	
5	Оформление пояснительной записки	20%	22.05.2023 – 01.06.2023	

Дата выдачи задания «22» декабря 2023г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

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

Реферат

Пояснительная записка содержит 148 листов, 51 рисунок, 10 таблиц, 15 использованных источников и 7 приложений.

ПРОГРАММНЫЕ СРЕДСТВА ПОДДЕРЖКИ ГРАФОВОГО ПРЕДСТАВЛЕНИЯ МЕТАДААННЫХ ДЛЯ СЕРВИСА ПОТОКОВОЙ ОБРАБОТКИ СОБЫТИЙ, ВИЗУАЛИЗАЦИЯ АКТИВНЫХ ПРАВИЛ, СОБЫТИЯ, ДЕЙСТВИЯ, УСЛОВИЯ, WEB-ПРИЛОЖЕНИЕ, JAVASCRIPT, TYPESCRIPT, HTML, REACT JS, VIS.JS, UML, WEBDRIVER IO.

Объект исследования – сервис потоковой обработки событий и исполнения активных правил.

Целью выпускной квалификационной работы является разработка программных средств поддержки графового представления метаданных для сервиса потоковой обработки событий.

Технологии разработки – программные средства написаны с использованием языков программирования JavaScript и TypeScript, фреймворка React JS и библиотеки Vis.js в среде разработки Webstorm.

Среда исполнения – Windows 10, web-браузер.

Результаты работы – по итогу выполнения выпускной квалификационной работы разработаны программные средства поддержки графового представления метаданных для сервиса потоковой обработки событий.

Все требования к разработке приложения соблюдены.

					ПГУ 09.03.04 – 08ВР191.05 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Боброва АВ.			Программные средства поддержки графового представления метаданных для сервиса потоковой обработки событий	Лит.	Лист	Листов
Пров.		Шибанов С.В.					3	148
Н. Контр.		Такташкин Д.В				Группа 19ВП1		
Утв.		Козлов А.Ю.						

Содержание

Введение.....	7
1 Анализ предметной области и постановка задачи на разработку	9
1.1 Сервис потоковой обработки событий и исполнения активных правил	9
1.1.1 Назначение и функциональные возможности сервиса потоковой обработки событий и исполнения активных правил.....	9
1.1.2 Архитектура сервиса потоковой обработки событий и исполнения активных правил.....	14
1.1.3 Схема работы сервиса потоковой обработки событий и исполнения активных правил.....	16
1.2 Метаданные сервиса потоковой обработки событий и исполнения активных правил.....	18
1.2.1 Структура метаданных сервиса	18
1.2.2 Жизненный цикл метаданных сервиса	19
1.2.3 Задачи управления метаданными сервиса.....	21
1.3 Постановка задачи на разработку программных средств.....	24
1.3.1 Цели разработки и назначение программных средств.....	24
1.3.2 Классы пользователей программных средств.....	25
1.3.3 Перечень автоматизируемых функций.....	27
1.4 Планирование и функционально-стоимостной анализ разработки программных средств	29
2 Анализ требований на разработку программных средств.....	37
2.1 Функциональные требования к программным средствам.....	37
2.2 Требования к архитектуре программных средств	39
2.3 Требования к пользовательскому интерфейсу программных средств	41
2.4 Обоснование выбора технологий и средств разработки.....	43
2.5 Требования к аппаратно-программному обеспечению.....	45
2.5.1 Требования к аппаратно-техническому обеспечению	45
2.5.2 Требования к системному программному обеспечению	46
2.5.3 Требования к прикладному программному обеспечению.....	46

3	Разработка программных средств	47
3.1	Разработка архитектуры развертывания программных средств поддержки графового представления метаданных для сервиса потоковой обработки событий.....	47
3.2	Разработка визуальных примитивов и общих средств визуализации для графового представления метаданных сервиса потоковой обработки событий	50
3.3	Разработка программных средств редактирования графового представления метаданных сервиса потоковой обработки событий.....	53
3.4	Разработка пользовательского интерфейса.....	57
4	Контроль качества программных средств.....	61
4.1	Планирование процесса контроля качества программных средств	61
4.1.1	Мероприятия по контролю качества программных средств	61
4.1.2	Обоснование выбора модели тестирования	62
4.1.3	Обоснование выбора метрик программного кода	64
4.2	Тестирование программных средств	65
4.2.1	Функциональное тестирование	65
4.2.2	Конфигурационное тестирование	66
4.2.3	Модульное тестирование	68
4.3	Оценка качества программного кода	70
4.3.1	Метрики устойчивости и абстрактности	70
4.3.2	Метрика Холстеда.....	74
4.3.3	Оценка качества пользовательского интерфейса	79
	Заключение	81
	Список используемых источников.....	83
	Приложение А. Глоссарий	85
	Приложение Б. Листинг моделей активных правил.....	87
	Приложение В. Пример системы для тестирования приложения.....	120
	Приложение Г. Результаты функционального тестирования.....	124
	Приложение Д. Результаты конфигурационного тестирования	129

Приложение Е. Результаты модульного тестирования	138
Приложение Ж. Листинг модульных тестов	141

Введение

В условиях непрерывного роста объемов собираемых и обрабатываемых данных, требуется развитие соответствующей инфраструктуры для эффективного управления этими данными. В настоящее время практически все данные генерируются как непрерывные потоки событий, будь то взаимодействие с пользователями на веб-сайтах или в мобильных приложениях, размещение заказов или получение измерений от датчиков. Событийно-ориентированные приложения, также известные как event-driven application, представляют собой потоковые приложения, которые принимают потоки событий и обрабатывают их с использованием бизнес-логики приложения. Эти приложения способны реагировать на события, инициировать отправку предупреждений или электронных писем ответственным лицам, а также передавать события другим приложениям [1]. Активное поведение позволяет оперативно реагировать на возникновение событий и обрабатывать их в процессе работы.

Сервис потоковой обработки событий является результативным средством для обработки, анализа и представления данных в режиме реального времени. Тем не менее, учитывая развитие сервиса, возникает потребность в клиентском приложении для эффективного управления метаданными и их визуализации. Кроме того, визуализация метаданных играет важную роль в облегчении понимания структуры данных и их анализа. Четкое представление метаданных в графическом виде способствует улучшению взаимодействия с данными, позволяет легко обнаруживать связи и зависимости между различными элементами данных и повышает удобство работы с сервисом потоковой обработки событий.

Целью выпускной квалификационной работы является создание такой части web-приложения управления метаданными сервиса потоковой обработки событий, которая отвечала бы за поддержку графового представления метаданных. Процесс разработки данного функционала состоит из нескольких этапов, включающих в себя обзор и изучение сервиса потоковой обработки

событий, анализ требований к разработке ПС графового представления, проектирование архитектуры и интерфейса, реализацию, а также тестирование программных средств.

1 Анализ предметной области и постановка задачи на разработку

1.1 Сервис потоковой обработки событий и исполнения активных правил

1.1.1 Назначение и функциональные возможности сервиса потоковой обработки событий и исполнения активных правил

Современные информационные системы все чаще сталкиваются с необходимостью обрабатывать и анализировать потоки событий, поступающих от наблюдаемых систем, а также реагировать на эти события в реальном времени. Для эффективной обработки и принятия решений на основе событий разрабатывается сервис потоковой обработки событий и исполнения активных правил. В данном разделе рассмотрим назначение и функциональные возможности сервиса, а также его применение в различных сферах человеческой деятельности.

Назначение этого сервиса заключается в обеспечении событийно-ориентированной обработки потоков событий, анализе данных и принятии соответствующих решений на основе заданных событийных шаблонов. Он предоставляет возможность обнаружения и анализа событий в режиме реального времени, а также применения активных правил для воздействия на управляемые системы [2].

Сервис потоковой обработки событий и исполнения активных правил находит применение в различных областях человеческой деятельности. В промышленности этот сервис используется для контроля и управления производственными процессами, позволяя оперативно реагировать на изменения и события в производственных линиях. В медицинских приложениях он применяется для мониторинга и предупреждения о состоянии пациентов, что позволяет врачам своевременно принимать необходимые меры. В системах «Умный дом» сервис обеспечивает автоматизацию и управление различными аспектами повседневной жизни, обеспечивая комфорт и безопасность для

жителей. Кроме того, он находит применение в управлении техническими объектами, в транспортных приложениях и в контроле воздушных сообщений.

Сервис потоковой обработки событий и исполнения активных правил опирается на мощную модель Event-Condition-Action (ECA), которая является фундаментом его функционирования. Эта модель позволяет системе анализировать и реагировать на события в режиме реального времени в соответствии с заранее определенными условиями и действиями. В своей сущности ECA-модель состоит из трех ключевых составляющих, каждая из которых играет важную роль в обеспечении эффективной обработки событий:

Событийная часть: Этот компонент является входной точкой системы и принимает сигналы от сторонних наблюдаемых систем. Входные события содержат не только сами данные, но и специализированный контекст, необходимый для последующей обработки. Система аккумулирует и классифицирует поступающие события, создавая поток данных для дальнейшего анализа и принятия решений.

Часть условия: Блок предоставляет механизм для оценки условий и критериев, которые заранее определены для активации соответствующих действий. Комплексные условия могут быть сформулированы с использованием логических операций, сравнений, агрегации и фильтрации данных. При соблюдении заданных условий, система переходит к следующей стадии - исполнению активных действий.

Часть действий: В этой составляющей определены разнообразные операции и действия, которые система может выполнять во взаимодействии с наблюдаемой средой. Исходя из контекста событий и соблюдения условий, сервис инициирует различные операции, изменяет состояние системы или передает команды другим компонентам для выполнения соответствующих действий. Это может включать отправку уведомлений, активацию predetermined процессов, управление устройствами или запуск автоматизированных процедур [3].

Благодаря модели ЕСА сервис потоковой обработки событий и исполнения активных правил демонстрирует высокую гибкость и адаптивность, позволяя системам в реальном времени анализировать и реагировать на события, основываясь на заранее.

В рамках сервиса потоковой обработки событий используются три группы репозиторий метаданных.

Первая группа состоит из глобальных репозиторий, которые содержат информацию о всех наблюдаемых системах, событиях, активных правилах, пользователях и конфигурационных данных компонентов сервиса. Эти репозитории обеспечивают централизованное хранение и доступ к основной информации, необходимой для работы сервиса.

Вторая группа состоит из оперативных хранилищ компонентов, которые содержат специфичную информацию для каждого отдельного компонента системы. В этих хранилищах хранятся данные, необходимые для непосредственной работы компонента, и они обеспечивают быстрый доступ к соответствующей информации.

Третья группа включает репозиторий журнализации, предназначенный для хранения информации о работе сервиса. В этот репозиторий записываются журналы, содержащие сведения о выполненных операциях, событиях и других важных аспектах функционирования сервиса. Репозиторий журнализации позволяет осуществлять отслеживание и анализ работы сервиса, а также восстановление информации в случае необходимости.

Данные репозитории обеспечивают эффективное управление и использование метаданных в рамках работы с системой, обеспечивая надежность и эффективность сервиса.

Функциональные возможности сервиса потоковой обработки событий и исполнения активных правил включают в себя [4]:

- а) конструирование и управление активными правилами:
 - 1) создание и редактирование правил и их компонентов;

- 2) проверка корректности правила, подключение и отключение правила;
- б) конструирование и управление метаданными событий:
 - 1) создание и редактирование объектов событий;
 - 2) привязка созданных типов к компонентам наблюдаемой системы;
- в) анализ активных правил:
 - 1) выявление и устранение коллизий;
 - 2) исполнение активных правил (статический анализ на этапе конструирования и динамический анализ на этапе исполнения);
- г) обработка событий:
 - 1) обработка потока событий от наблюдаемых систем;
 - 2) первичная обработка входных событий и формирование событий на основе агрегации входных данных;
 - 3) выявление сложных событий на основе заданных шаблонов;
 - 4) доставка событий к соответствующим компонентам сервиса для их обработки;
- д) исполнение активных правил:
 - 1) выбор экземпляра активного правила (группы правил) для исполнения;
 - 2) вычисление условия экземпляра активного правила;
 - 3) принятие решения о выполнении действия экземпляра активного правила;
 - 4) применение действия к исполняемому объекту;
- е) журнализация и анализ данных журналов событий:
 - 1) журнализация обрабатываемых событий, исполнения активных правил, обработки компонент активных правил;
 - 2) выявление закономерностей в журналах событий;
 - 3) моделирование новых событий и активных правил на основе

выявленных закономерностей;

4) выявление «проблемных» событий;

ж) настройка и масштабирование сервиса:

1) изменение параметров функционирования отдельных компонент с учетом результатов аудита;

2) масштабирование отдельных компонент (изменение параметров и распределение очередей и потоков обработки) и сервиса в целом (подключение/отключение компонентов);

3) синхронизация глобальных и локальных метаданных, описывающих конфигурацию системы и отдельных компонент.

На основе определенных выше функций можно составить диаграмму вариантов использования сервиса потоковой обработки событий, которая отображена на рисунке 1.

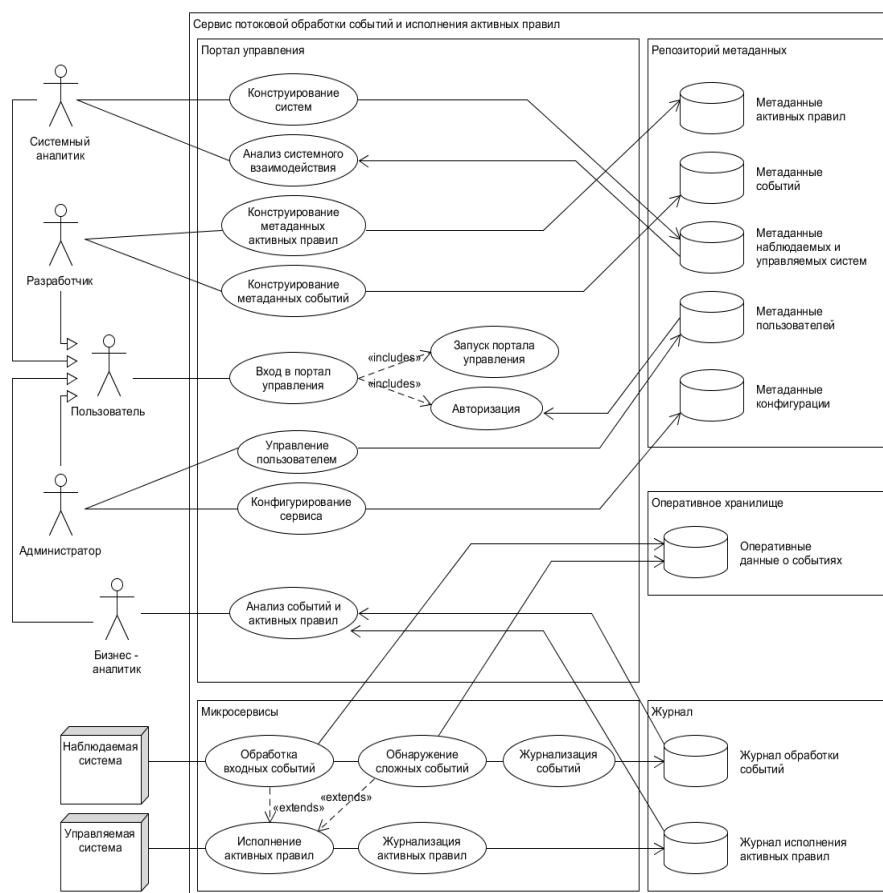


Рисунок 1 – Диаграмма вариантов использования сервиса потоковой обработки событий

Таким образом, сервис потоковой обработки событий и исполнения активных правил представляет собой мощный инструмент для обработки и анализа потоков событий, позволяющий оперативно реагировать на события и принимать соответствующие решения. Его применение находит широкое применение в различных областях, где требуется обработка данных в реальном времени и эффективное управление системами.

1.1.2 Архитектура сервиса потоковой обработки событий и исполнения активных правил

Принципы интеграции микросервисного и событийно-ориентированных подходов лежат в основе архитектуры сервиса потоковой обработки событий и исполнения активных правил.

Так как в ходе построения системы потоковой обработки событий наилучшим решением было разделение приложения на отдельные сервисы - модули, за основу была взята микросервисная архитектура.

Модульность системы является ее важным свойством, так как она позволяет изменять масштаб конкретного модуля, заменять его, разворачивать и обновлять, не затрагивая остальные. Существует несколько вариантов реализации модулей, в которых используются различные языки программирования, фреймворки, связующее программное обеспечения, работа с контейнерами, виртуализация и т.д.

Доставка событий, поступающих от наблюдаемых систем, реализуется на основе событийно-ориентированного подхода, который использует в своей работе механизм очередей и платформу для потоковой обработки данных Kafka, в частности, фреймворк Kafka Team.

Синхронизация метаданных, обращение к компонентам сервиса во время настройки, запуска и остановки реализуется через REST-сервер с использованием специального сценария. Этот сценарий позволяет снизить время ожидания при запросе к метаданным без ущерба для актуальности информации. Таким образом, обеспечивается эффективное управление метаданными и

оптимизация работы сервиса потоковой обработки событий и исполнения активных правил.

Архитектура сервиса потоковой обработки событий и исполнения активных правил представлена на рисунке 2.

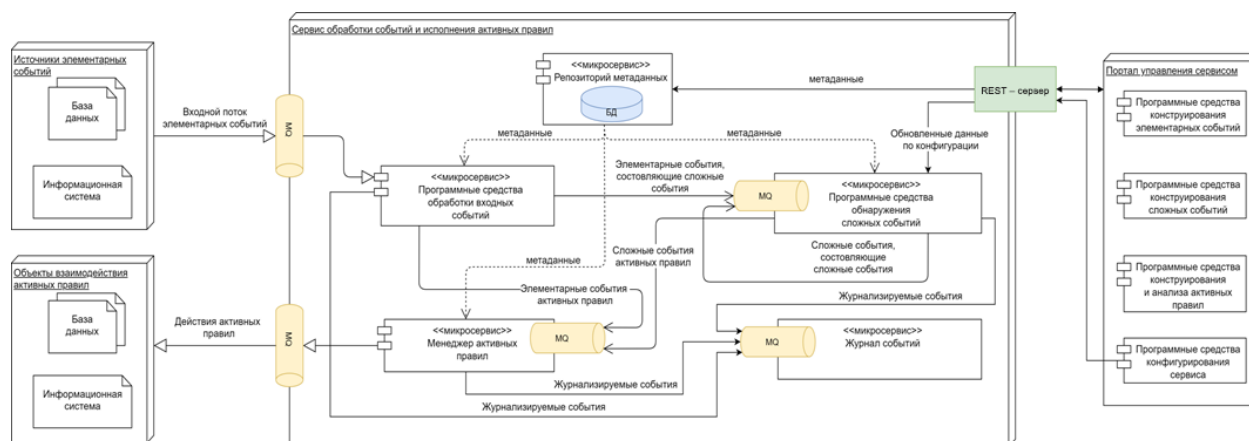


Рисунок 2 – Архитектура сервиса потоковой обработки событий

Сервис потоковой обработки событий и исполнения активных правил получает входные события от различных наблюдаемых систем, таких как базы данных, системы управления техническими объектами и другие подобные системы. После получения этих событий, сервис их обрабатывает и, в случае наличия активных правил, осуществляет соответствующие действия в отношении управляемых систем.

Для эффективного хранения и управления метаданными в процессе работы сервиса потоковой обработки событий и исполнения активных правил был разработан централизованный репозиторий метаданных. Этот репозиторий служит для хранения информации о различных аспектах функционирования сервиса [5].

Для обеспечения более эффективной обработки событий, исполнения активных правил и работы сервиса в целом, каждый компонент сервиса имеет свой локальный репозиторий метаданных в виде in-memory хранилища, управляемого Redis. Это позволяет компонентам быстро получать доступ к

соответствующей информации, не прибегая к централизованному репозиторию, а также повышает отказоустойчивость и обеспечивает автономность каждого сервиса в отдельности [6].

1.1.3 Схема работы сервиса потоковой обработки событий и исполнения активных правил

Автоматизация обработки данных играет значительную роль в контексте управления, например, процессами производства на предприятии. Алгоритмы помогают удаленно контролировать и наблюдать за всем ходом производственного процесса. Благодаря этому, возможно заранее обнаруживать потенциальные неисправности и собирать ценные статистические данные, относящиеся к работе конкретных узлов системы. Инструмент эффективно взаимодействует с жизненным циклом любой системы, обеспечивая оптимальное ее управление.

Процесс обработки событий осуществляется с использованием передовой технологии, известной как ESP (Event Stream Processing). Эта методика позволяет системе распознавать часто встречающиеся шаблоны поведения и выявлять потенциальные проблемы, включая циклическое повторение запрограммированных сценариев, коллизии и иерархию событий. Благодаря этому, возможно оперативно реагировать на сложные и проблемные случаи, связанные с функционированием системы [7].

В связи с непрерывной потоковой обработкой событий сервис обладает встроенной способностью автоматической реакции на предопределенные ситуации. Например, в случае обнаружения отказа оборудования или возникновения угрозы пожарной безопасности, система моментально реагирует и предпринимает соответствующие меры. Это значительно повышает безопасность и эффективность работы, а также снижает вероятность возникновения серьезных проблем.

На рисунке 3 представлена диаграмма деятельности сервиса потоковой обработки событий и исполнения активных правил, показывающая функционирование рассматриваемой системы [8].

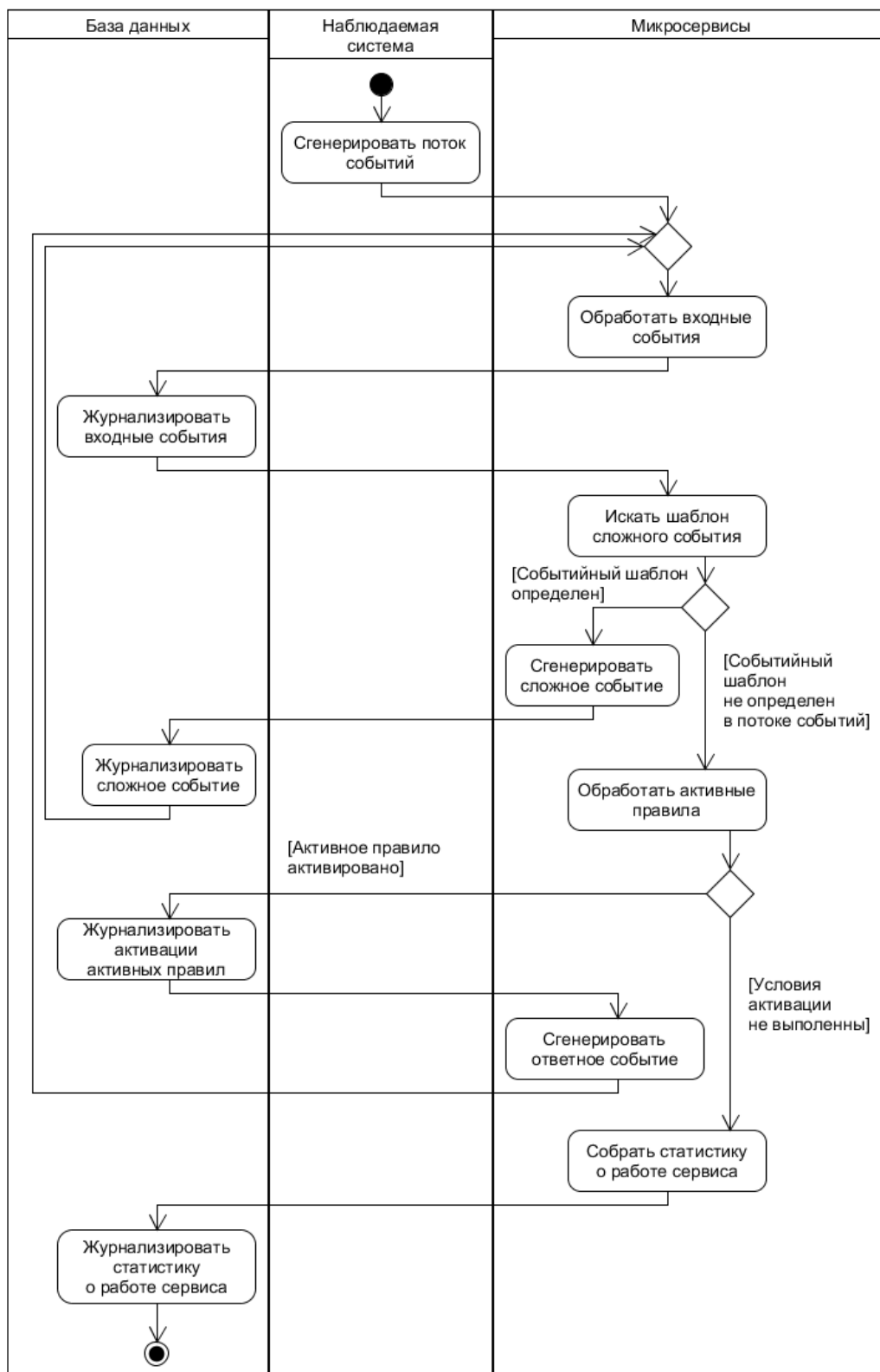


Рисунок 3 – Диаграмма деятельности сервиса потоковой обработки событий

В случае применения подобной технологии на предприятиях, открывается возможность проведения обслуживания аппаратуры в зависимости от ее текущего состояния, а не от жестких регламентированных сроков ремонтных работ. Это позволило бы существенно сократить издержки и оптимизировать затраты на обслуживание и ремонт систем.

1.2 Метаданные сервиса потоковой обработки событий и исполнения активных правил

1.2.1 Структура метаданных сервиса

Без метаданных функционирование сервиса потоковой обработки событий и выполнения активных правил было бы невозможно. Они описывают различные типы объектов, которые хранятся и обрабатываются в системе. Можно выделить несколько типов метаданных:

Метаданные событий. Они представляют собой описание разных видов событий, которые могут происходить в системе. Эти метаданные могут приходить извне от отслеживаемых устройств или генерироваться внутри пользовательского окружения. В зависимости от своего назначения, метаданные событий подразделяются на [9]:

- а) метаданные элементарных событий;
- б) метаданные агрегирующих событий;
- в) метаданные сложных событий.

Метаданные элементарных событий представляют базовую информацию об атомарных событиях, которые поступают от определенного устройства и содержат измерение одного конкретного параметра. Они являются основой для анализа и обработки данных.

Метаданные агрегирующих событий содержат информацию об агрегирующем событии, которое объединяет несколько элементарных событий и предоставляет обобщенные данные. Эти метаданные используются для обнаружения, вычисления или выделения полезной информации из последовательности событий.

Метаданные сложных событий отображают информацию о сложных событиях, которые генерируются в системе при обнаружении определенного событийного шаблона. Шаблон состоит из последовательности событий указанных типов, которые происходят с определенными интервалами времени.

Метаданные компонентов описывают интегрированные в пользовательское окружение устройства, измерительные датчики и другие приборы, которые отправляют потоки событий в систему. Они предоставляют информацию о компонентах системы и их свойствах.

Метаданные систем содержат информацию о системах, в которых располагаются отслеживаемые устройства. Это может быть отдельное помещение, здание или подсистема компонентов в большой группе устройств.

Для создания большинства типов метаданных пользователи с соответствующими ролями в системе могут использовать специальные инструментальные средства и сохранять их в централизованном репозитории метаданных. Такая структура позволяет эффективно управлять и использовать информацию о различных объектах и событиях в системе потоковой обработки событий.

В базах данных сервиса потоковой обработки событий также хранятся данные о пользователях, работающих с окружением, а также вспомогательные метаданные, включающие параметры событий, контекст, уровни и роли компонентов, и другую информацию.

1.2.2 Жизненный цикл метаданных сервиса

Процесс эволюции метаданных в сервисе потоковой обработки событий начинается с момента, когда принимается решение о интеграции внешних систем мониторинга. Целью этой интеграции является обработка поступающих от этих систем событий, а также управление ими с использованием активных правил.

ЖЦ метаданных включает в себя несколько этапов, показанных на рисунке 4.



Рисунок 4 – Жизненный цикл метаданных сервиса обработки событий

Процесс формирования метаданных включает несколько этапов. На первом этапе происходит создание или модификация объектов метаданных. Затем проводится автономная проверка этих объектов, включающая проверку ограничений, не зависящих от других объектов метаданных. На последнем этапе происходит сохранение объектов метаданных в центральном репозитории.

Комплексный анализ метаданных также включает несколько этапов. На первом этапе происходит проверка согласованности с другими объектами метаданных и выявление коллизий в метаданных. Также проводится анализ коллизий взаимодействия активных правил в процессе их исполнения.

Преобразование метаданных проходит через несколько операций. Оно включает устранение коллизий в метаданных, планирование мероприятий для контроля возможных коллизий на этапе динамического анализа в процессе исполнения активных правил, а также преобразование метаданных в удобный для практического применения вид.

Актуализация метаданных осуществляется путем включения объектов метаданных в использование и доставки их в локальные репозитории компонентов сервиса.

Деактуализация метаданных, в свою очередь, включает отключение объектов метаданных от использования, их удаление из локальных репозиториях компонентов сервиса, а также полное удаление объектов метаданных из центрального репозитория.

Важно отметить, что на каждом из этих этапов учитывается семантическая связанность объектов метаданных.

1.2.3 Задачи управления метаданными сервиса

Управление метаданными в сервисе является важной составляющей эффективной работы системы. Метаданные содержат информацию о структуре, свойствах и взаимосвязях данных, необходимых для правильного функционирования сервиса. Задачи управления метаданными направлены на обеспечение их целостности, актуальности и соответствия требованиям системы и пользователей.

Одной из задач управления метаданными является формирование метаданных. Этот процесс включает создание и модификацию объектов метаданных, которые описывают структуру и свойства данных. Важным аспектом этой задачи является автономная проверка объектов метаданных, чтобы удостовериться в их корректности и соответствии заданным ограничениям. Затем объекты метаданных сохраняются в центральном репозитории для дальнейшего использования.

Комплексный анализ метаданных также является важной задачей управления. В процессе анализа проводится проверка согласованности между объектами метаданных, чтобы исключить возможные коллизии и противоречия. Это включает анализ взаимодействия активных правил и выявление потенциальных проблем. Результаты анализа метаданных позволяют определить оптимальные наборы метаданных, необходимых для эффективной работы сервиса.

Преобразование метаданных также стоит в списке задач управления. Оно включает устранение коллизий в метаданных, которые могут возникнуть в процессе работы системы. Планирование мероприятий для детального контроля коллизий на этапе динамического анализа активных правил является важным аспектом этой задачи. Кроме того, преобразование метаданных к удобному для практического применения виду обеспечивает оптимальное использование данных сервисом.

Актуализация метаданных является еще одной задачей управления. Она включает включение в использование объектов метаданных и их доставку в локальные репозитории компонентов сервиса. Это позволяет обеспечить доступность и актуальность метаданных для всех необходимых процессов и операций.

Наконец, задача деактуализации метаданных заключается в отключении от использования объектов метаданных и их удалении из локальных репозиториях компонентов сервиса. Полное удаление объектов метаданных из центрального репозитория также может быть необходимым в определенных случаях.

Все эти задачи управления метаданными направлены на обеспечение целостности, актуальности и эффективного использования данных в сервисе.

На рисунке 5 показан процесс управления метаданными в виде диаграммы деятельности.

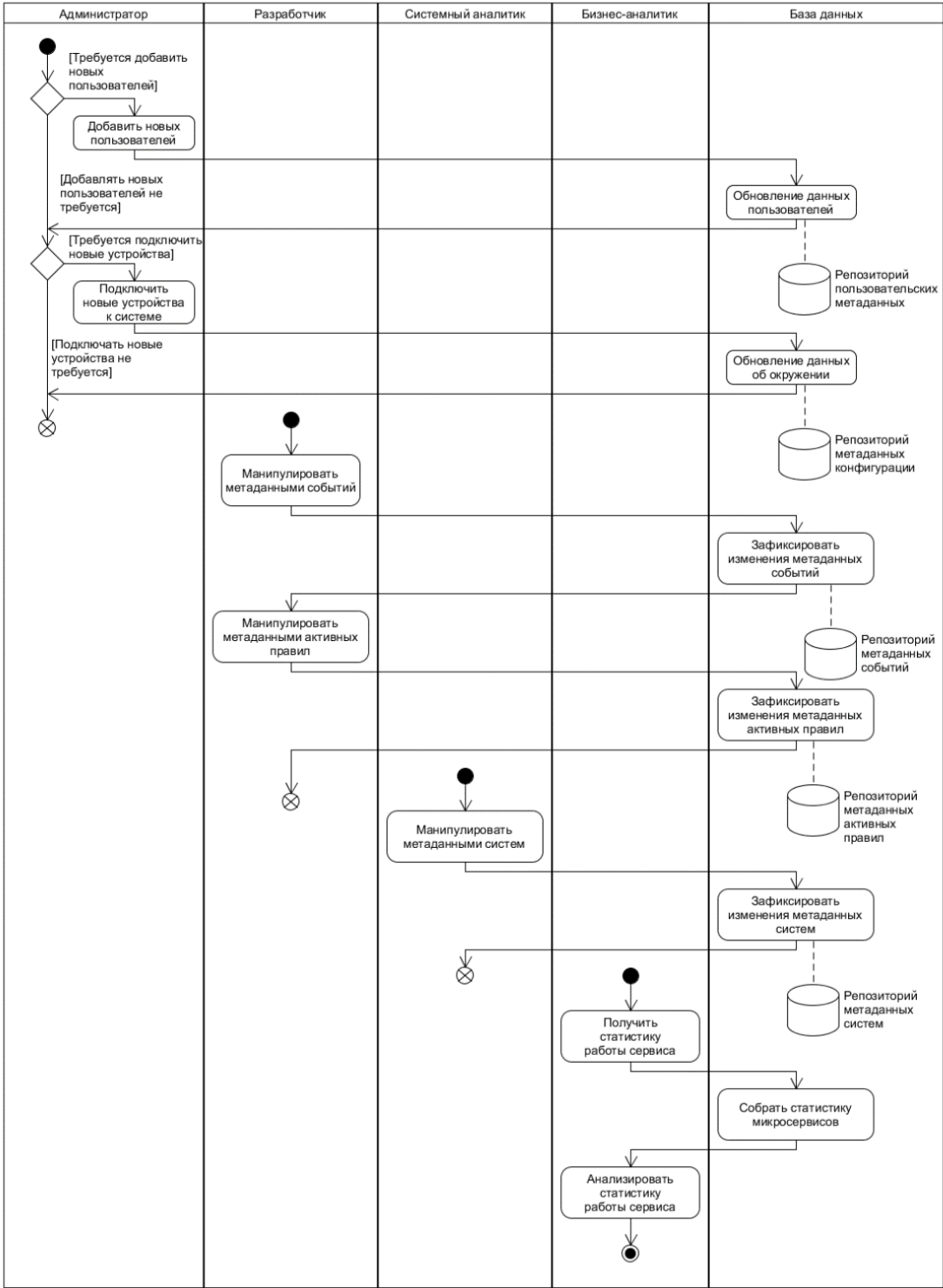


Рисунок 5 – Процесс управления метаданными в виде диаграммы деятельности

Выполнение задач, необходимых для реализации представленного на диаграмме деятельности процесса требует использования соответствующих инструментов и подходов, которые будут рассмотрены в последующих разделах.

1.3 Постановка задачи на разработку программных средств

1.3.1 Цели разработки и назначение программных средств

Цели разработки программных средств сервиса потоковой обработки событий направлены на обеспечение эффективного и надежного управления потоком событий, а также на достижение оптимальной обработки данных и исполнения активных правил. Назначение этих программных средств состоит в обеспечении возможности обработки и анализа событий в реальном времени с целью выявления интересующих паттернов, принятия решений и выполнения соответствующих действий.

Разработка программных средств сервиса потоковой обработки событий включает в себя такие цели:

- а) обработка и анализ событий;
- б) исполнение активных правил;
- в) масштабируемость и отказоустойчивость.

Разработка программных средств направлена на эффективное принятие, обработку и анализ потока событий. Целью является выявление важных событий, определение паттернов и трендов, а также принятие решений на основе полученных данных.

Программные средства должны обеспечивать исполнение активных правил в режиме реального времени. Это включает применение бизнес-логики и выявление ситуаций, требующих особых действий или уведомлений.

Разработка программных средств должна учитывать возможность масштабирования системы для обработки больших объемов событий и обеспечения отказоустойчивости. Целью является обеспечение надежности и доступности сервиса в условиях высокой нагрузки и возможных сбоев.

Главной целью разработки программных средств управления метаданными является создание и предоставление пользователям удобного инструмента для манипулирования метаданными системы, а также комплекса

программных средств с обширными возможностями мониторинга, контроля и управления созданным окружением.

1.3.2 Классы пользователей программных средств

Классы пользователей программных средств сервиса потоковой обработки событий включают различные роли, каждая из которых имеет свои особенности и доступ к определенным функциональностям.

На рисунке 6 отображены основные процессы управления и роли доступа пользователей к данным процессам.

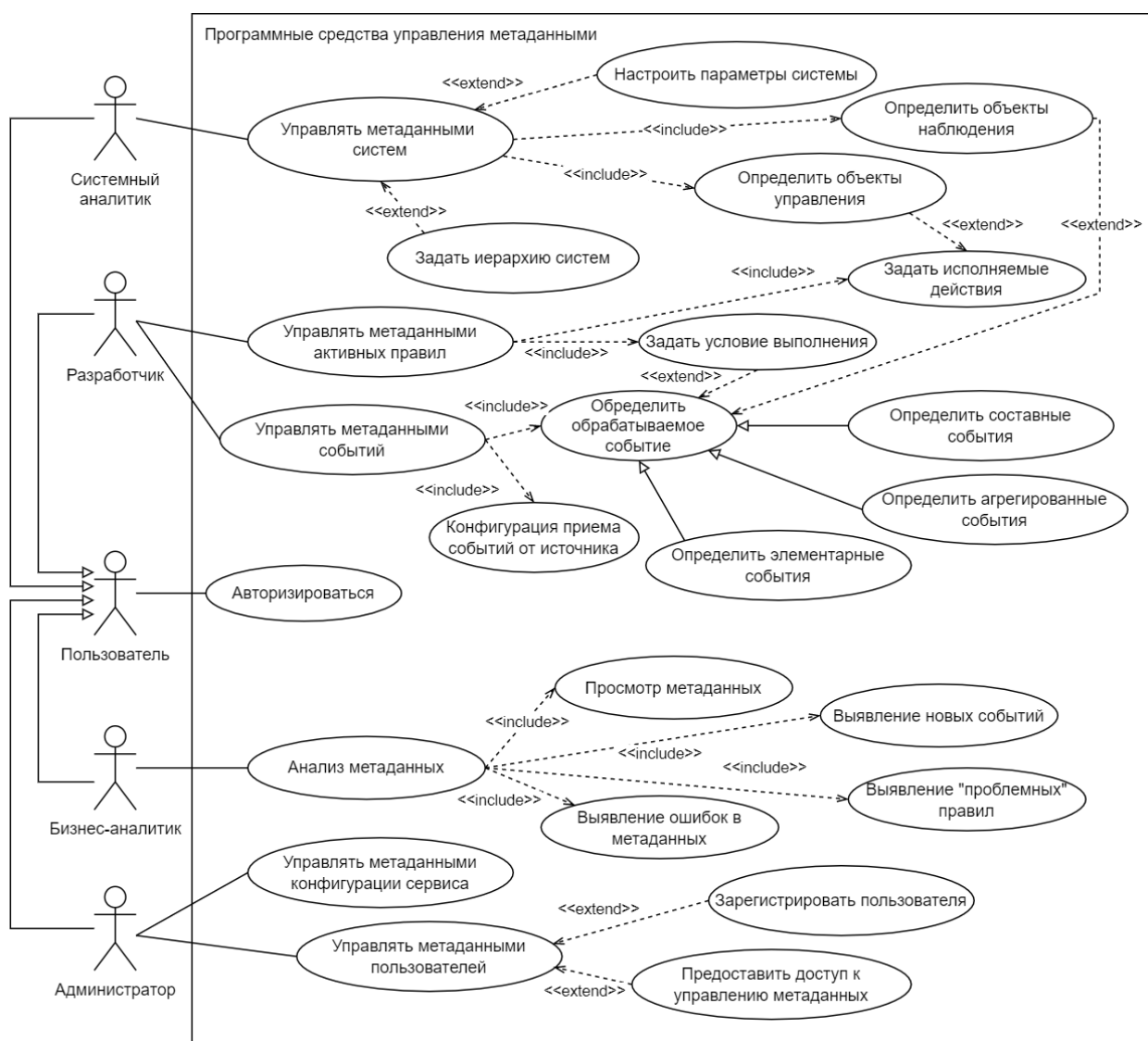


Рисунок 6 – Основные процессы управления и роли доступа пользователей к данным процессам

Подробнее рассмотрим некоторые из отображенных на диаграмме классы пользователей:

- а) администраторы системы;
- б) разработчики;
- в) бизнес-аналитики;
- г) системный аналитик;
- д) пользователи.

Администраторы являются высшими уровнями пользователей и обладают полным контролем над сервисом потоковой обработки событий. Они отвечают за установку, настройку и обновление системы, а также за управление пользователями, ролями и доступом. Администраторы также могут настраивать правила обработки событий и определять параметры системы для обеспечения ее эффективной работы.

Разработчики занимаются конструированием метаданных активных правил и метаданных событий, а также управлением связанных с ними метаданными.

Бизнес-аналитики используют программные средства сервиса потоковой обработки событий для анализа и извлечения значимой информации из потока событий. Они могут проводить аудит обрабатываемых событий и исполнения активных правил. Аналитики данных используют результаты анализа для выявления ошибок, новых событий и «проблемных» правил.

Роль системного аналитика позволяет конструировать метаданные систем, заниматься анализом взаимодействия систем и их компонентов, а также настраивать наблюдаемые системы посредством изменения их параметров и определения объектов наблюдения и управления.

Пользователи могут авторизоваться в системе и выполнять некоторые операции с метаданными.

Каждый класс пользователей имеет свои задачи, роли и ограничения, которые определяются настройками и правами доступа в программных

средствах сервиса потоковой обработки событий. Это позволяет разграничить функциональность и обеспечить безопасность и эффективность работы системы в зависимости от потребностей разных категорий пользователей.

Разрабатываемые в данной выпускной квалификационной работе программные средства графового представления предназначены главным образом для работы с ними разработчика.

1.3.3 Перечень автоматизируемых функций

Перечень автоматизируемых функций сервиса потоковой обработки событий должен включать ряд важных возможностей, которые способствуют эффективной работе и повышают функциональность системы. Далее подробнее рассмотрим каждую из них.

Автоматическая генерация запросов к репозиторию метаданных в зависимости от требований сервиса позволяет системе автоматически формировать и выполнять запросы к репозиторию метаданных на основе заданных параметров. Такой подход значительно упрощает процесс извлечения необходимых данных и позволяет получать результаты более эффективно.

Следующей возможностью является контроль безопасности и предоставление доступа пользователей к механизмам управления метаданными. Безопасность является одним из важных аспектов любой системы. Эта функция обеспечивает управление доступом пользователей к механизмам управления метаданными и гарантирует, что только авторизованные пользователи имеют доступ к соответствующим функциям.

Сервис потоковой обработки событий может содержать набор активных правил, которые могут выполняться в реальном времени для обработки событий. Автоматическое оповещение бизнес-приложений об исполнении активных правил оповещает связанные бизнес-приложения о срабатывании этих правил, что позволяет им реагировать на события и выполнять соответствующие действия.

Автоматическое оповещение микросервисов об изменении, удалении или создании метаданных позволяет автоматически оповещать связанные микросервисы о любых изменениях в метаданных, таких как изменение схемы данных или добавление новых объектов. Такое оповещение обеспечивает синхронизацию между различными компонентами системы и позволяет им динамически адаптироваться к изменениям.

Часто системы обработки событий имеют распределенную архитектуру, состоящую из различных узлов и компонентов. Контроль актуальности метаданных в локальных репозиториях каждого элемента архитектуры контролирует актуальность метаданных в локальных репозиториях каждого элемента системы, обеспечивая их согласованность и предотвращая возможные проблемы синхронизации.

Важно иметь возможность отслеживать и анализировать операции, связанные с управлением метаданными. Журнализация операций управления метаданными обеспечивает журналирование операций, выполняемых над метаданными, что позволяет отслеживать и проверять изменения, производимые в системе.

Системы потоковой обработки событий могут столкнуться с различными проблемами и ошибками в процессе работы. Журналирование ошибок является важным компонентом, позволяющим отслеживать и анализировать возникающие проблемы. Такая информация помогает в идентификации и устранении ошибок, повышая надежность и стабильность работы системы.

Кроме перечисленных функций, программные средства также должны обладать минимальным набором защиты от атак злонамеренных сторонних сервисов и программ, а также предотвращать несанкционированный доступ. Это включает механизмы аутентификации, авторизации и контроля целостности данных, которые помогают защитить систему от потенциальных угроз.

Автоматизированные функции значительно улучшают процессы управления метаданными в сервисе потоковой обработки событий. Они

обеспечивают безопасность, надежность и гибкость системы, позволяя пользователям эффективно работать с данными и максимально использовать потенциал сервиса.

1.4 Планирование и функционально-стоимостной анализ разработки программных средств

Планирование и функционально-стоимостной анализ являются важными аспектами в разработке программных средств. Эти два процесса взаимосвязаны и помогают обеспечить успешную реализацию проекта с оптимальным использованием ресурсов.

Планирование включает в себя определение целей проекта, составление плана работ, а также оценку требований и рисков. Хорошо спланированный проект обеспечивает оптимальное использование ресурсов, а также позволяет управлять сроками и качеством разработки.

Функционально-стоимостной анализ является инструментом для определения стоимости и эффективности разработки программных средств. Он помогает оценить затраты на различные аспекты проекта, включая трудозатраты, материальные ресурсы, инфраструктуру и другие факторы. Такой анализ позволяет принимать обоснованные решения и оптимизировать бюджет проекта.

Планирование и функционально-стоимостной анализ взаимосвязаны. На этапе планирования учитываются результаты функционально-стоимостного анализа для определения оптимальных решений и распределения ресурсов. В свою очередь, результаты планирования могут влиять на функционально-стоимостной анализ, помогая уточнить стоимость и эффективность разработки.

Правильное планирование и проведение функционально-стоимостного анализа способствуют достижению поставленных целей проекта, минимизации затрат и повышению эффективности разработки программных средств. Эти процессы являются неотъемлемой частью жизненного цикла проекта и требуют внимания и компетентности со стороны команды разработчиков и управленцев.

В таблице 1 представлено подробное описание ключевых задач проекта, которые требуется решить для успешной разработки серверных программных средств, а также определены их последовательность и оценочная длительность. При планировании было принято решение организовать работу в шестидневном режиме.

Таблица 1 – Список задач проекта

№	Название задачи	Длительность	Предшественники
1	Начало проекта	-	-
2	Анализ предметной области и постановка задачи на разработку	-	-
3	Анализ актуальности разработки	2 дня	1
4	Анализ принципов работы сервиса потоковой обработки событий	2 дня	3
5	Анализ видов метаданных и их жизненного цикла	1 день	4
6	Анализ схем управления метаданными	1 день	5
7	Изучение и сопоставление аналогичных программных средств	3 дня	6
8	Определение списка автоматизируемого функционала	2 дня	7
9	Оценка функционально-стоимостных затрат проекта	1 день	8
10	Анализ предметной области и постановка задачи на разработку завершены	4 дня	9
11	Определение требований на разработку	-	2
12	Определение функциональных требований	3 дня	10
13	Определение требований к архитектуре	2 дня	12
14	Определение требований к пользовательскому интерфейсу	2 дня	13
15	Выбор технологий и средств разработки	2 дня	14
16	Определение требований к аппаратно-программному обеспечению	2 дня	15
17	Определение требований на разработку завершено	-	16
18	Разработка программных средств	-	10

Продолжение таблицы 1

19	Разработка программных средств взаимодействия с серверной частью сервиса	3 дня	17
20	Разработка пользовательского интерфейса	2 дня	19
21	Разработка программных средств графового представления	4 дней	20
22	Разработка программных средств завершена	-	21
23	Тестирование программных средств	-	18
24	Анализ и выбор методов тестирования	2 дня	22
25	Проведение тестирования программных средств	3 дня	24
26	Анализ качества программного кода	2 дня	25
27	Тестирование программных средств завершено	-	26
28	Сдача проекта	-	23
29	Составление документации	5 дней	27
30	Ввод в эксплуатацию	1 день	29
31	Сдача проекта завершена	-	30
32	Конец проекта	-	31

Результаты разбиения проекта на фазы, вехи и задачи показали, что для реализации потребуется выполнить 32 задачи. В таблице 2 детализированы затрачиваемые ресурсы.

Таблица 2 – Описание затрачиваемых ресурсов

Название ресурса	Тип	Стандартная ставка	Ставка сверхурочных
Аналитик	Трудовой	1000 Р/ч	1100 Р/ч
Программист	Трудовой	1200 Р/ч	1300 Р/ч
Дизайнер	Трудовой	900 Р/ч	1000 Р/ч
Тестировщик	Трудовой	1000 Р/ч	1100 Р/ч
Компьютер	Материальный	60 000 Р	-
Электричество	Затраты		-
Интернет	Затраты		-

Распределение затрачиваемых ресурсов по запланированным задачам представлено в таблице 3.

Таблица 3 – Распределение ресурсов по задачам

№	Название задачи	Ресурс	Единицы
1	Начало проекта	-	-
2	Анализ предметной области и постановка задачи на разработку	-	-
3	Анализ актуальности разработки	Аналитик	100%
4	Анализ принципов работы сервиса потоковой обработки событий	Аналитик	100%
5	Анализ видов метаданных и их жизненного цикла	Аналитик	100%
6	Анализ схем управления метаданными	Аналитик	100%
7	Изучение и сопоставление аналогичных программных средств	Аналитик Электричество Интернет	100% 20 р. 100 р.
8	Определение списка автоматизируемого функционала	Аналитик	100%
9	Оценка функционально-стоимостных затрат проекта	Аналитик Электричество	100% 10 р.
10	Анализ предметной области и постановка задачи на разработку завершены	-	-
11	Определение требований на разработку	-	-
12	Определение функциональных требований	Программист	100%
13	Определение требований к архитектуре	Программист	100%
14	Определение требований к пользовательскому интерфейсу	Программист	100%
15	Выбор технологий и средств разработки	Программист	100%
16	Определение требований к аппаратно-программному обеспечению	Программист	100%
17	Определение требований на разработку завершено	-	-
18	Разработка программных средств	-	-

Продолжение таблицы 3

19	Разработка программных средств взаимодействия с серверной частью сервиса	Программист Электричество	100% 20 р.
20	Разработка пользовательского интерфейса	Программист Дизайнер Электричество Интернет	100% 100% 20 Р 100 Р
21	Разработка программных средств графового представления	Программист Электричество Интернет	100% 20 р. 100 р.
22	Разработка программных средств завершена	-	-
23	Тестирование программных средств	-	-
24	Анализ и выбор методов тестирования	Тестирующий	100%
25	Проведение тестирования программных средств	Тестирующий Электричество Интернет	100% 20 Р 100 Р
26	Анализ качества программного кода	Тестирующий Электричество	100% 20 Р
27	Тестирование программных средств завершено	-	-
28	Сдача проекта	-	-
29	Составление документации	Аналитик Программист Электричество Интернет	100% 100 % 20 р. 100 р.
30	Ввод в эксплуатацию	Программист Электричество Интернет	100% 20 р. 100 р.
31	Сдача проекта завершена	-	-
32	Конец проекта	-	-

Планирование и анализ временных и финансовых затрат разработки было осуществлено в ПО Microsoft Project (версия Microsoft Project 2007) [10]. Также создана диаграмма Ганта, которая представлена на рисунке 7.

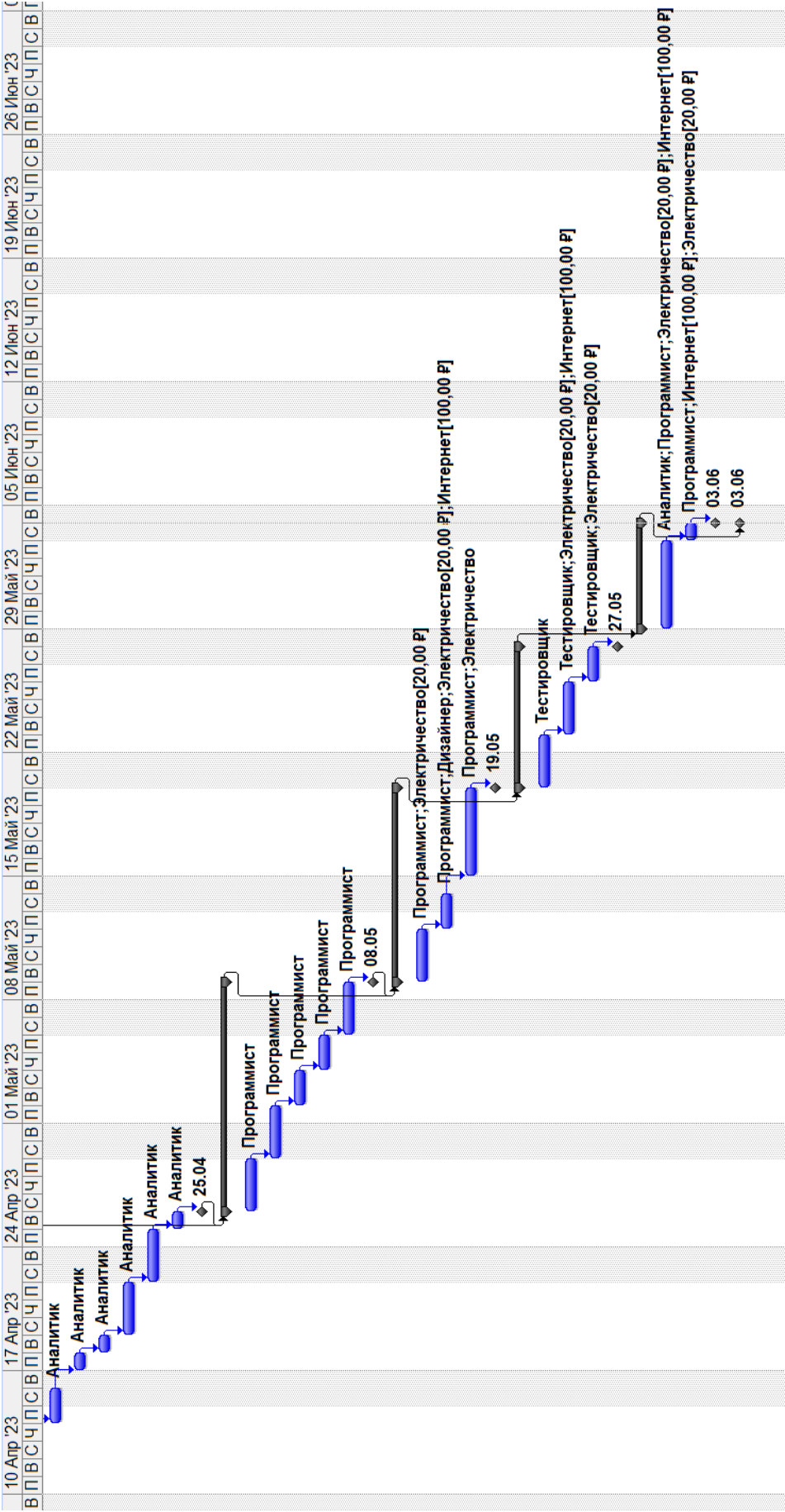


Рисунок 7 – Диаграмма Ганта задач с распределением ресурсов

На диаграмме отражены ресурсы, основные этапы разработки, последовательность и сроки выполнения каждой задачи. Выполнение задач имеет последовательный характер, так как доступные трудовые ресурсы ограничены, и каждый этап зависит от успешного завершения предыдущего. Это позволяет эффективно организовать рабочий процесс и обеспечить прогрессивное развитие проекта.

Далее был сформирован отчет о бюджете (рисунок 8), демонстрирующий затраты на реализацию проекта.

После проведения анализа и подготовки отчета о бюджете было установлено, что общая стоимость разработки программных средств составляет 466 370 рублей.

Отчет о бюджете от Вс 04.06.23
ПроектВКР

Ид.	Название задачи	Фиксированные затраты	Начисление фикс. затрат	Общие затраты	Базовые	Отклонение
29	Составление документации	0,00 Р	Пропорциональное	88 120,00 Р	0,00 Р	88 120,00 Р
21	Разработка программных средств	0,00 Р	Пропорциональное	48 120,00 Р	0,00 Р	48 120,00 Р
20	Разработка пользовательского инт	0,00 Р	Пропорциональное	33 720,00 Р	0,00 Р	33 720,00 Р
19	Разработка программных средств в	0,00 Р	Пропорциональное	28 820,00 Р	0,00 Р	28 820,00 Р
12	Определение функциональных тре	0,00 Р	Пропорциональное	28 800,00 Р	0,00 Р	28 800,00 Р
7	Изучение и сопоставление аналогич	0,00 Р	Пропорциональное	24 120,00 Р	0,00 Р	24 120,00 Р
25	Проведение тестирования програм	0,00 Р	Пропорциональное	24 120,00 Р	0,00 Р	24 120,00 Р
13	Определение требований к архитек	0,00 Р	Пропорциональное	19 200,00 Р	0,00 Р	19 200,00 Р
14	Определение требований к пользо	0,00 Р	Пропорциональное	19 200,00 Р	0,00 Р	19 200,00 Р
15	Выбор технологий и средств разраб	0,00 Р	Пропорциональное	19 200,00 Р	0,00 Р	19 200,00 Р
16	Определение требований к аппара	0,00 Р	Пропорциональное	19 200,00 Р	0,00 Р	19 200,00 Р
26	Анализ качества программного код	0,00 Р	Пропорциональное	16 020,00 Р	0,00 Р	16 020,00 Р
3	Анализ актуальности разработки	0,00 Р	Пропорциональное	16 000,00 Р	0,00 Р	16 000,00 Р
4	Анализ принципов работы сервиса	0,00 Р	Пропорциональное	16 000,00 Р	0,00 Р	16 000,00 Р
8	Определение списка автоматизируе	0,00 Р	Пропорциональное	16 000,00 Р	0,00 Р	16 000,00 Р
24	Анализ и выбор методов тестирова	0,00 Р	Пропорциональное	16 000,00 Р	0,00 Р	16 000,00 Р
30	Ввод в эксплуатацию	0,00 Р	Пропорциональное	9 720,00 Р	0,00 Р	9 720,00 Р
9	Оценка функционально-стоимостны	0,00 Р	Пропорциональное	8 010,00 Р	0,00 Р	8 010,00 Р
5	Анализ видов метаданных и их жи	0,00 Р	Пропорциональное	8 000,00 Р	0,00 Р	8 000,00 Р
6	Анализ схем управления метаданн	0,00 Р	Пропорциональное	8 000,00 Р	0,00 Р	8 000,00 Р
1	Начало проекта	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
10	Анализ предметной области и пост	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
17	Определение требований на разраб	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
22	Разработка программных средств з	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
27	Тестирование программных средств	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
31	Сдача проекта завершена	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
32	Конец проекта	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
		0,00 Р		466 370,00 Р	0,00 Р	466 370,00 Р

Рисунок 8 – Отчет о бюджете

После формирования отчета о бюджете был сформирован отчет «Сводка по проекту» (рисунок 9), отражающий основные показатели, в том числе и объем трудозатрат, необходимых для успешной разработки.

<u>Даты</u>			
Начало:	Ср 12.04.23	Окончание:	Сб 03.06.23
Баз. начало:	НД	Баз. окончание:	НД
Факт. начало:	НД	Факт. окончание:	НД
Отклон. начала:	0 дней	Отклон. окончания:	0 дней
<u>Длительность</u>			
Текущий план:	46 дней	Оставшаяся:	46 дней
Базовый план:	0 дней?	Фактическая:	0 дней
Отклонение:	46 дней	Процент завершени:	0%
<u>Трудозатраты</u>			
Текущий план:	424 ч	Оставшиеся:	424 ч
Базовый план:	0 ч	Фактические:	0 ч
Отклонение:	424 ч	Процент завершени:	0%
<u>Затраты</u>			
Текущий план:	466 370,00 Р	Оставшиеся:	466 370,00 Р
Базовый план:	0,00 Р	Фактические:	0,00 Р
Отклонение:	466 370,00 Р		
<u>Состояние задачи</u>		<u>Состояние ресурса</u>	
Еще не начатые задачи	32	Трудовые ресурсы:	6
Выполняющиеся задачи	0	Трудовые ресурсы с прев. дост	0
Завершенные задачи:	0	Материальные ресурсы:	1
Итого задач:	32	Итого ресурсов:	7

Рисунок 9 – Отчет «Обзор ресурсов»

Общий срок выполнения всех задач, необходимых для разработки, по результатам анализа составил 46 дней.

2 Анализ требований на разработку программных средств

2.1 Функциональные требования к программным средствам

В дополнение к основным механизмам управления, портал управления метаданными для сервиса потоковой обработки событий должен обладать следующими функциями:

- а) авторизация пользователя;
- б) просмотр списка правил;
- в) просмотр систем и их подсистем;
- г) добавление систем и их подсистем;
- д) просмотр событий систем;
- е) просмотр активных правил конкретной системы/подсистемы:
 - 1) просмотр в виде таблицы;
 - 2) просмотр в виде тегов;
- ж) просмотр события конкретной системы/подсистемы:
 - 1) отображение активных правил;
 - 2) графовом представлении;
 - 3) языковом представлении;
 - 4) списковом представлении;
- з) редактирование активных правил:
 - 1) добавление события;
 - 2) добавление условия;
 - 3) добавление действия;
 - 4) удаление правил.

Спецификой данной выпускной квалификационной работы является реализация программных средств поддержки графового представления метаданных для сервиса потоковой обработки событий. Эта цель подразумевает:

- а) выбирать тип объекта:
 - 1) осуществлять быстрое создание абстрактного объекта;
 - 2) назначать связь объекта с другими объектами графа;

- 3) задавать объекту название;
- б) изменять параметры визуализации графового представления:
 - 1) изменять расположение объекта метаданных на графе;
 - 2) изменять расположение связей, соединяющих объекты метаданных;
 - 3) удалять объекты метаданных или связи с графа;
 - 4) редактировать объекты метаданных.

Учитывая данные требования, была разработана диаграмма вариантов использования (рисунок 10).

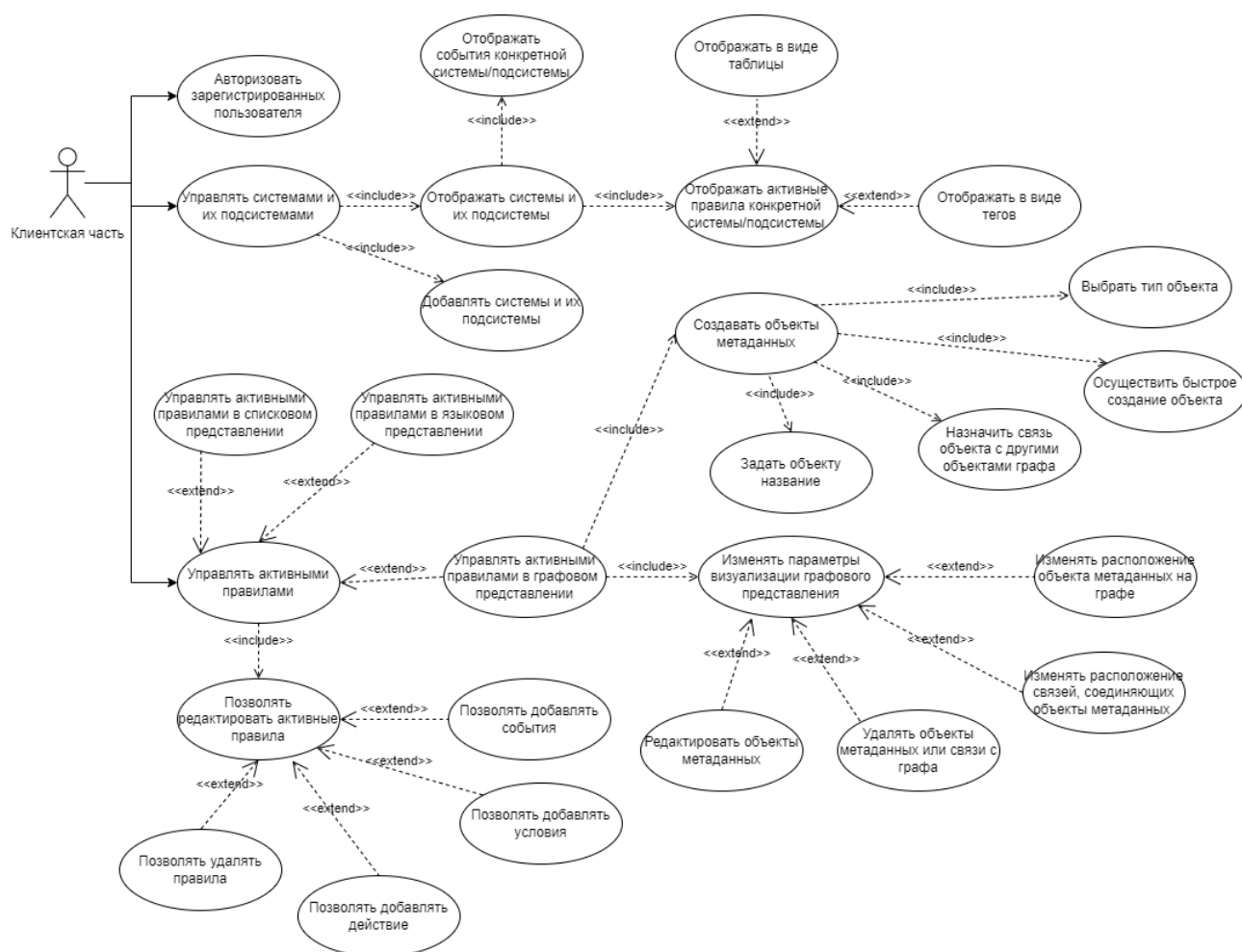


Рисунок 10 – Диаграмма вариантов использования

После изучения функциональных требований была разработана диаграмма вариантов использования для клиентской части портала управления. В рамках этой диаграммы были рассмотрены основные варианты использования.

2.2 Требования к архитектуре программных средств

Для программных средств портала управления метаданными предъявляются следующие требования к их архитектуре.

Клиентская часть должна быть реализована в виде веб-приложения, что позволит обеспечить удобство использования и доступность для пользователей. Использование веб-приложения имеет ряд преимуществ. Во-первых, оно обеспечивает гибкость и мобильность, так как пользователи смогут получить доступ к функциональности и данным приложения с любого устройства, подключенного к интернету. Во-вторых, веб-технологии упрощают процесс развертывания и обновления приложения, так как оно будет установлено только на серверной стороне. При этом, веб-технологии позволят взаимодействовать с активными правилами и событиями как через веб-браузер, так и через отдельное приложение.

Для реализации программных средств поддержки графового представления метаданных для сервиса потоковой обработки событий используется фреймворк React JS, однако не все части приложения написаны на нем. Клиентская часть веб-приложения должна состоять из трех модулей: модуля для Angular (языковое представление), модуля для React (портал и графовое представление) и общего Root модуля, который объединяет эти модули в единую структуру. Такой подход позволяет создать гибкую и масштабируемую архитектуру приложения.

Диаграмма развертывания – это тип UML-диаграммы, которая показывает архитектуру исполнения системы, включая такие узлы, как аппаратные или программные среды исполнения, а также промежуточное программное обеспечение, соединяющее их. Используя диаграмму данного вида, можно понять, как система будет физически развернута на аппаратном обеспечении. Диаграмма развертывания, включающая описанные выше модули, представлена на рисунке 11.

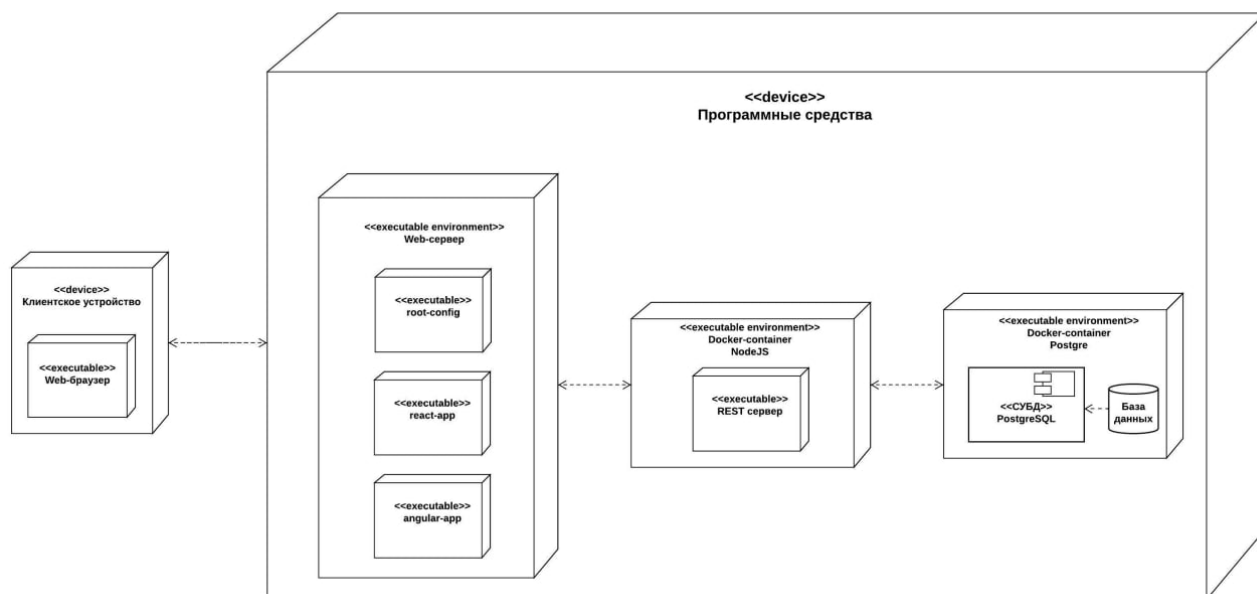


Рисунок 11 – Диаграмма развертывания

Объединяет описанные модули Single SPA, который обеспечивает взаимодействие между ними, предоставляет единый пользовательский интерфейс, а также обеспечивает согласованность данных и состояния приложения, упрощая процесс развертывания и поддержки системы [11].

Серверная часть архитектуры портала управления метаданными играет важную роль в обработке данных и взаимодействии с клиентской частью. Она состоит из специального REST-сервера, который выполняет функцию промежуточного звена между клиентами или микросервисами и базой данных.

REST-сервер обрабатывает поступающие HTTP-запросы от клиентов, проверяет их и выполняет SQL-запросы к базе данных. Он выполняет роль посредника, обеспечивая безопасную и эффективную передачу данных между клиентами и базой данных.

Модули REST-сервера включают различные функциональные блоки. В них реализованы следующие возможности: регистрация пользователей, авторизация и аутентификация, хранение метаданных пользователей и их ролей, модуль событий, компонентов, системы, активных правил, работы с базой данных, микросервисами и локальными репозиториями. Каждый модуль

выполняет определенные задачи, связанные с обработкой данных и предоставлением необходимой функциональности.

При получении запросов от клиентов, REST-сервер проверяет их и выполняет соответствующие SQL-запросы к базе данных. Полученные данные обрабатываются и упаковываются в формат JSON, который затем отправляется обратно клиентам или микросервисам.

Для повышения производительности и быстрого доступа к данным в архитектуре используется Redis - резидентная система управления базами данных, работающая в качестве буферного слоя. Управляющий модуль на REST-сервере и локальные хранилища данных на микросервисах используют Redis для хранения и получения данных.

При запросе метаданных клиент или микросервис сначала обращается к локальному репозиторию Redis для проверки наличия данных. Если данные отсутствуют, то происходит обращение к REST-серверу, который обновляет локальные репозитории при отправке метаданных. Это обеспечивает актуальность данных и оптимизирует работу с кэшем микросервисов, что способствует повышению производительности и снижению нагрузки на базу данных.

2.3 Требования к пользовательскому интерфейсу программных средств

Разработка пользовательского интерфейса, который сочетает в себе функциональность, понятность и логичную структуру, необходима для обеспечения удобного доступа пользователей к метаданным сервиса потоковой обработки событий. Интерфейс должен быть легко освоен для новых пользователей, поскольку его основной задачей является обеспечение высокой производительности работы и минимальной возможности ошибки со стороны пользователей.

Интерфейс пользователя разрабатываемого приложения реализуется средствами фреймворка React JS совместно с инструментарием визуализации

графов VIS.js. Основной упор в интерфейсе должен быть сделан на графическое представление метаданных в виде графов. Графическое отображение позволяет наглядно представить связи и зависимости между элементами данных, упрощает анализ и понимание структуры информации [12].

Пользователям следует предоставить возможность просмотра и редактирования атрибутов и свойств элементов графа. Интерфейс должен предоставлять понятное отображение информации об элементах, их атрибутах и связях.

Также пользователям должны быть доступны различные инструменты и возможности для манипуляции с графами. Они должны иметь возможность добавлять, удалять и изменять элементы графа, а также задавать свойства и связи между ними. Интерфейс должен предоставлять интуитивные средства редактирования графов, например, возможность перетаскивания элементов или изменения их размеров.

По результатам анализа функциональных требований, необходимо разработать страницу управления объектами метаданных в графовом представлении. Прототип страницы, выполненный в графическом редакторе Figma, представлен на рисунке 12.



Рисунок 12 - Прототип страницы управления объектами метаданных в графовом представлении

В целом, пользовательский интерфейс программных средств поддержки графового представления метаданных должен соответствовать таким параметрам, как: интуитивность, наглядность и удобство работы с данными, способствуя более эффективной потоковой обработке событий и анализу структуры данных.

2.4 Обоснование выбора технологий и средств разработки

В создании программных средств поддержки графового представления метаданных для сервиса потоковой обработки событий были выбраны технологии, описанные далее.

HTML является основным языком разметки веб-страниц и позволяет создавать структуру и содержимое страницы. Он широко поддерживается всеми браузерами, что обеспечивает совместимость с различными платформами.

CSS используется для стилизации веб-страниц, позволяя задавать внешний вид и расположение элементов на странице. Он обеспечивает гибкость и возможность создания эстетически привлекательного пользовательского интерфейса.

JavaScript является широко распространенным языком программирования, который поддерживается на всех основных платформах и браузерах. Он обеспечивает интерактивность и динамичность веб-приложений, позволяя создавать сложную функциональность.

TypeScript является расширением JavaScript, которое добавляет статическую типизацию и другие возможности для более безопасной и организованной разработки. Он упрощает обнаружение ошибок и обеспечивает лучшую предсказуемость кода.

VIS.js предоставляет богатые возможности для работы с графовым представлением данных. Он позволяет визуализировать и манипулировать графами, что является важным для потоковой обработки событий.

React JS – это популярный JavaScript-фреймворк, который обеспечивает эффективное создание пользовательского интерфейса с использованием

компонентного подхода. Он обладает большим сообществом разработчиков и богатой экосистемой, что облегчает разработку сложных приложений.

Single SPA – это фреймворк для создания микрофронтенд-архитектуры. Он позволяет комбинировать различные фреймворки и библиотеки в единое приложение, что обеспечивает гибкость и масштабируемость разработки.

CSS module позволяет организовать модульную структуру стилей, предотвращая конфликты и обеспечивая локальную область видимости стилей. Это улучшает поддерживаемость и расширяемость кода.

Redux – это библиотека управления состоянием приложения. Она обеспечивает предсказуемость и упрощает управление данными, особенно при работе с большими объемами информации и сложными состояниями.

БЭМ (Блок-Элемент-Модификатор) – это методология разработки пользовательского интерфейса, которая обеспечивает модульность, переиспользование и четкую структуру кода. Она способствует разделению интерфейса на независимые компоненты.

GIT является системой контроля версий, которая позволяет эффективно управлять исходным кодом при работе в команде. Он обеспечивает отслеживание изменений, ветвление, слияние и управление версиями кода.

Eslint и Prettier – это инструменты для статического анализа и форматирования кода. Они помогают поддерживать стандарты кодирования, обнаруживать ошибки и повышать читаемость и согласованность кода.

Yarn – это пакетный менеджер, который облегчает установку, обновление и управление зависимостями проекта. Он предлагает быструю и надежную работу с пакетами.

SCSS (Sass) – это препроцессор CSS, который предоставляет расширенные возможности для написания стилей, такие как переменные, миксины и вложенность. Он повышает эффективность и улучшает организацию стилей.

Webpack – это инструмент сборки, который позволяет объединять и упаковывать различные модули и ресурсы в один оптимизированный пакет. Он

упрощает развертывание приложения и обеспечивает эффективное управление зависимостями.

Docker – это платформа для контейнеризации приложений, которая обеспечивает изоляцию и портативность при развертывании приложений в различных средах. Он упрощает установку и управление приложением в контейнере.

Figma – это инструмент для дизайна интерфейсов, который обеспечивает совместную работу и проектирование пользовательского интерфейса. Он позволяет создавать прототипы и визуальные макеты, что помогает в процессе разработки.

Разработка программных средств будет произведена с использованием операционной системы Windows и браузера Google Chrome в среде разработки Webstorm.

Описанные выше технологии выбраны благодаря их способности решать конкретные задачи, обеспечивать эффективность разработки, удобство использования, совместимость и расширяемость приложения.

2.5 Требования к аппаратно-программному обеспечению

2.5.1 Требования к аппаратно-техническому обеспечению

Аппаратно-техническое обеспечение для средств поддержки графового представления метаданных сервиса потоковой обработки событий играет важную роль в обеспечении эффективности работы и производительности приложения. Для удовлетворения требований функциональности и производительности следует учесть следующие аспекты аппаратного и технического обеспечения: центральный процессор, оперативную память, сетевое оборудование и прочие периферийные устройства.

В таблице 4 приведены рекомендуемые требования к устройству, на котором будет запускаться web-приложение.

Таблица 4 – Требования к аппаратно-техническому оборудованию

Требование	Рекомендуемые
Процессор	2,8 ГГц
Количество вычислительных потоков процессора	8
Оперативная память	4Гб
Свободное дисковое пространство	10 ГБ

Разрабатываемое программное средство не требует значительных ресурсов и нуждается в стандартных значениях.

2.5.2 Требования к системному программному обеспечению

Требования к системному программному обеспечению определяют те операционные системы, под которыми будет происходить работа приложения. Рекомендуемыми версиями для операционных систем являются:

- Microsoft Windows (64-bit) 10, 11;
- Apple OS X 10.7 и более поздние;
- Ubuntu 16.04 и более поздние.

2.5.3 Требования к прикладному программному обеспечению

На основе выбранных технологий для реализации определен перечень поддерживаемых браузеров, на которых приложение будет отображаться и работать наиболее корректно и быстро:

- Google Chrome 28.0 и более поздние;
- Mozilla Firefox 47.0 и более поздние;
- Apple Safari 8.0 и более поздние (для OS X);
- Microsoft Edge 13.0 и более поздние;
- Opera 36.0 и более поздние.

3 Разработка программных средств

3.1 Разработка архитектуры развертывания программных средств поддержки графового представления метаданных для сервиса потоковой обработки событий

Исходя из списка требований, изложенного в предыдущем разделе, был сформирован перечень ключевых компонентов, составляющих разрабатываемые программные средства поддержки графового представления метаданных для сервиса потоковой обработки событий. Данные компоненты наглядно представлены на диаграмме, изображенной на рисунке 13, где проиллюстрирована их взаимосвязь и организация.

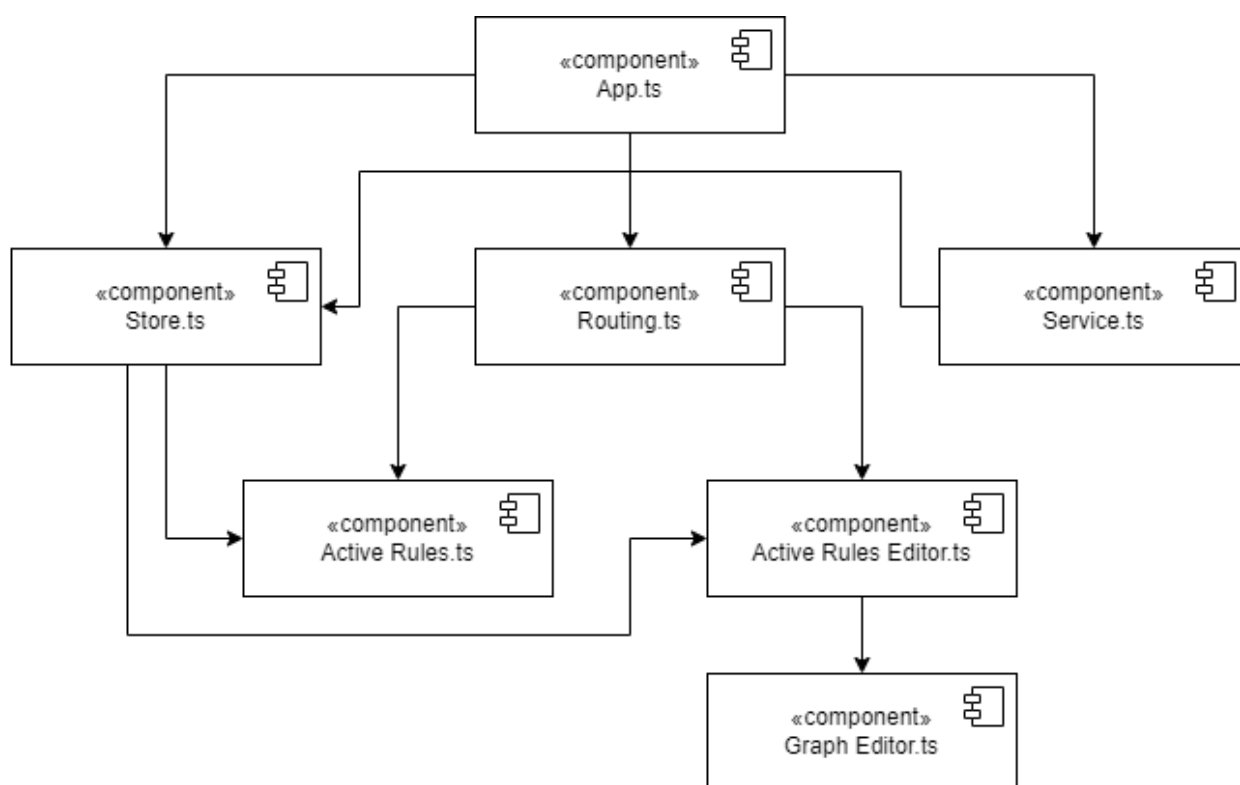


Рисунок 13 – Диаграмма основных компонентов веб-приложения

Более детальное содержание компонентов приведено на диаграмме пакетов, которая изображена на рисунке 14.

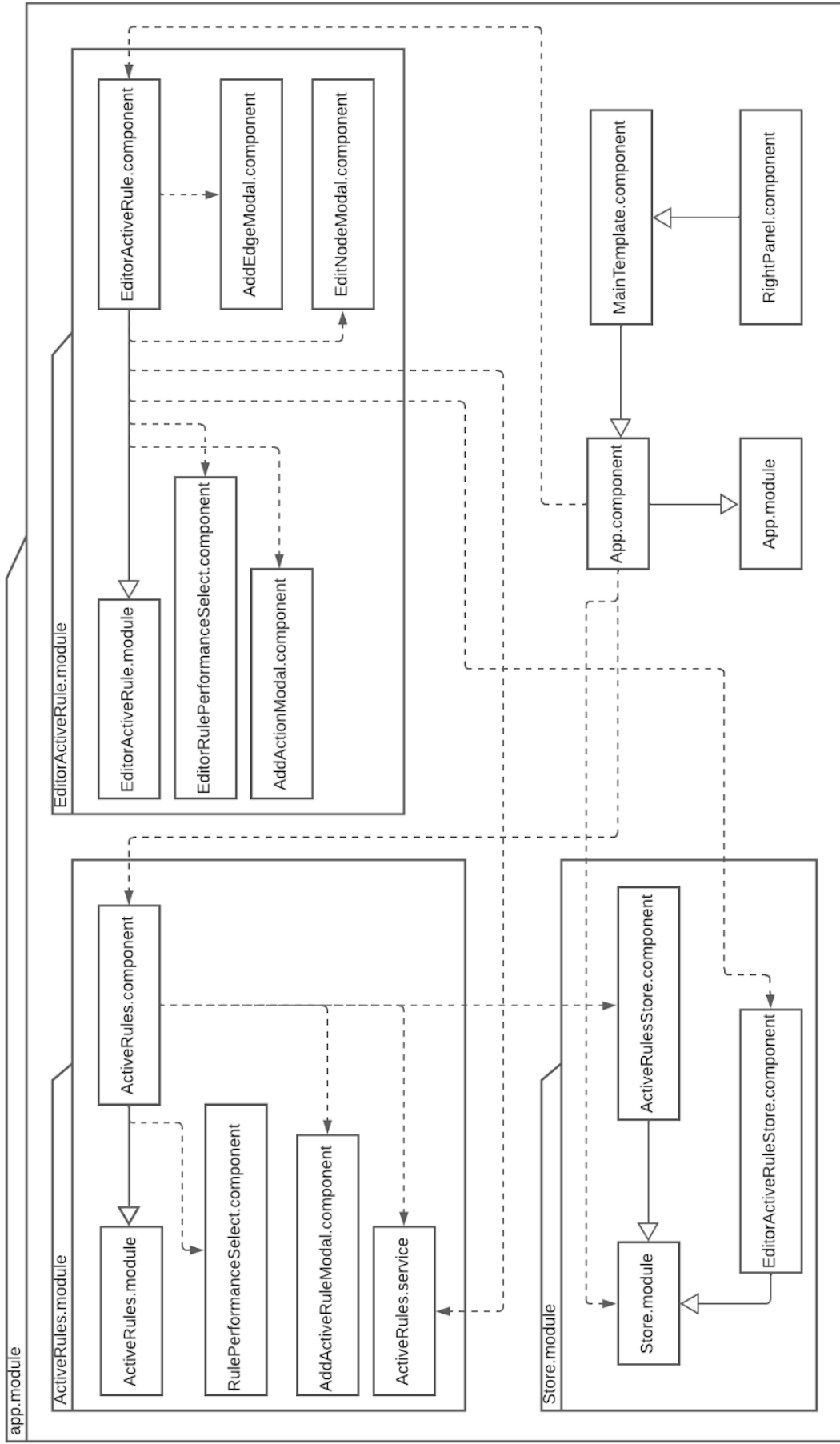


Рисунок 14 –Архитектура элементов приложения в виде диаграммы пакетов

Компоненты диаграммы пакетов описаны в таблице 5.

Таблица 5 – Компоненты программных средств поддержки графового представления метаданных для сервиса потоковой обработки событий

№	Сущность	Описание
1	App.module	Основной модуль приложения, включающий в себя зависимости всех остальных модулей и сервисов
2	Route.module	Модуль, управляющий переадресацией на страницах
3	Api.module	Модуль для хранения всех запросов для связи с бэкенд-частью приложения
4	Store.module	Модуль для хранения различных состояний системы
5	ActiveRules.module	Модуль для управления активными правилами
6	EditorActiveRule.module	Модуль для редактирования состояния активного правила
7	MainTemplate.component	Компонент, отвечающий за внешнюю стилизацию всех компонентов системы
8	RightPanel.component	Компонент правой боковой панели для управления текущими модулями и навигации
9	AddSystemModal.component	Компонент модального окна создания системы
10	ActiveRuleTable.component	Компонент таблицы с активными правилами
11	ActiveRuleTag.component	Компонент страницы с тэговым представлением активных правил
12	RulePerformanceSelect.component	Компонент выпадающего списка для выбора текущего отображения для активных правил
13	AddActiveRuleModal.component	Компонент модального окна для добавления активного правила
14	EditorRulePerformanceSelect.component	Компонент выпадающего списка для выбора текущего представления отображения редактора активного правила
15	Helpers.component	Компонент для основного управляющего функционала (добавление, редактирование, удаление)
16	ActiveRulesStore.component	Компонент для хранения состояний активных правил
17	EditorActiveRuleStore.component	Компонент для хранения состояний редактора активных правил
18	EditorRuleGraph.component	Компонент для взаимодействия с графовым представлением
19	GraphComponent.component	Компонент отображения графа
20	AddActionModal.component	Компонент модального окна для добавления действия
21	AddEdgeModal.component	Компонент модального окна для добавления связей между узлами

Продолжение таблицы 5

№	Сущность	Описание
22	EditNodeModal.component	Компонент модального окна для редактирования узлов графа
23	ActiveRules.service	Сервис, хранящий http-запросы для управления активными правилами

В вышеизложенной таблице есть и модули, и компоненты, и сервисы.

Модули представляют собой обобщающие блоки, содержащие компоненты и сервисы, связанные с этим модулем.

Компоненты реализуют пользовательский интерфейс и содержат программный код по взаимодействию с данным интерфейсом. Внутри себя компоненты также содержат дополнительные файлы HTML и CSS для визуализации соответствующих элементов.

Сервисы предоставляют компонентам дополнительную логику взаимодействия с другими компонентами, сервисами, моделями или внешними системами.

3.2 Разработка визуальных примитивов и общих средств визуализации для графового представления метаданных сервиса потоковой обработки событий

Визуальное представление является ключевым аспектом в разработке графа для управления метаданными. Перед реализацией графового представления необходимо разработать визуальные примитивы.

Для отображения таких объектов метаданных, как события, условия и действия были выбраны следующие визуальные примитивы:

- а) эллипс для события;
- б) ромб для условия;
- в) прямоугольник для действия.

Пример использования визуального представления отображен на рисунке 15.

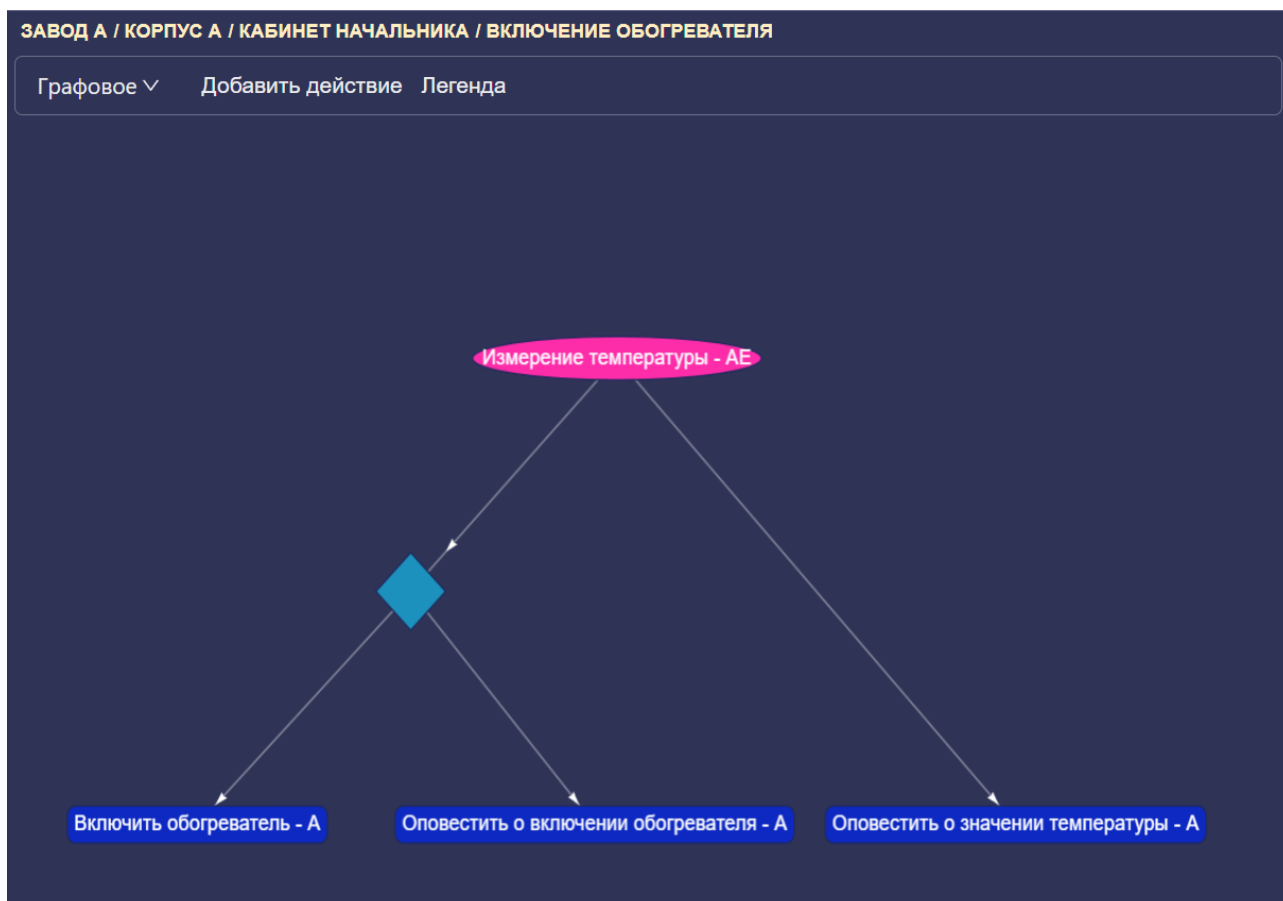


Рисунок 15 – Визуальное представление события, условия и действий

Связи между перечисленными выше объектами отображаются в структуре графа при помощи стрелок. Существует несколько вариантов построения таких связей, однако все они должны отвечать следующим условиям:

- в активном правиле должно быть только одно событие;
- в активном правиле должно быть только одно условие;
- в активном правиле может быть неограниченное количество действий;
- связи между объектами активного правила могут строиться как напрямую от события к действию, так и через условие.

На основе вышеперечисленных условий сформированы виды связей события с действием:

- от события к условию, а затем к действию;

- от события к одному или нескольким действиями, без использования условия;
- от события к условию вида «if-else», а затем, соответственно, к двум противоположным действиям.

Внутри геометрических примитивов любого вида должно отображаться название объекта. Дополнительная же информация, например, код объекта, его источник и т.д., отражаются во всплывающем при наведении на объект окне.

Все описанные объекты и сам граф отрисовываются с помощью библиотеки VIS.js. В листинге 1 представлен фрагмент кода для редактирования объекта метаданных в структуре графа.

Листинг 1 – Редактирования объекта метаданных в структуре графа

```
export const onEditNodeFromAR = async (currentNodeId: string,
newLabel: string) => {
  const actions = JSON.parse(currentActiveRuleObject.action);
  const editedNode = actions.data.find((act: {code: string}) =>
act.code === currentNodeId);
  const newData = actions.data.map((d: {category: string,
description: string, code: string}) => {
    if (d.code === editedNode?.code) {
      return {
        ...editedNode,
        description: `${newLabel}`
      };
    }
    return {...d};
  });
  const requestBody = {
    description: currentActiveRuleObject.description,
    condition: currentActiveRuleObject.condition,
    action: JSON.stringify({
      data: [...newData],
      edges: [...actions.edges]
    }),
    code: currentActiveRuleObject.code
  };
  currentSystemCode && await
store.dispatch(updateActiveRule(requestBody, currentSystemCode));
};
```

Полный код управления объектами метаданных представлен в приложении Б.

3.3 Разработка программных средств редактирования графового представления метаданных сервиса потоковой обработки событий

Графовое представление позволяет управлять активными правилами и их составляющими. Для редактирования графа сначала необходимо выбрать то активное правило, внутри которого пользователь будет работать.

После успешной авторизации пользователя в веб-приложении ему становится видна страница со списком всех доступных для просмотра и управления систем. В левой боковой панели отображаются родительские системы, по умолчанию выбрана первая в списке родительская система. Для просмотра подсистем нужно дважды кликнуть на выбранную систему, после чего в основной рабочей области отобразятся карточки подсистем, входящих в выбранную систему. Получение системы происходит с помощью метода GET /system.

Выбрав подсистему, пользователь, кликнув на ссылку «Список правил» в правом боковом меню, может перейти на страницу отображения всех активных правил, относящихся к выбранной подсистеме.

Диаграмма классов модели активных правил показана на рисунке 16.

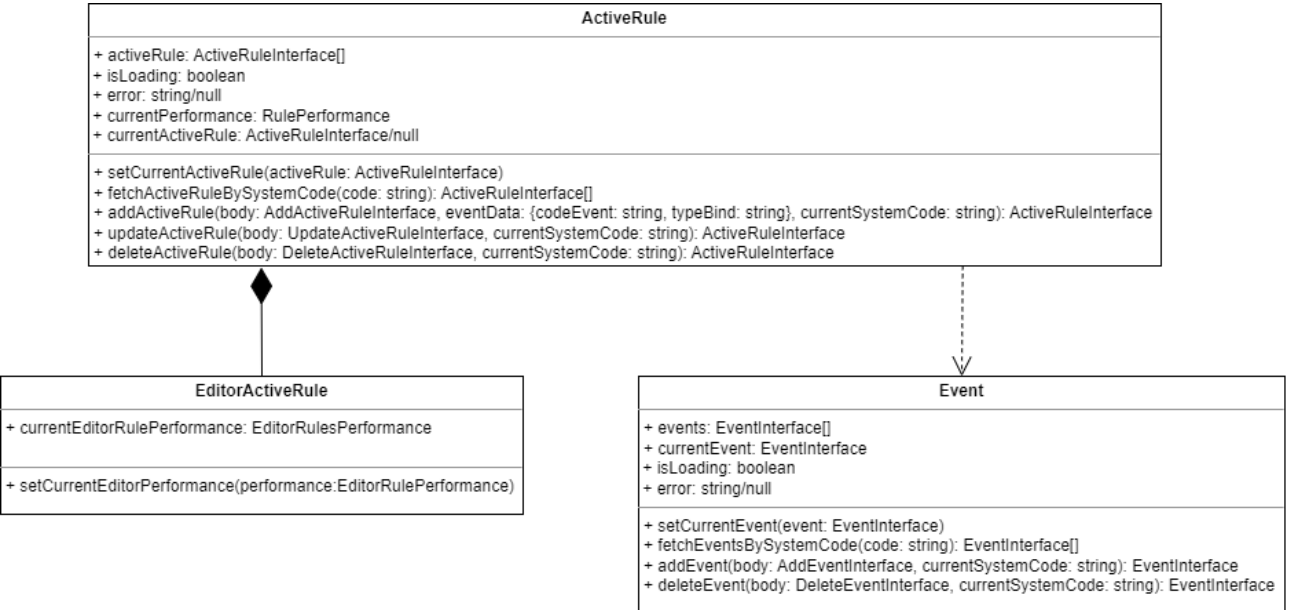


Рисунок 16 – Диаграмма классов модели активных правил

Получение правил происходит с помощью метода GET /active-rule/system/*код выбранной подсистемы*. Кликнув дважды по активному правилу, пользователь переходит на страницу редактирования. Редактировать активное правило можно в нескольких представлениях:

- списочное;
- графовое;
- языковое.

Подробнее рассмотрим графовое представление.

После выбора из выпадающего списка графового представления в рабочей панели отображается граф, состоящий из компонентов выбранного на предыдущих шагах активного правила.

Процесс создания и редактирования активного правила представлен на диаграмме деятельности на рисунке 17.

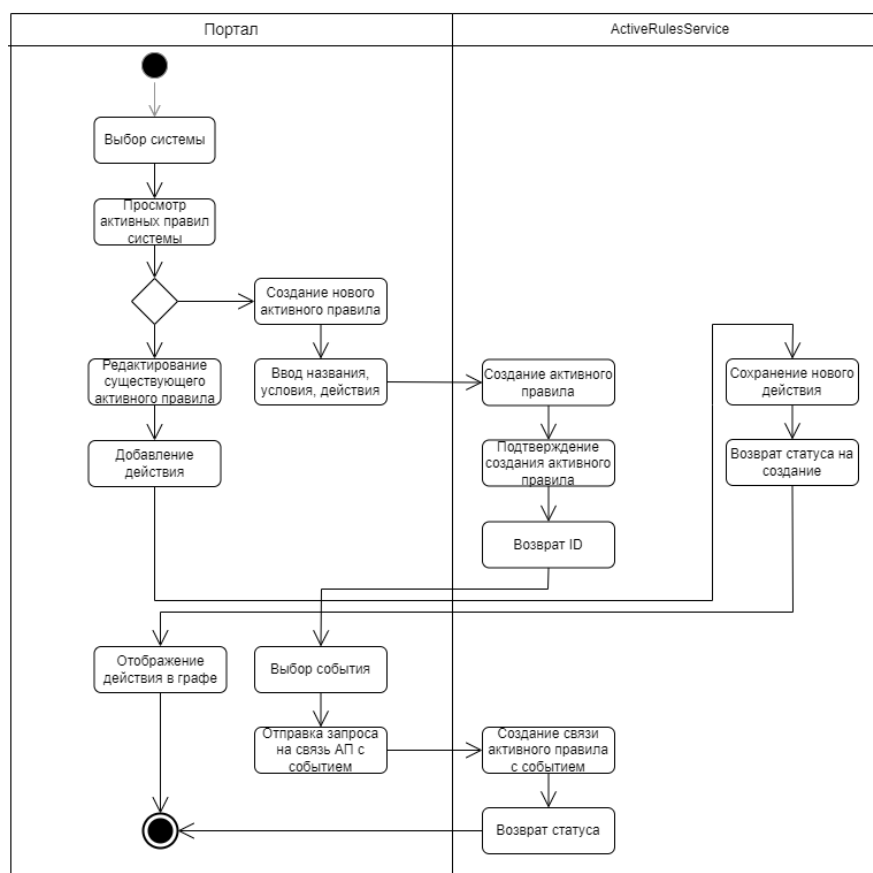


Рисунок 17 – Диаграмма деятельности создания и редактирования
активного правила

В граф пользователь может добавить новый узел действия. Для этого ему необходимо кликнуть на кнопку «Добавить действие», а затем в открывшемся модальном окне ввести название действия и условие, от которого оно будет строиться. После сохранения структура графа изменяется с помощью метода PATCH /active-rule/, который обновляет активное правило.

Помимо добавления действия, в системе также реализовано и добавление связей между объектами. Для этого необходимо в правой боковой панели кликнуть на кнопку «Добавить связь», а затем в открывшемся модальном окне выбрать из уже созданных в этом активном правиле и отображенных в графе те событие, условие и действие, с которыми будет взаимодействовать новая связь. При этом выбор условия является необязательным. После сохранения структура графа изменяется с помощью метода PATCH /active-rule/, который обновляет активное правило. В листинге 2 представлен фрагмент кода для добавления связи между элементами в структуре графа.

Листинг 2 – Редактирования объекта метаданных в структуре графа

```
export const onAddEdgeInAR = async (data: {condition: string,
event: string, action: string}) => {
  const actions = JSON.parse(currentActiveRuleObject.action);
  if (currentActiveRuleObject.event[0].association.typeBind ===
'Event to Rule') {
    if (data.condition) {
      actions.edges.push({from: data.event, to:
data.condition});
      actions.edges.push({from: data.condition, to:
data.action});
    } else {
      actions.edges.push({from: data.event, to:
data.action});
    }
  } else {
    if (data.condition) {
      actions.edges.push({from: data.condition, to:
data.event});
      actions.edges.push({from: data.condition, to:
data.action});
    } else {
      actions.edges.push({from: data.action, to:
data.event});
    }
  }
}
```

```

    }
    const requestBody = {
      description: currentActiveRuleObject.description,
      condition: currentActiveRuleObject.condition,
      action: JSON.stringify({data: actions.data, edges:
actions.edges}),
      code: currentActiveRuleObject.code
    };
    currentSystemCode && await
store.dispatch(updateActiveRule(requestBody, currentSystemCode));
};

```

Диаграмма состояний активного правила показана на рисунке 18.

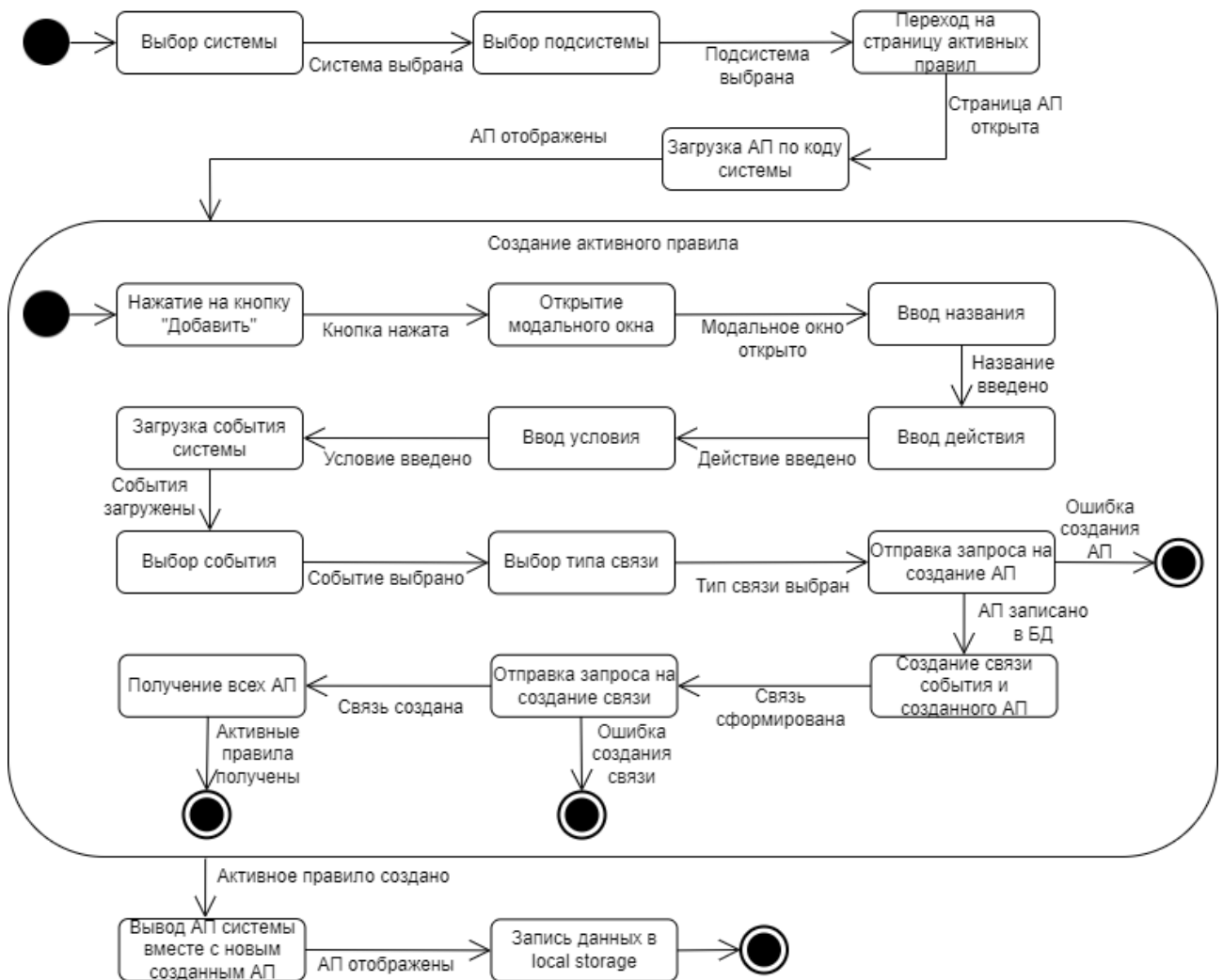


Рисунок 18 – Диаграмма состояний создания активного правила

Кроме того, пользователь может удалять элементы графа. Если необходимо удалить ноду графа, то можно сделать это, выбрав кликом необходимый объект, а затем нажав на кнопку «Удалить» в правом боковом меню. Удалить связь между элементами графа можно аналогичным способом.

При удалении также используется метод PATCH /active-rule/, который перезаписывает объекты текущего активного правила.

Все объекты активного правила, отрисованные на графе, хранятся в локальном хранилище браузера для того, чтобы при перезагрузке страницы визуализация графа не изменялась. При любых действиях управления (таких как добавление, изменение и удаление) локальное хранилище перезаписывается.

3.4 Разработка пользовательского интерфейса

После анализа требований и разработки архитектуры программных средств был спроектирован удобный пользовательский интерфейс для управления активными правилами в графовом представлении.

Все управление элементами активного правила на графе происходит в рамках одной страницы, представленной на рисунке 19.

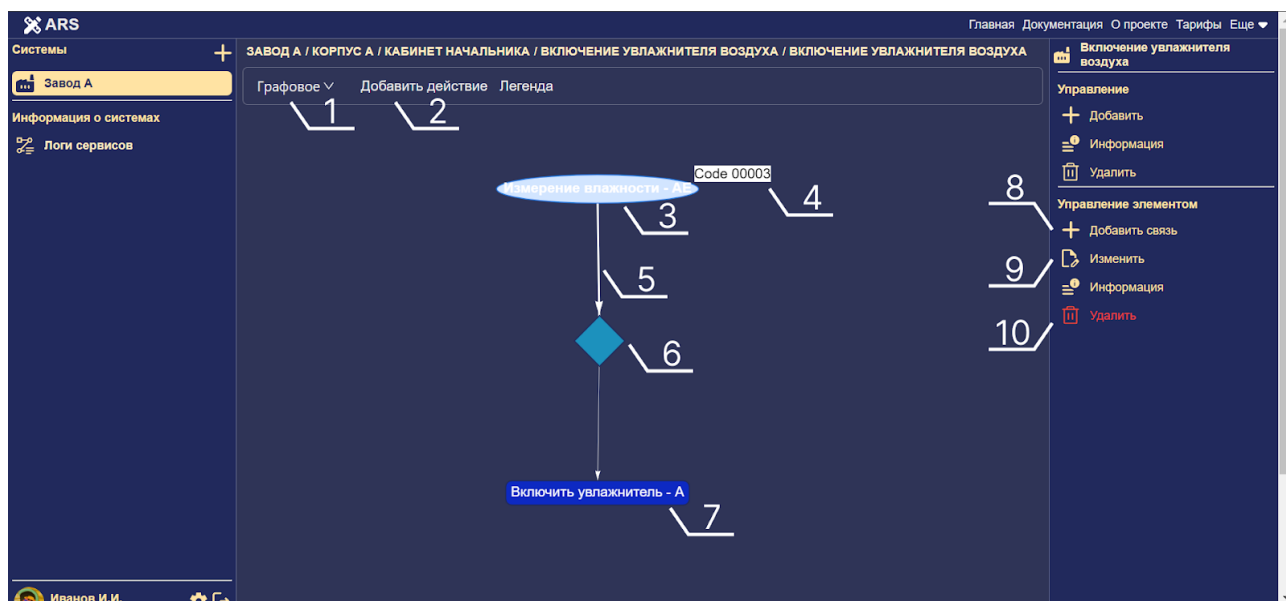


Рисунок 19 – Страница графового представления активного правила

Назначение элементов страницы графового представления активных правил описано ниже:

- 1 – выпадающий список для выбор представления активного правила;
- 2 – кнопка добавления нового действия в структуру графа;
- 3 – событие активного правила;

- 4 – информация о коде элемента, всплывающая при наведении;
- 5 – связь элементов графа;
- 6 – условие активного правила;
- 7 – действие активного правила;
- 8 – кнопка добавления новой связи для уже существующих элементов графа;
- 9 – кнопка изменения элементов графа;
- 10 – удаление элементов графа.

После нажатия на кнопку добавления действия, пользователь может заполнить необходимые для создания действия поля в модальном окне, отображенном на рисунке 20.

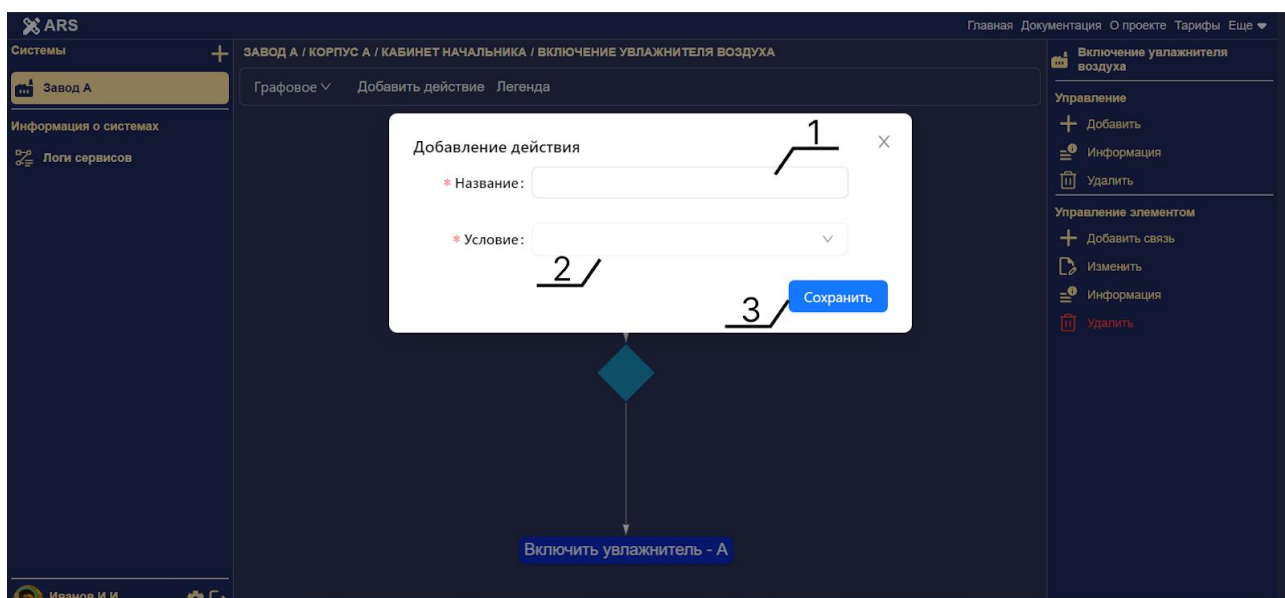


Рисунок 20 – Модальное окно добавления нового действия в структуру графа выбранного активного правила

Рассмотрим подробнее структуру модального окна:

- 1 – поле ввода названия добавляемого действия;
- 2 – выпадающий список для условия, от которого будет строиться действие в структуре графа;
- 3 – кнопка сохранения введенных данных;

Помимо добавления новых действий, пользователь может добавить и новую связь в структуру графа. Модальное окно для добавления связи отображено на рисунке 21.

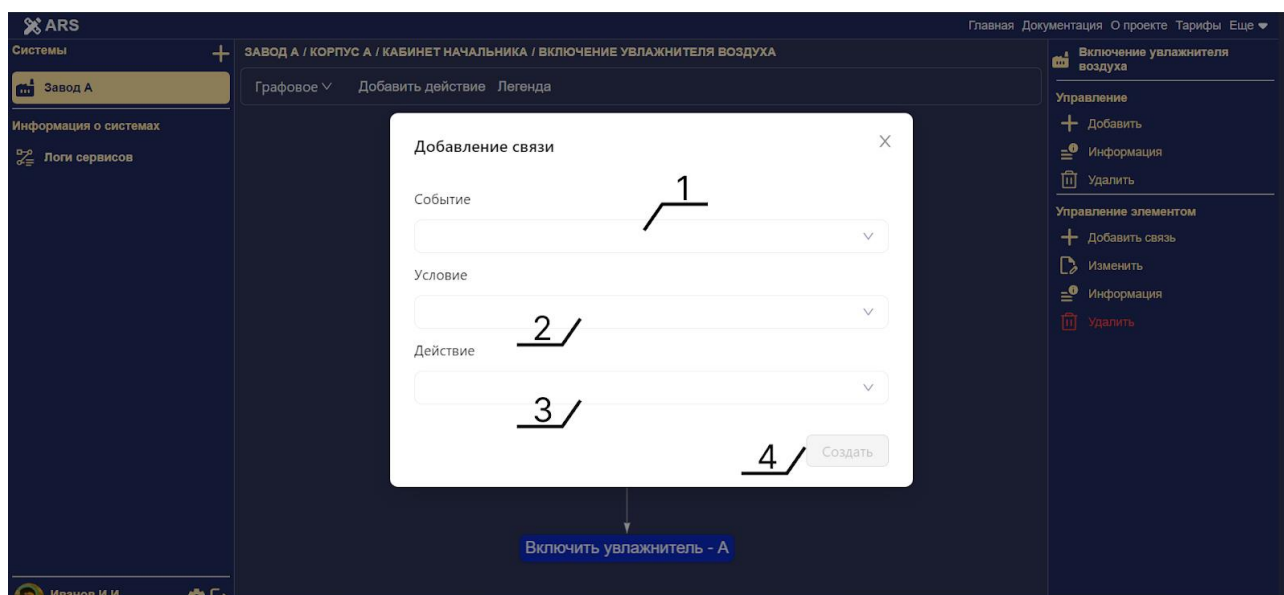


Рисунок 21 – Модальное окно добавления связи элементов графа

Рассмотрим подробнее структуру модального окна добавления связи элементов графа:

- 1 – выпадающий список для выбора события;
- 2 – выпадающий список для выбора условия;
- 3 – выпадающий список для выбора действия;
- 3 – кнопка создания связи.

В данном модальном окне доступна возможность выбора отсутствия связи, а также предусмотрена блокировка кнопки “Создать” до заполнения всех полей.

Кроме добавления элементов графа пользователю доступно также и их редактирование. Для изменения названия любого элемента графа используется модальное окно, показанное на рисунке 22. Пользователь может редактировать как действия, так и события и условия.

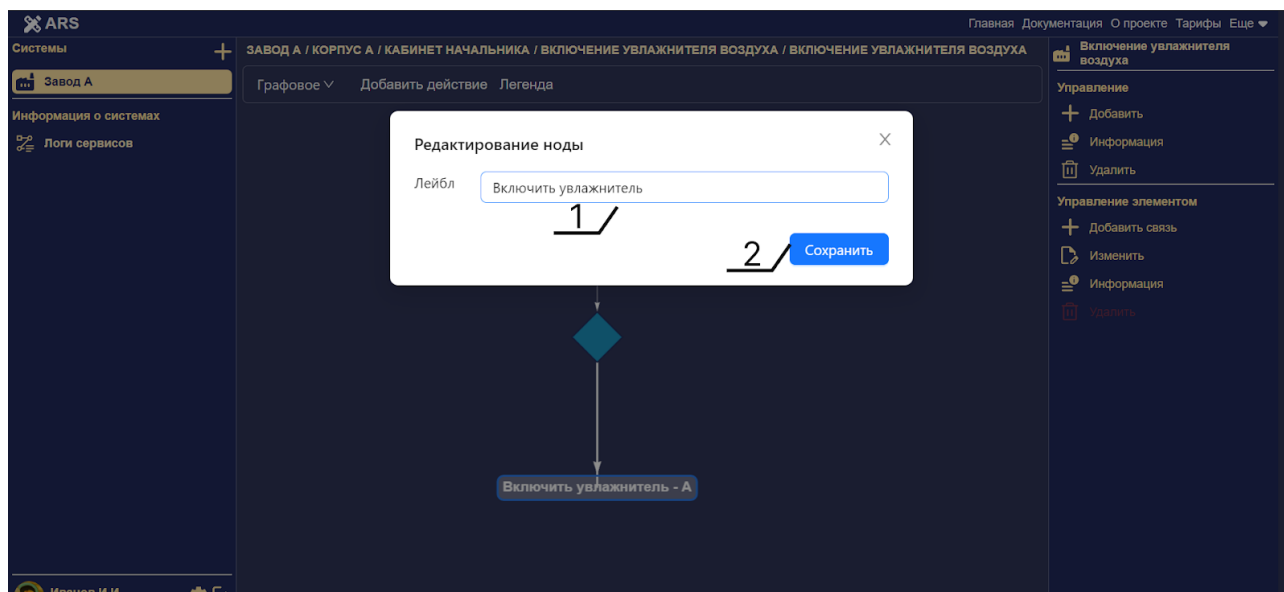


Рисунок 22 – Модальное окно редактирования элементов графа

В данном модальном окне присутствуют следующие элементы:

- 1 – поле ввода названия редактируемого элемента;
- 2 – кнопка сохранения введенных данных;

Разработанный пользовательский интерфейс позволяет легко совершать различные операции управления элементами активных правил в графовом представлении.

4 Контроль качества программных средств

4.1 Планирование процесса контроля качества программных средств

4.1.1 Мероприятия по контролю качества программных средств

Мероприятия по контролю качества программных средств представляют собой систематические действия, направленные на обеспечение соответствия программного продукта установленным стандартам, требованиям и ожиданиям заказчиков продукта. Они являются неотъемлемой частью процесса разработки и обслуживания программных средств.

Основная цель проведения мероприятий по контролю качества заключается в обнаружении и устранении дефектов, а также в повышении общего уровня надежности и эффективности программных средств. Эти мероприятия позволяют предотвратить потенциальные проблемы, связанные с работой программного продукта, и улучшить его функциональность и удобство использования.

Мероприятия по контролю качества следует проводить на различных этапах жизненного цикла программного продукта. Они могут быть начаты еще на стадии проектирования, чтобы предупредить ошибки и недоработки в архитектуре системы. Также контроль качества необходимо осуществлять во время разработки, чтобы выявлять и исправлять дефекты, и в процессе эксплуатации программного продукта, чтобы обеспечивать его стабильную работу и поддержку.

Мероприятия по контролю качества программных средств позволяют снизить риски, связанные с непредвиденными сбоями и ошибками, что в свою очередь способствует повышению доверия пользователей к программному продукту [13].

Мероприятия по контролю качества программных средств могут быть разделены на две основные группы: анализ кода и подсчет метрик, и тестирование программных средств.

Анализ кода и подсчет метрик включают в себя оценку качества программного кода на основе набора критериев и стандартов. Это позволяет выявить потенциальные проблемы, такие невыполнение стандартов форматирования кода, нарушения архитектурных принципов и другие аспекты, которые могут отрицательно сказаться на качестве программного продукта. Метрики в свою очередь предоставляют количественные оценки характеристик программного кода, таких как сложность, размер, читаемость и другие.

Тестирование программных средств включает в себя набор процедур, которые позволяют проверить работоспособность, функциональность и соответствие требованиям программного продукта. Виды тестирования могут варьироваться в зависимости от конкретных требований и характеристик программного продукта. Они могут включать модульное тестирование, интеграционное тестирование, системное тестирование, приемочное тестирование и другие формы проверки.

4.1.2 Обоснование выбора модели тестирования

Существует несколько основных видов тестирования ПО, каждый из которых имеет свои особенности и цели [14].

Функциональное тестирование направлено на проверку функциональности программного обеспечения. В ходе функционального тестирования проверяются входные данные, выходные результаты, поведение системы при различных сценариях использования и соответствие функциональным требованиям.

Нагрузочное тестирование проводится для оценки производительности и стабильности ПО при различных нагрузках. Целью нагрузочного тестирования является определение предельных возможностей системы, выявление узких мест

и проблем, связанных с производительностью, а также определение максимальной нагрузки, которую система может выдержать.

Конфигурационное тестирование направлено на проверку работы программного обеспечения на различных конфигурациях и настроенных условиях среды. Конфигурация включает в себя такие факторы, как аппаратное обеспечение, операционные системы, браузеры, базы данных и другие компоненты, которые могут влиять на работу ПО.

Интеграционное тестирование проверяет взаимодействие различных компонентов или модулей ПО. Целью этого тестирования является проверка того, что компоненты работают корректно вместе и передают друг другу необходимые данные.

Регрессионное тестирование проводится после внесения изменений в программное обеспечение, чтобы убедиться, что изменения не привели к появлению новых дефектов или не повлияли на уже работающую функциональность. Регрессионное тестирование помогает обнаружить и устранить возможные проблемы, возникающие в результате изменений.

Автоматизированное тестирование подразумевает использование специальных инструментов и программных средств позволяет автоматизировать выполнение тестовых сценариев и проверку результатов. Автоматизация тестирования помогает сократить время и ресурсы, улучшить покрытие тестирования и повысить его эффективность.

Каждый из этих видов тестирования имеет свои особенности и может быть применен в зависимости от конкретных требований, характеристик и целей проекта разработки программного обеспечения. Для контроля качества программных средств поддержки графового представления метаданных сервиса потоковой обработки событий были выбраны функциональное и конфигурационное виды тестирования.

Функциональное тестирование является одним из наиболее распространенных видов тестирования ПО. В случае ПС поддержки графового

представления метаданных, функциональное тестирование позволяет проверить корректность визуализации данных и управления ими.

Конфигурационное тестирование, в свою очередь, направлено на проверку работоспособности программного обеспечения в различных конфигурациях и средах. Для программных средств поддержки графового представления метаданных, это может означать проверку работы в разных браузерах, операционных системах или на разных устройствах. Такое тестирование позволит убедиться в том, что ПС адаптированы и работает стабильно в различных условиях. Конфигурационное тестирование важно для обеспечения совместимости и надежности программных средств в различных средах использования.

Обоснование выбора функционального тестирования и конфигурационного тестирования основывается на необходимости обеспечения высокой степени соответствия функциональных требований и надежности работы программного средства в разных конфигурациях и средах. Эти виды тестирования помогут выявить потенциальные проблемы и дефекты, а также обеспечат уверенность в работоспособности и стабильности программного средства, что в свою очередь приведет к улучшению пользовательского опыта и общего качества системы потоковой обработки событий.

Модульное же тестирование позволяет изолировать и тестировать отдельные модули или компоненты программного обеспечения независимо от других модулей. Модульное тестирование позволяет проверить каждый модуль на правильность работы в изоляции, обеспечивая более надежное и предсказуемое поведение.

4.1.3 Обоснование выбора метрик программного кода

Выбор в качестве метрик программного кода анализа абстрактности и устойчивости программного обеспечения, метрик Холстеда, а также оценки качества пользовательского интерфейса основывается на способности предоставить достаточную информацию о качестве и характеристиках

программного кода и оценить насколько быстро страница графowego представления отвечает на действия пользователя.

Анализ абстрактности и устойчивости программного обеспечения является важным аспектом контроля качества.

Метрика абстрактности показывает отношение количества абстрактных классов к общему числу классов в компоненте. На основании данных метрик может быть вычислено нормированное расстояние до главной последовательности, которое является результирующим показателем. Расстояние до главной последовательности – показатель, который отображает степень отдаленности компонента от данной последовательности. Чем значение ближе к нулю, тем качество компонента лучше.

Метрика устойчивости отражает количественную меру трудозатрат, которые необходимо произвести при изменении зависимых компонентов от анализируемого компонента.

Метрика Холстеда предоставляет информацию о сложности программного кода, его объеме и уровне понятности. Она основана на подсчете количества уникальных операторов и операндов в коде и вычислении различных показателей, таких как объем программы, сложность и количество ошибок, которые могут возникнуть. Метрика Холстеда позволяет оценить сложность и понятность кода, выявить его недостатки и потенциальные проблемы, связанные с его поддержкой и сопровождением [15].

4.2 Тестирование программных средств

4.2.1 Функциональное тестирование

Для проверки соответствия реального поведения разработанных ПС ожидаемому поведению необходимо провести функциональное тестирование.

Функциональные тесты представлены в таблице 6.

Таблица 6 – План функционального тестирования

Название теста	Входные данные	Ожидаемый результат	Тест пройден/не пройден
Добавление действия с условием в активное правило	Название - «Оповестить об уровне влажности» Условие - «Влажность<10%»	От условия активного правила построено новое действие с названием «Оповестить об уровне влажности»	Тест пройден (рисунок Г.1)
Удаление существующей связи и добавление связи события и действия без условия		Связь от условия к действию «Оповестить об уровне влажности» удалена, а связь от события к действию «Оповестить об уровне влажности» построена	Тест пройден (рисунок Г.2)
Удаление объекта метаданных		В обновленной структуре графа нет созданного ранее события «Оповестить об уровне влажности».	Тест пройден (рисунок Г.3)
Редактирование полей объекта метаданных	Название - «Включить увлажнитель №1»	Название действия «Включить увлажнитель» изменено на «Включить увлажнитель №1»	Тест пройден (рисунок Г.4)
Перемещение элементов по рабочей области в рамках структуры графа		Элементы графа изменили свое расположение в структуре графа.	Тест пройден (рисунок Г.5)
Отображение дополнительных данных об элементе графа		Отображен код элемента события.	Тест пройден (рисунок Г.6)
Проверка валидности полей добавления действия	Пустое поле	При попытке сохранения пустых полей, выведено сообщение с просьбой заполнить поля	Тест пройден (рисунок Г.7)

Тестирование разработанного приложения проводилось на примере системы, описанной в приложении В. Рисунки, отражающие результаты успешно пройденных тестов приведены в приложении Г.

4.2.2 Конфигурационное тестирование

Конфигурационное тестирование ПС поддержки графового представления метаданных было проведено с использованием нескольких наиболее

популярных веб-браузеров для проверки работоспособности приложения в разных исполняемых окружениях.

Тестирования осуществлялось путем запуска web-приложения в разных браузерах. План конфигурационного тестирования указан в таблице 7.

Результаты конфигурационного тестирования расположены в приложении Д.

Таблица 7 – План конфигурационного тестирования

Тестируемый браузер	Тестируемая ситуация	Ожидаемый результат	Фактический результат
Google Chrome (версия 114.0.5735.110)	Отображение страницы графового представления	Страница графового представления отображена	Фактический результат совпал с ожидаемым (рисунок Д.1)
	Отображение созданного элемента графа	Созданный объект метаданных отобразился в структуре графе	Фактический результат совпал с ожидаемым (рисунок Д.2)
	Сохранение и отображение редактируемых данных элемента графа	Редактируемые данные обновлены	Фактический результат совпал с ожидаемым (рисунок Д.3)
Microsoft Edge (версия 114.0.1823.37)	Отображение страницы графового представления	Страница графового представления отображена	Фактический результат совпал с ожидаемым (рисунок Д.4)
	Отображение созданного элемента графа	Созданный объект метаданных отобразился в структуре графе	Фактический результат совпал с ожидаемым (рисунок Д.5)
	Сохранение и отображение редактируемых данных элемента графа	Редактируемые данные обновлены	Фактический результат совпал с ожидаемым (рисунок Д.6)
Яндекс.Браузер (версия 23.5.1.721)	Отображение страницы графового представления	Страница графового представления отображена	Фактический результат совпал с ожидаемым (рисунок Д.7)
	Отображение созданного элемента графа	Созданный объект метаданных отобразился в структуре графе	Фактический результат совпал с ожидаемым (рисунок Д.8)

Продолжение таблицы 7

Тестируемый браузер	Тестируемая ситуация	Ожидаемый результат	Фактический результат
Яндекс.Браузер (версия 23.5.1.721)	Сохранение и отображение редактируемых данных элемента графа	Редактируемые данные обновлены	Фактический результат совпал с ожидаемым (рисунок Д.9)
Firefox Browser (версия 72.0.2)	Отображение страницы графового представления	Страница графового представления отображена	Фактический результат совпал с ожидаемым (рисунок Д.10)
	Отображение созданного элемента графа	Созданный объект метаданных отобразился в структуре графе	Фактический результат совпал с ожидаемым (рисунок Д.11)
	Сохранение и отображение редактируемых данных элемента графа	Редактируемые данные обновлены	Фактический результат совпал с ожидаемым (рисунок Д.12)
Опера (версия 85.0.4341.18)	Отображение страницы графового представления	Страница графового представления отображена	Фактический результат совпал с ожидаемым (рисунок Д.13)
	Отображение созданного элемента графа	Созданный объект метаданных отобразился в структуре графе	Фактический результат совпал с ожидаемым (рисунок Д.14)
	Сохранение и отображение редактируемых данных элемента графа	Редактируемые данные обновлены	Фактический результат совпал с ожидаемым (рисунок Д.15)

Программные средства графового представления метаданных корректно запустились и выдали ожидаемый результат во всех тестируемых web-браузерах. Результаты тестирования представлены в приложении Г.

4.2.3 Модульное тестирование

Модульное тестирование является важным этапом разработки программных средств. Оно позволяет проверить корректность работы отдельных модулей, функций и компонентов программы. Для проведения модульного тестирования программных средств графового представления, был использован

популярный фреймворк WebdriverIO, предоставляющий мощные инструменты для автоматизации тестирования веб-приложений.

Выбор фреймворка WebdriverIO для модульного тестирования программных средств графового представления обусловлен его широким функционалом и простотой в использовании. WebdriverIO предоставляет разнообразные методы и инструменты для написания и выполнения тестов, включая возможность эмуляции пользовательских действий, взаимодействия с элементами интерфейса, проверки ожидаемых результатов и обработки ошибок. Благодаря поддержке различных браузеров и платформ, фреймворк WebdriverIO обеспечивает широкий охват тестирования и гарантирует совместимость программных средств графового представления с различными окружениями. Использование фреймворка WebdriverIO для модульного тестирования программных средств графового представления позволяет обеспечить проверку и подтверждение корректности работы функциональности, взаимодействия с элементами графов и обработки данных.

Модульное (Unit) тестирование проведено на примере компонентов графа, получения активных правил, получения данных для редактора активных правил.

Все протестированные компоненты прошли модульное тестирование. Результаты тестирования компонента получения активных правил представлены на рисунке 23.

```
-----
[chrome 114.0.5735.106 mac os x #0-0] Running: chrome (v114.0.5735.106) on mac os x
[chrome 114.0.5735.106 mac os x #0-0] Session ID: 3dd8c2792e840065776403033aafd399
[chrome 114.0.5735.106 mac os x #0-0]
[chrome 114.0.5735.106 mac os x #0-0] » /tests/e2e/activerule.e2e.js
[chrome 114.0.5735.106 mac os x #0-0] Active rules page test
[chrome 114.0.5735.106 mac os x #0-0]      ✓ login
[chrome 114.0.5735.106 mac os x #0-0]      ✓ get subsystems
[chrome 114.0.5735.106 mac os x #0-0]      ✓ get active rules
[chrome 114.0.5735.106 mac os x #0-0]
[chrome 114.0.5735.106 mac os x #0-0] 3 passing (10.8s)
```

Рисунок 23 - Результат модульного тестирования компонента получения
активных правил

Результаты тестирования остальных модулей представлены в приложении Е. Исходный код модульных тестов приведен в приложении Ж.

4.3 Оценка качества программного кода

4.3.1 Метрики устойчивости и абстрактности

Для разработанного web-приложения были рассчитаны соотношения между абстрактностью и устойчивостью. Вычисление рассматриваемых метрик производилось исходя из пакетов, в которые были организованы программные классы приложения. Всего имеется 10 основных модулей и один встроенный от фреймворка: App, EditorActiveRulePage, EditorActiveRuleService, EditorActiveRuleStore, GraphComponent, EditorRulePerformanceSelect, RightPanel, AddActionModal, AddEdgeModal, EditNodeModal.

Для расчёта устойчивости компонентов используется следующая формула:

$$I \text{ (Устойчивость)} = \frac{Ce}{(Ca + Ce)},$$

где I – показатель устойчивости компонента, которая изменяется в диапазоне от $[0, 1]$ (если $I=0$, то компонент максимально устойчив, если $I=1$, то компонент максимально неустойчив);

Ca – количество входящих связей к классам внутри компонента от классов вне данного компонента;

Ce – количество исходящих связей от классов внутри данного компонента к классам вне компонента.

Была рассчитана величина устойчивости для каждого компонента системы:

$$I_1 = \frac{0}{1+0} = 0;$$

$$I_2 = \frac{1}{9+1} = 0,1;$$

$$I_3 = \frac{0}{1+0} = 0;$$

$$I_4 = \frac{1}{1+1} = 0,5;$$

$$I_5 = \frac{8}{2+8} = 0,8;$$

$$I_6 = \frac{1}{2+1} = 0,33;$$

$$I_7 = \frac{3}{4+3} = 0,43;$$

$$I_8 = \frac{2}{2+2} = 0,5;$$

$$I_9 = \frac{2}{2+2} = 0,5.$$

$$I_{10} = \frac{2}{2+2} = 0,5.$$

Для расчета абстрактности компонентов использовалась формула, приведенная ниже:

$$A \text{ (Абстрактность)} = \frac{Na}{Nc},$$

где A – показатель абстрактности компонента, которая изменяется в диапазоне от $[0, 1]$ (если $A=0$, то компонент не имеет абстрактных классов, если $A=1$, то компонент состоит только из абстрактных классов);

Nc – количество классов в компоненте;

Na – количество абстрактных классов в компоненте;

Данная метрика также была рассчитана для каждого компонента сервиса:

$$A_1 = \frac{1}{5} = 0,2;$$

$$A_2 = \frac{2}{6} = 0,33;$$

$$A_3 = \frac{0}{1} = 0;$$

$$A_4 = \frac{0}{1} = 0;$$

$$A_5 = \frac{1}{1} = 1;$$

$$A_6 = \frac{1}{1} = 1;$$

$$A_7 = \frac{1}{2} = 0,5;$$

$$A_8 = \frac{1}{4} = 0,25;$$

$$A_9 = \frac{1}{4} = 0,25;$$

$$A_{10} = \frac{1}{4} = 0,25.$$

На основе вычисленных метрик абстрактности и устойчивости для каждого модуля приложения было найдено нормированное расстояние от главной последовательности по формуле:

$$D' = |A + I - 1|,$$

где A – абстрактность;

I – устойчивость.

Был проведен расчёт данной метрики для каждого компонента:

$$D'_1 = |0,2 + 0 - 1| = 0,8;$$

$$D'_2 = |0,33 + 0,1 - 1| = 0,57;$$

$$D'_3 = |0 + 0 - 1| = 1;$$

$$D'_4 = |0 + 0,5 - 1| = 0,5;$$

$$D'_5 = |1 + 0,8 - 1| = 0,8;$$

$$D'_6 = |1 + 0,33 - 1| = 0,33;$$

$$D'_7 = |0,5 + 0,43 - 1| = 0,07;$$

$$D'_8 = |0,25 + 0,5 - 1| = 0,25;$$

$$D'_9 = |0,25 + 0,5 - 1| = 0,25;$$

$$D'_{10} = |0,25 + 0,5 - 1| = 0,25.$$

Данный показатель возможно представить при помощи диаграммы, изображенной на рисунке 24. В данной координатной сетке располагаются зона неприятностей (где компоненты слишком устойчивы и не абстрактны), зона бесполезности (компоненты в данной области в большой степени неустойчивы и абстрактны), а также главная последовательность, к которой должны стремиться компоненты по рассматриваемым показателям.

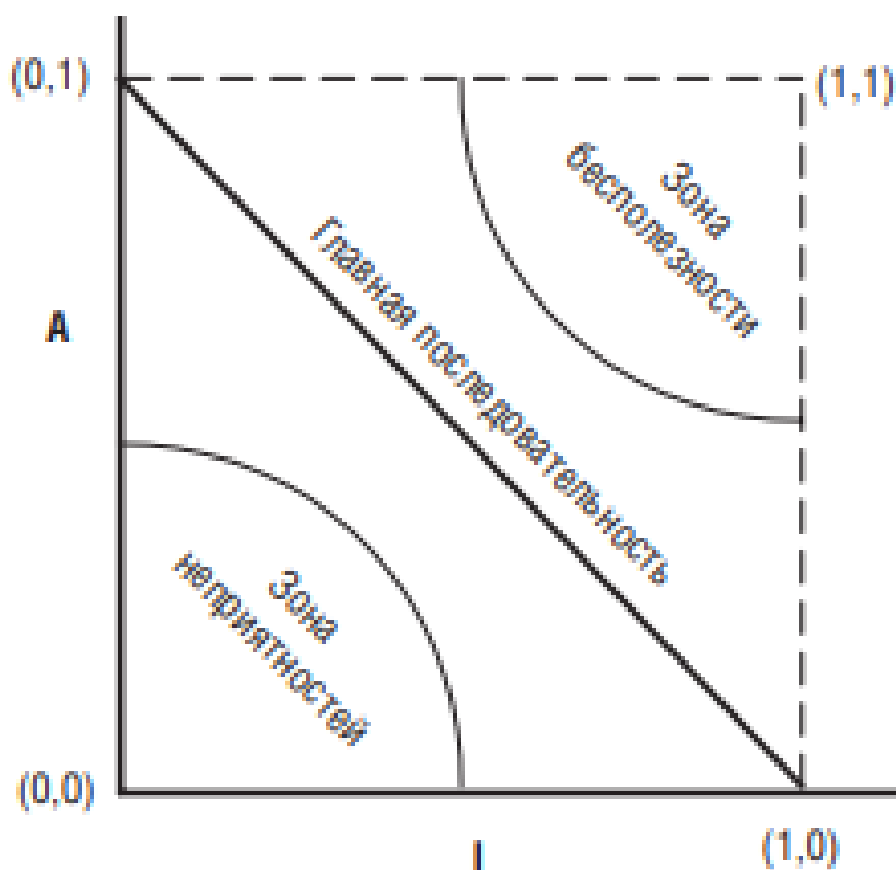


Рисунок 24 – Диаграмма возможных значений главной последовательности

По рассчитанным значениям для компонентов приложения был построен аналогичный график, изображенный на рисунке 25.

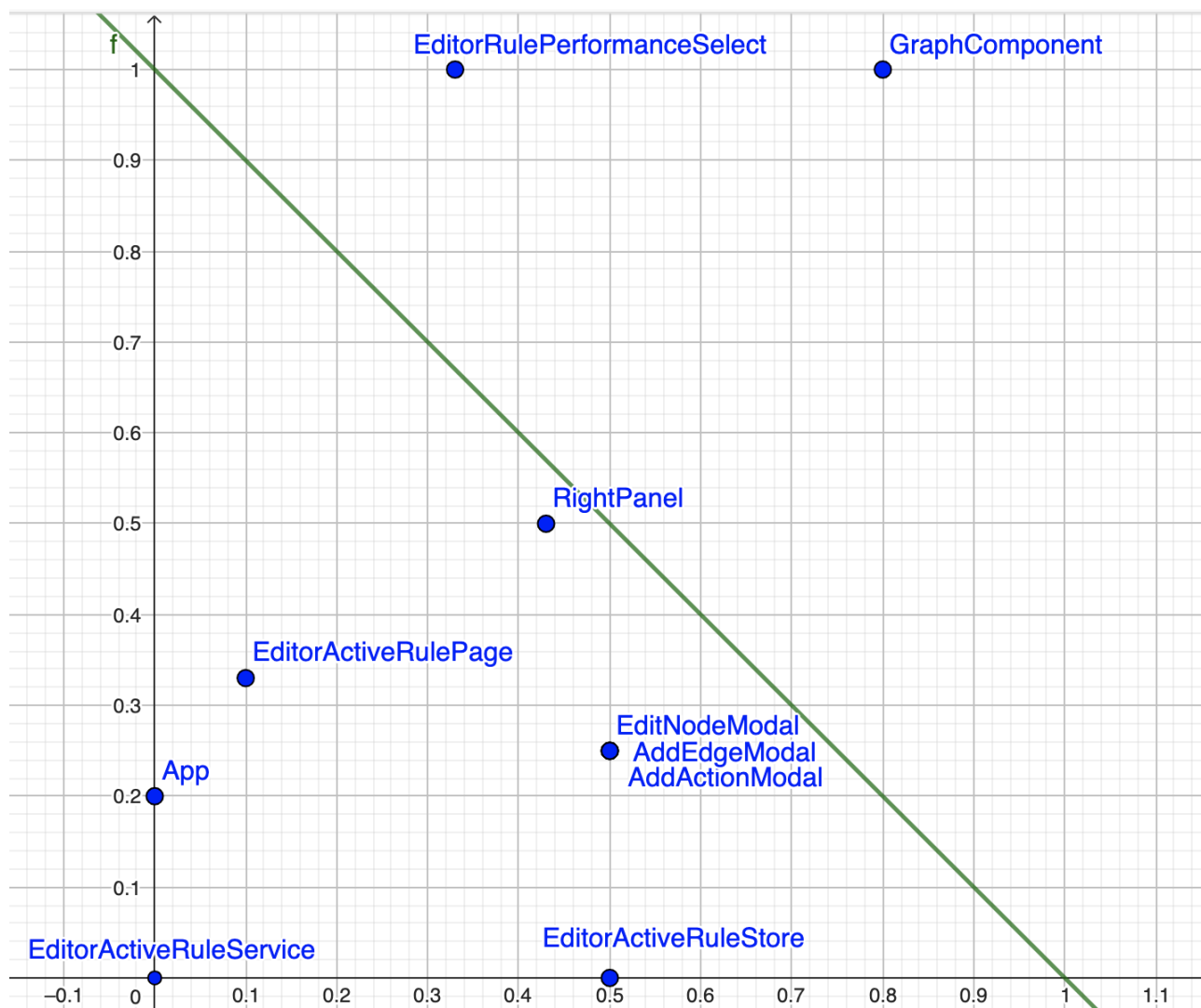


Рисунок 25 – Метрики кода на графе значений главной последовательности

Компоненты App, EditorRuleService и GraphComponent располагаются далеко от главной последовательности и являются достаточно устойчивыми. Это объясняется тем, что данные компоненты реализуют базовые функции сервиса и не зависят от других блоков приложения.

Остальные компоненты лежат достаточно близко к главной последовательности. Это значит, что отношение абстрактности и устойчивости компонент является верным.

4.3.2 Метрика Холстеда

Для ПС поддержки графового представления метаданных были подсчитаны метрики Холстеда. Метрики Холстеда относятся к метрикам,

вычисляемым на основании анализа числа строк и синтаксических элементов исходного кода программы.

Для расчёта метрик Холстеда были взяты функции, выполняющие с основным функционалом управления в графовом представлении: добавление узлов и связей, редактирование и удаление. Код данного модуля представлен в листинге 3.

Листинг 3 – Код модального окна для добавления активного правила

```
import {updateActiveRule} from '../..store/action-creators/activeRules';
import {store} from '../..portal-react-angular-react-app';

const currentSystemCode =
localStorage.getItem('currentSystemCode');
const currentActiveRuleObject =
JSON.parse(localStorage.getItem('currentActiveRuleObject') ||
'{}');

export const onRemoveNodeFromAR = async (currentNodeId: string) =>
{
    const actions = JSON.parse(currentActiveRuleObject.action);
    const deletedNode =
actions.data.indexOf(actions.data.find((act: { code: string }) =>
act.code === currentNodeId));

    actions.edges = actions.edges.filter((edge: { from: string,
to: string }) =>
    edge.from !== currentNodeId && edge.to !== currentNodeId);
    actions.data.splice(deletedNode, 1);
    const requestBody = {
        description: currentActiveRuleObject.description,
        condition: currentActiveRuleObject.condition,
        action: JSON.stringify({data: [...actions.data], edges:
[...actions.edges]}),
        code: currentActiveRuleObject.code
    };

    currentSystemCode && await
store.dispatch(updateActiveRule(requestBody, currentSystemCode));
};

export const onRemoveEdgeFromAR = async (currentEdge: { from:
string, to: string }) => {
    const actions = JSON.parse(currentActiveRuleObject.action);
    const deletedEdge =
actions.edges.indexOf(actions.edges.filter((ed: { from: string })
```

```

=> ed.from === currentEdge.from)
    .find((ed: { to: string }) => ed.to === currentEdge.to));
actions.edges.splice(deletedEdge, 1);
const requestBody = {
    description: currentActiveRuleObject.description,
    condition: currentActiveRuleObject.condition,
    action: JSON.stringify({data: [...actions.data], edges:
[...actions.edges]}),
    code: currentActiveRuleObject.code
};
currentSystemCode && await
store.dispatch(updateActiveRule(requestBody, currentSystemCode));
};

export const onEditNodeFromAR = async (currentNodeId: string,
newLabel: string) => {
    const actions = JSON.parse(currentActiveRuleObject.action);
    const editedNode = actions.data.find((act: {code: string}) =>
act.code === currentNodeId);
    const newData = actions.data.map((d: {category: string,
description: string, code: string}) => {
        if (d.code === editedNode?.code) {
            return {
                ...editedNode,
                description: `${newLabel}`
            };
        }
        return {...d};
    });
    const requestBody = {
        description: currentActiveRuleObject.description,
        condition: currentActiveRuleObject.condition,
        action: JSON.stringify({
            data: [...newData],
            edges: [...actions.edges]
        }),
        code: currentActiveRuleObject.code
    };
    currentSystemCode && await
store.dispatch(updateActiveRule(requestBody, currentSystemCode));
};

export const onAddEdgeInAR = async (data: {condition: string,
event: string, action: string}) => {
    const actions = JSON.parse(currentActiveRuleObject.action);
    if (currentActiveRuleObject.event[0].association.typeBind ===
'Event to Rule') {
        if (data.condition) {
            actions.edges.push({from: data.event, to:
data.condition});

```

```

        actions.edges.push({from: data.condition, to:
data.action});
    } else {
        actions.edges.push({from: data.event, to:
data.action});
    }
    } else {
        if (data.condition) {
            actions.edges.push({from: data.condition, to:
data.event});
            actions.edges.push({from: data.condition, to:
data.action});
        } else {
            actions.edges.push({from: data.action, to:
data.event});
        }
    }
    }
    const requestBody = {
        description: currentActiveRuleObject.description,
        condition: currentActiveRuleObject.condition,
        action: JSON.stringify({data: actions.data, edges:
actions.edges}),
        code: currentActiveRuleObject.code
    };

    currentSystemCode && await
store.dispatch(updateActiveRule(requestBody, currentSystemCode));
};

```

Количество используемых операндов и операторов, а также их общее число в программе приведены в таблицах 8 и 9 соответственно.

Таблица 8 – Операторы

Операторы	Число вхождений
:	62
?	1
=	27
()	49
.	129
=>	10
{ }	40
,	36
;	35
‘ ’	6
filter	2
find	3

Продолжение таблицы 8

Операторы	Число вхождений
indexOf	2
JSON.parse	5
JSON.stringify	4
dispatch	2
map	1
push	6
return	2
n1 = 19	N1 = 422

Таблица 9 – Операнды

Операнды	Число вхождений
actions	27
deletedNode	2
requestBody	8
deletedEdge	2
editedNode	3
newData	2
currentSystemCode	10
currentActiveRuleObject	19
n2 = 8	N2 = 73

Длина программы рассчитывается по формуле:

$$N = N1 + N2 = 422 + 73 = 495.$$

Словарь программы рассчитывается по формуле:

$$n = n1 + n2 = 19 + 8 = 27.$$

Объем программы рассчитывается по формуле:

$$H_V = N * \log_2(n) = 495 * 4,755 = 2353,725.$$

Потенциальный объем программы:

$$V^* = (2 + n) * \log_2(2 + n) = 5 * \log_2(5) = 11,61.$$

Теоретический уровень программы рассчитывается по формуле:

$$L^* = V^* / H_V = 11,61 / 2353,725 = 0.005.$$

Сложность программы рассчитывается по формуле:

$$HD = (n1 / 2) * (N2 / n2) = (19 / 2) * (73 / 8) = 86,6875.$$

Полученные показатели объема и сложности программы удовлетворяют требованиям по метрикам программного кода.

4.3.3 Оценка качества пользовательского интерфейса

Оценка эффективности построенного пользовательского интерфейса оценивалась при помощи метода Keystroke-level model (KLM). При использовании данного метода задача, поставленная перед пользователем, разбивается на составляющие действия, для каждого из которых известно усредненное время исполнения пользователем. В данные операции включаются:

- а) нажатие на клавишу клавиатуры (K): 0,28 сек.;
- б) нажатие на кнопку мыши (M): 0,1 сек.;
- в) перемещение курсора мыши (П): 1,1 сек.;
- г) перенос руки от мыши к клавиатуре и обратно (Р): 0,4 сек.;
- д) продолжительность выбора действия (Д): 1,2 сек.;
- е) время реакции системы (В): от 0,1 сек.

Проведем тестирование рассмотренным методом функцию добавления связи между узлами со следующими данными:

- событие – «Измерение влажности»;
- условие – «Увлажнитель выключен»;
- действие – «Нажатие на кнопку off, выключение».

В таблице 10 приведен список составляющих действий, а также длительность каждой проведенной операции.

Таблица 10 – План тестирование эффективности пользовательского интерфейса модального окна добавления пользователя

№	Тип действия	Продолжительность	Комментарий
1	Д	1,2	Анализ задачи и создание алгоритма ее решения
2	П	1,1	Перемещение курсора мыши к кнопке «Добавить связь»
3	М	0,1	Нажатие кнопки мыши
4	В	0,1	Время реакции системы. Открывается всплывающее окно добавления связи

Продолжение таблицы 10

№	Тип действия	Продолжительность	Комментарий
5	П	1,1	Перемещение курсора к полю выбора события
6	М	0,1	Нажатие кнопки мыши. Раскрытие списка доступных событий
7	П	1,1	Перемещение курсора к нужному событию
8	М	0,1	Нажатие кнопки мыши. Ввод события произошел
9	П	1,1	Перемещение курсора к полю выбора условия
10	М	0,1	Нажатие кнопки мыши. Раскрытие списка доступных условий
11	П	1,1	Перемещение курсора к нужному условию
12	М	0,1	Нажатие кнопки мыши. Ввод условия произошел
13	П	1,1	Перемещение курсора к полю выбора действия
14	М	0,1	Нажатие кнопки мыши. Раскрытие списка доступных действий
15	П	1,1	Перемещение курсора к нужному действию
16	М	0,1	Нажатие кнопки мыши. Ввод действия произошел
17	П	1,1	Перемещение курсора к кнопке «Сохранить»
18	М	0,1	Нажатие кнопки мыши
19	В	0,3	Время реакции системы. Отправка данных на сервер и получение ответа. Последующее закрытие диалогового окна

В итоге для реализации функции добавления связи между узлами пользователь произвел 19 действий, затратив на данные операции суммарно 11,2 секунд. Объем затраченного времени на столь комплексную задачу невелик, что показывает хороший уровень построения юзабилити в разрабатываемых инструментальных средствах.

Заключение

По итогам выполнения выпускной квалификационной работы были разработаны программные средства поддержки графового представления для сервиса потоковой обработки событий.

В начале работы был проведен анализ предметной области и выделены функциональные возможности программных средств поддержки графового представления для сервиса потоковой обработки событий. Далее были описаны архитектура ПС, сценарии работы, механизмы управления метаданными, их жизненный цикл и состав. Затем проведено планирование и оценка бюджета проекта в MS Project. Затраты на проект составили 466370 рублей, а время разработки 46 рабочих дней.

Также до этапа разработки был проведен этап проектирования программных средств, который включал в себя проектирование архитектуры и разработку функциональных требований. Был обоснован выбор технологий и средств разработки, а также выделены требования к аппаратно-программного обеспечению.

В качестве основных этапов разработки можно выделить такие этапы как: разработка средств отображения графового представления, разработка пользовательского интерфейса управления метаданными в графовом представлении, разработка функционала создания элементов активного правила, а также разработка средств редактирования элементов графа.

Разработка программных средств была произведена с использованием операционной системы Windows и браузера Google Chrome в среде разработки Webstorm на языке программирования JavaScript с использованием фреймворка React JS и библиотеки VIS.js.

В завершении был проведен контроль качества разработанных программных средств. Было проведено функциональное, конфигурационное и модульное тестирования, а также были подсчитаны метрики устойчивости и

абстрактности, метрики Холстеда и проведена оценка качества пользовательского интерфейса.

Все этапы сопровождались диаграммами и пояснительным текстом к ним. Разработанное программное обеспечение отвечает всем требованиям технического задания.

Список используемых источников

1. Event-driven application // Documentation [Электронный ресурс]. URL: <https://www.techtarget.com/searchitoperations/definition/event-driven-application> (дата обращения 13.03.2023).
2. Шибанов, С. В. Интегрированная модель активных правил / С. В. Шибанов, А. А. Скоробогатько, Э. В. Лысенко // Математическое и программное обеспечение систем в промышленной и социальной сферах: междунар. сб. науч. трудов – Магнитогорск : Изд-во Магнитогорск. гос. техн. ун-та им. Г.И. Носова, 2011. – С. 41 – 46.
3. Шибанов, С. В. Формализация ЕСА и SECA-моделей активных правил / С. В. Шибанов, Э. В. Лысенко, А. А. Скоробогатько // Прикладная математика, управление и информатика: сборник трудов Междунар. молодеж. конф.: в 2 т. – Белгород: ИД «Белгород», 2012 – Т. 1 – С. 319 – 322.
4. Шибанов, С. В. Сервис потоковой обработки событий и исполнения активных правил / С. В. Шибанов, Я. С. Шлепнёв // Математическое моделирование и суперкомпьютерные технологии. – Нижний Новгород: Изд-во Нижегородского госуниверситета, 2021. - С. 403-407.
5. Шибанов, С. В. Архитектура метаданных сервиса потоковой обработки событий и исполнения активных правил / С. В. Шибанов, А. С. Гусаров, Я. С. Шлепнёв // Математическое и программное обеспечение систем в промышленной и социальной сферах: междунар. сб. науч. трудов – Магнитогорск : Изд-во Магнитогорск. гос. техн. ун-та им. Г.И. Носова, 2011. – С. 66 – 76.
6. Carlson J.L. Redis in Action / Josiah L. Carlson. – MANNING Shelter Island. - 2013. – 322 с.
7. Q. Jiang. Towards an Integrated Model for Event and Stream Processing / Q. Jiang, R. Adaikkalavan, S. Chakravarthy // Technical Report CSE – Texas : Изд-во CSE, 2004 – 35 с.

8. Прошкина, Е. Н. Язык моделирования информационных процессов UML 2.0 / Е. Н. Прошкина, И. Ю. Балашова – Пенза : Изд-во ПГУ, 2018. – 112 с.

9. Шибанов, С.В. Мета модель сервиса потоковой обработки событий / С. В. Шибанов, Я. С. Шлепнёв, А. С. Гусаров // XIV Междунар. науч.-техн. конф. с элементами науч. шк. и конкурсом науч.-исслед. работ для студентов, аспирантов и молодых ученых. – Пенза : Изд-во ПГУ, 2022. – С. 240-244.

10. Обучение по Project [Электронный ресурс]. URL: <https://support.microsoft.com/ru/office/%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5-%D0%BF%D0%BE-project-63f1db89-1ee0-4113-bedc-03fc5ec6223a> (дата обращения 16.04.2023).

11. Getting Started with single-spa // Single SPA [Электронный ресурс]. URL: <https://single-spa.js.org/docs/getting-started-overview> (дата обращения 17.05.2023).

12. Vis.js Documentation [Электронный ресурс]. URL: <https://visjs.github.io/vis-network/docs/network/> (дата обращения 17.04.2023).

13. Управление качеством программного обеспечения [Электронный ресурс]. URL: <https://coderlessons.com/tutorials/kachestvo-programmnogo-obespecheniia/izuchite-upravlenie-kachestvom-programmnogo-obespecheniia/upravlenie-kachestvom-programmnogo-obespecheniia-kratkoe-rukovodstvo> (дата обращения 05.05.2023).

14. Виды и типы тестирования // Подробный разбор [Электронный ресурс]. URL: <https://testengineer.ru/vidy-tipy-testirovaniya/> (дата обращения 17.05.2023).

15. Метрики Холстеда [Электронный ресурс]. URL: https://studwood.net/2361020/informatika/metriki_holsteda (дата обращения 27.05.2023).

Приложение А
ГЛОССАРИЙ
(справочное)

Event Stream Processing (ESP) – набор технологий, предназначенных для построения информационных систем обработки событий, которые включают в себя визуализацию событий, их хранение, связующее программное обеспечение и язык программирования обработки событий.

ЖЦ – (жизненный цикл) охватывает процесс работы с метаданными с момента их формирования и до момента их деактуализации.

REST API (Representational State Transfer) – архитектурный стиль взаимодействия компонентов распределенного приложения в сети.

Программное средство (ПС) – объект, состоящий из программ, процедур, правил, а также, если предусмотрено, сопутствующих им документации и данных, относящихся к функционированию системы обработки информации.

Keystroke-level model (KLM) – версия CMN модели, в которой используются только операторы, нет методов или правил выбора. Аналитик перечисляет нажатия на клавиши, движения мышью, которые должен произвести пользователь, чтобы решить задачу, а затем использует несколько простых эвристических правил, чтобы расставить операторы M.

Web-браузер – прикладное программное обеспечение для поиска и просмотра web-страниц, содержания web-документов, компьютерных файлов, а также управления web-приложениями.

Приложение Б
ЛИСТИНГ МОДЕЛЕЙ АКТИВНЫХ ПРАВИЛ
(рекомендуемое)

Листинг Б.1 – Файл ActiveRulesPage/index.tsx

```
import React, {useEffect} from 'react';
import MainTemplate from '../common/MainTemplate';
import RuleTags from './components/Performance/RuleTags';
import {useAppDispatch, useTypedSelector} from
'../hooks/useTypedSelector';
import {useAction} from '../hooks/useAction';
import {RulesPerformance} from
'../store/types/activeRulesTypes';
import RuleTable from './components/Performance/RuleTable';
import RightPanel from './components/RightPanel';

const ActiveRulesPage = () => {
  const {currentSystem, systemPagesWay} = useTypedSelector(state
=> state.systemsValues);
  const {isLoading, error, currentPerformance} =
useTypedSelector(state => state.activeRulesValues);
  const dispatch = useAppDispatch();
  const {setSystemPagesWay, fetchActiveRuleBySystemCode,
fetchEventsBySystemCode} = useAction();
  const currentSystemCode =
localStorage.getItem('currentSystemCode');

  useEffect(() => {
    setSystemPagesWay([...systemPagesWay, {
      name: ` / ${currentSystem.name}`,
      code: currentSystem.code,
      systems: currentSystem.children
    }]);

    dispatch(() =>
fetchActiveRuleBySystemCode(currentSystemCode ||
currentSystem.code));
    dispatch(() => fetchEventsBySystemCode(currentSystemCode
|| currentSystem.code));
  }, [currentSystem]);

  if (isLoading) {
    return <div>Идёт загрузка активных правил...</div>;
  }

  if (error) {
    return <div>{error}</div>;
  }

  const getCurrentPerformance = () => {
    switch (currentPerformance) {
      case RulesPerformance.tag:
        return <MainTemplate blocks={<RightPanel/>}
id="active-rules-container">
```

```

        <RuleTags />
      </MainTemplate>;
      case RulesPerformance.table:
        return <MainTemplate blocks={<RightPanel/>}
id="active-rules-container">
        <RuleTable />
      </MainTemplate>;
    }
  };

  return getCurrentPerformance();
};

export default ActiveRulesPage;

```

Листинг Б.2 – Файл EditorActiveRulesPage/index.tsx

```

import React, {useEffect} from 'react';
import MainTemplate from '../../common/MainTemplate';
import {useTypedSelector} from '../../hooks/useTypedSelector';
import RuleGraph from
'./components/Performance/EditorRuleGraph/index';
import RightPanel from '../../ActiveRulesPage/components/RightPanel';
import {EditorRulesPerformance} from
'../../store/types/editorActiveRulesTypes';
import EditorRulePerformanceSelect from
'./components/EditorRulePerformanceSelect';
import {useAction} from '../../hooks/useAction';
import EditorRuleList from
'./components/Performance/EditorRuleList';
import {setCurrentEditorRulePerformance} from '../../store/action-
creators/editorActiveRules';
import commonStyles from '../../common/styles/styles.module.scss';

const EditorActiveRulesPage = ({isLang}: {isLang: boolean}) => {
  const {currentEditorRulePerformance} = useTypedSelector(state
=> state.editorActiveRulesValues);
  const {currentActiveRule, activeRules} =
useTypedSelector(state => state.activeRulesValues);
  const {systemPagesWay} = useTypedSelector(state =>
state.systemsValues);
  const {setSystemPagesWay} = useAction();

  useEffect(() => {
    if(currentActiveRule) {
      setSystemPagesWay([...systemPagesWay, {
        name: ` / ${currentActiveRule.description}`,
        code: currentActiveRule.code,
        systems: activeRules
      }]);
    }
  }, []);

```

```

    useEffect(() => {
        isLang &&
        setCurrentEditorRulePerformance(EditorRulesPerformance.lang);
    }, [isLang]);

    const getCurrentEditorRulePerformance = () => {
        switch (currentEditorRulePerformance) {
            case EditorRulesPerformance.list:
                return <EditorRuleList />;
            case EditorRulesPerformance.lang:
                return <MainTemplate blocks={<RightPanel/>}
id="editor-active-rules-container">
                    <div className={commonStyles.toolbar}>
                        <EditorRulePerformanceSelect/>
                    </div>
                </MainTemplate>;
            case EditorRulesPerformance.graph:
                return <RuleGraph />;
        }
    };

    return getCurrentEditorRulePerformance();
};

export default EditorActiveRulesPage;

```

Листинг Б.3 – Файл EditorActiveRulesPage/helpers.tsx

```

import {updateActiveRule} from '../..//store/action-
creators/activeRules';
import {store} from '../..//portal-react-angular-react-app';

export const onRemoveNodeFromAR = async (currentNodeId: string) =>
{
    const currentSystemCode =
localStorage.getItem('currentSystemCode');
    const currentActiveRuleObject =
JSON.parse(localStorage.getItem('currentActiveRuleObject') ||
'{}');
    const actions = JSON.parse(currentActiveRuleObject.action);
    const deletedNode =
actions.data.indexOf(actions.data.find((act: { code: string }) =>
act.code === currentNodeId));

    actions.edges = actions.edges.filter((edge: { from: string,
to: string }) =>
    edge.from !== currentNodeId && edge.to !== currentNodeId);
    if(deletedNode !== -1) {
        actions.data.splice(deletedNode, 1);
        const requestBody = {
            description: currentActiveRuleObject.description,

```

```

        condition: currentActiveRuleObject.condition,
        action: JSON.stringify({data: [...actions.data],
edges: [...actions.edges]}),
        code: currentActiveRuleObject.code
    };

    currentSystemCode && await
store.dispatch(updateActiveRule(requestBody, currentSystemCode));
    }
};

export const onRemoveEdgeFromAR = async (currentEdge: { from:
string, to: string }) => {
    const currentSystemCode =
localStorage.getItem('currentSystemCode');
    const currentActiveRuleObject =
JSON.parse(localStorage.getItem('currentActiveRuleObject') ||
'{}');
    const actions = JSON.parse(currentActiveRuleObject.action);
    const deletedEdge =
actions.edges.indexOf(actions.edges.filter((ed: { from: string })
=> ed.from === currentEdge.from)
    .find((ed: { to: string }) => ed.to === currentEdge.to));

    if(deletedEdge !== -1) {
        actions.edges.splice(deletedEdge, 1);
        const requestBody = {
            description: currentActiveRuleObject.description,
            condition: currentActiveRuleObject.condition,
            action: JSON.stringify({data: [...actions.data],
edges: [...actions.edges]}),
            code: currentActiveRuleObject.code
        };

        currentSystemCode && await
store.dispatch(updateActiveRule(requestBody, currentSystemCode));
    }
};

export const onEditNodeFromAR = async (currentNodeId: string,
newLabel: string) => {
    const currentSystemCode =
localStorage.getItem('currentSystemCode');
    const currentActiveRuleObject =
JSON.parse(localStorage.getItem('currentActiveRuleObject') ||
'{}');
    const actions = JSON.parse(currentActiveRuleObject.action);
    const editedNode = actions.data.find((act: {code: string}) =>
act.code === currentNodeId);
    const newData = actions.data.map((d: {category: string,
description: string, code: string}) => {
        if (d.code === editedNode?.code) {

```

```

        return {
            ...editedNode,
            description: `${newLabel}`
        };
    }
    return {...d};
});

const requestBody = {
    description: currentActiveRuleObject.description,
    condition: currentActiveRuleObject.condition,
    action: JSON.stringify({
        data: [...newData],
        edges: [...actions.edges]
    }),
    code: currentActiveRuleObject.code
};

currentSystemCode && await
store.dispatch(updateActiveRule(requestBody, currentSystemCode));
};

export const onAddEdgeInAR = async (data: {condition: string,
event: string, action: string}) => {
    const currentSystemCode =
localStorage.getItem('currentSystemCode');
    const currentActiveRuleObject =
JSON.parse(localStorage.getItem('currentActiveRuleObject') ||
'{}');
    const actions = JSON.parse(currentActiveRuleObject.action ??
'');
    if (currentActiveRuleObject.event[0].association.typeBind ===
'Event to Rule') {
        if (data.condition) {
            actions.edges.push({from: data.event, to:
data.condition});
            actions.edges.push({from: data.condition, to:
data.action});
        } else {
            actions.edges.push({from: data.event, to:
data.action});
        }
    } else {
        if (data.condition) {
            actions.edges.push({from: data.condition, to:
data.event});
            actions.edges.push({from: data.condition, to:
data.action});
        } else {
            actions.edges.push({from: data.action, to:
data.event});
        }
    }
}

```

```

    }

    const requestBody = {
      description: currentActiveRuleObject.description,
      condition: currentActiveRuleObject.condition,
      action: JSON.stringify({data: [...actions.data], edges:
[...actions.edges]}),
      code: currentActiveRuleObject.code
    };

    currentSystemCode && await
store.dispatch(updateActiveRule(requestBody, currentSystemCode));
};

```

Листинг Б.4 – Файл EditorRulesPerformanceSelect/index.tsx

```

import React from 'react';
import {Select} from 'antd';
import {useAction} from '../../../hooks/useAction';
import {useTypedSelector} from
'../../../hooks/useTypedSelector';
import {EditorRulesPerformance} from
'../../../store/types/editorActiveRulesTypes';
import {useNavigate} from 'react-router-dom';

const EditorRulePerformanceSelect = () => {
  const {setCurrentEditorRulePerformance} = useAction();
  const {currentEditorRulePerformance} = useTypedSelector(store
=> store.editorActiveRulesValues);
  const navigate = useNavigate();

  return <>
    <Select
      placeholder="Представление"
      size="small"
      onChange={value => {
        setCurrentEditorRulePerformance(value);
        if(value === EditorRulesPerformance.lang){
          navigate('/active-rules-script');
        } else {
          navigate('/editor-active-rules');
        }
      }}
      dropdownStyle={{zIndex: 1102}}
      value={currentEditorRulePerformance}
      options={[
        {value: 'graph', label: 'Графовое'},
        {value: 'list', label: 'Списочное'},
        {value: 'lang', label: 'Языковое'}
      ]}
    />
  </>;
};

```

```
export default EditorRulePerformanceSelect;
```

Листинг Б.5 – Файл EditorRulesGraph/index.js

```
import React, {useEffect, useMemo, useState} from 'react';
import commonStyles from
'../../../../../common/styles/styles.module.scss';
import GraphComponent from '../components/GraphComponent';
import EditorRulePerformanceSelect from
'../../EditorRulePerformanceSelect';
import {getColor, getShape} from '../helpers/handlers';
import EditNodeModal from '../components/EditNodeModal';
import AddEdgeModal from '../components/AddEdgeModal';
import Icon from '../../../../../common/components/Icon';
import variables from '../../../../../variables.module.scss';
import MainTemplate from '../../../../../common/MainTemplate';
import Header from '../../../../../common/components/Header';
import Managing from
'../../../../../ActiveRulesPage/components/RightPanel/Managing';
import {useTypedSelector} from
'../../../../../hooks/useTypedSelector';
import AddActionModal from '../components/AddActionModal';
import {useAction} from '../../../../../hooks/useAction';
import classNames from 'classnames/bind';
import styles from './styles.module.scss';
import {onAddEdgeInAR, onEditNodeFromAR, onRemoveEdgeFromAR,
onRemoveNodeFromAR} from '../../../../../helpers';

const cn = classNames.bind(styles);

const EditorRuleGraph = () => {
  const [currentNodeId, setCurrentNodeId] = useState();
  const [currentNode, setCurrentNode] = useState();
  const [currentEdgeId, setCurrentEdgeId] = useState();
  const [currentEdge, setCurrentEdge] = useState();
  const [currentPointer, setCurrentPointer] = useState({x: 0, y:
0});
  const [isEditGraphModalVisible, setIsEditGraphModalVisible] =
useState(false);
  const [isAddEdgeModalVisible, setIsAddEdgeModalVisible] =
useState(false);
  const [isAddActionModalVisible, setIsActionModalVisible] =
useState(false);
  const {currentActiveRule, activeRules} =
useTypedSelector(state => state.activeRulesValues);
  const {setCurrentActiveRule} = useAction();
  const currentActiveRuleObject =
JSON.parse(localStorage.getItem('currentActiveRuleObject'));
  const currentSystemCode =
localStorage.getItem('currentSystemCode');

  useEffect(() => {
```

```

        currentActiveRuleObject && activeRules.length > 0 &&
        setCurrentActiveRule(activeRules.find(ar => ar.code ===
currentActiveRuleObject.code));
    }, [activeRules]);

    const [state, setState] = useState({
        graph: {
            nodes: [],
            edges: []
        },
        events: {
            select: function ({nodes, edges, pointer: {canvas}}) {
                setCurrentNodeId(undefined);
                setCurrentEdgeId(undefined);
                if (!nodes[0]) {
                    setCurrentEdgeId(edges[0]);
                    setCurrentPointer({x: canvas.x, y: canvas.y});
                } else {
                    setCurrentNodeId(nodes[0]);
                    setCurrentPointer({x: canvas.x, y: canvas.y});
                }
            },
        }
    });

    const eventNodes = useMemo(() =>
currentActiveRuleObject?.event.map(ev => {
        return {
            id: ev.code,
            title: `Code ${ev.code}`,
            label: `${ev.description} -
${ev.categoryEvent[0].toUpperCase()}E`,
            shape: getShape(`${ev.categoryEvent} event`),
            color: {background: getColor(`${ev.categoryEvent}
event`)},
            x: Math.random() * 600 - 300,
            y: Math.random() * 600 - 300
        };
    })), [currentActiveRuleObject]);

    const conditions =
JSON.parse(currentActiveRuleObject?.condition);
    const actions = JSON.parse(currentActiveRuleObject?.action);

    const eventEdges = useMemo(() =>
currentActiveRuleObject?.event.map(ev => {
        if (ev.association.typeBind === 'Event to Rule') {
            return {
                from: ev.code,
                to: conditions.data.length > 0 ?
conditions.data[0].code : actions.data.length > 0 ?
actions.data[0].code : ''
            }
        }
    }));

```

```

        };
    } else {
        return {
            from: conditions.data.length > 0 ?
conditions.data[0].code : actions.data.length > 0 ?
actions.data[0].code : '',
            to: ev.code
        };
    }
    }), [currentActiveRuleObject]));

    const conditionNodes = useMemo(() => conditions?.data.map(cond
=> {
        return {
            id: cond.code,
            title: `Code ${cond.code}`,
            font: {
                color: 'transparent',
            },
            label: `${cond.description} - ${cond.category[0]}\n`,
            shape: getShape(cond.category),
            color: {background: getColor(cond.category)},
            x: Math.random() * 600 - 300,
            y: Math.random() * 600 - 300
        };
    }), [currentActiveRuleObject]));

    const actionNodes = useMemo(() => actions?.data.length > 0 &&
actions?.data.map(act => {
        return {
            id: act.code,
            title: `Code ${act.code}`,
            label: `${act.description} -
${act.category[0].toUpperCase()}`,
            shape: getShape(act.category),
            color: {background: getColor(act.category)},
            x: Math.random() * 600 - 300,
            y: Math.random() * 600 - 300
        };
    }), [currentActiveRuleObject]));

    useEffect(() => {
        setState(({graph, counter, ...rest}) => {
            return {
                ...state,
                graph: {
                    nodes: [
                        ...eventNodes,
                        ...actionNodes,
                        ...conditionNodes
                    ],
                    edges: [

```

```

        ...eventEdges,
        ...actions.edges
    ]
    },
    ...rest
  };
});
}, [currentActiveRule]);

useEffect(() => {
  setCurrentNode(state?.graph?.nodes.find(node => node.id
=== currentNodeId));
}, [currentNodeId, state?.graph?.nodes]);

useEffect(() => {
  setCurrentEdge(state?.graph?.edges.find(edge => edge.id
=== currentEdgeId));
}, [currentEdgeId, state?.graph?.edges]);

const RulesGraphRightPanel = () => {
  return (<>
    <div className={commonStyles.rightPanelBlock}>
      <Header/>
      <hr className={commonStyles.line}/>
      <Managing/>
      <hr className={commonStyles.line}/>
      <div
className={commonStyles.rightPanelBlockTitle}>Управление
элементом</div>
      <div
className={commonStyles.rightPanelBlockAction}>
        <Icon name="plus"/>
        <div
className={commonStyles.rightPanelBlockActionText}
          onClick={() =>
setIsAddEdgeModalVisible(true)}>
          Добавить связь
        </div>
      </div>
      <div
className={commonStyles.rightPanelBlockAction}>
        <Icon name="edit"/>
        <div
className={commonStyles.rightPanelBlockActionText}
          onClick={() =>
setIsEditGraphModalVisible(true)}>
          Изменить
        </div>
      </div>
    </div>
  );
};

```

```

className={commonStyles.rightPanelBlockAction}>
    <Icon name="info"/>
    <div

className={commonStyles.rightPanelBlockActionText}
    onClick={() =>
setIsEditGraphModalVisible(true)}>
    Информация
    </div>
    </div>
    <div
className={cn(commonStyles.rightPanelBlockAction,
    {deleteDisabled: currentNode &&
actions.data.length < 2
    || currentEdge && actions.edges.length
< 2}})}
    style={{color: variables.redColor}}>
    <Icon name="korzina"
color={variables.redColor}/>
    <div

className={commonStyles.rightPanelBlockActionText}
    onClick={() => !currentNode
    ? actions.edges.length > 1 && (
        onRemoveEdgeFromAR(currentEdge),
        // onRemoveEdge(state,
setState, currentEdge),
        setCurrentEdgeId(undefined)
    )
    : actions.data.length > 1 && (
        // onRemoveNode(state, setState,
currentNode),
onRemoveNodeFromAR(currentNode.id),
        setCurrentNodeId(undefined)
    )
    }>
    Удалить
    </div>
    </div>
    </div>
    </>);
};

const RuleGraphToolbar = () => {
    return (
        <div className={commonStyles.toolbar}>
            <EditorRulePerformanceSelect/>
            <div onClick={() =>
setIsActionModalVisible(true)}>Добавить действие</div>
            <div>Легенда</div>
        </div>
    );
};

```

```

    );
  };

  return <MainTemplate blocks={<RulesGraphRightPanel/>}
id="editor-active-rules-container">
    <RuleGraphToolbar/>
    <GraphComponent
      graphData={state}
    />
    <EditNodeModal
      node={currentNode}
      onEdit={data => {
        // onEdit(data, state, setState, currentNode,
currentPointer);
        onEditNodeFromAR(currentNode.id, data.label);
        setIsEditGraphModalVisible(false);
      }}
      isVisible={isEditGraphModalVisible}
      onCancel={() => setIsEditGraphModalVisible(false)}
    />
    <AddEdgeModal
      nodesState={state.graph.nodes}
      node={currentNode}
      onOk={data => {
        onAddEdgeInAR(data);
        // onAddEdge(data, setState);
        setIsAddEdgeModalVisible(false);
      }}
      isVisible={isAddEdgeModalVisible}
      onCancel={() => setIsAddEdgeModalVisible(false)}
    />
    <AddActionModal
      currentSystemCode={currentSystemCode}
      currentActiveRule={currentActiveRuleObject}
      isVisible={isAddActionModalVisible}
      onCancel={() => setIsActionModalVisible(false)}
    />
  </MainTemplate>;
};

export default EditorRuleGraph;

```

Листинг Б.6 – Файл GraphComponent/index.js

```

import React from 'react';
import Graph from 'react-graph-vis';
import variables from
'../../../../../../../../variables.module.scss';
import commonStyles from
'../../../../../../../../common/styles/styles.module.scss';
import PropTypes from 'prop-types';

const options = {

```

```

    layout: {
      hierarchical: {
        direction: 'UD',
        sortMethod: 'directed',
        nodeSpacing: 200
      },
      improvedLayout: false
    },
    nodes: {
      color: {
        border: variables.darkBlueColor,
        background: variables.backgroundColor,
        highlight: {
          background: variables.borderColor
        }
      },
      font: {
        color: variables.whiteColor
      },
      shape: 'box'
    },
    edges: {
      color: variables.whiteColor,
    },
    physics: {
      enabled: false
    },
    interaction: {
      hover: true
    }
  };

const GraphComponent = ({graphData}) => {
  const {graph, events} = graphData;

  return <>
    <div className={commonStyles.elementsContainer}>
      <Graph graph={graph} options={options} events={events}
style={{height: '640px', width: '100%'}}/>
    </div>
  </>;
};

GraphComponent.propTypes = {
  graphData: PropTypes.object
};

export default GraphComponent;

```

Листинг Б.7 – Файл EditorRuleGraph/handlers.js

```

import variables from
'../../../../../variables.module.scss';

```

```

export const getColor = (type) => {
  switch (type) {
    case 'action':
      return variables.actionColor;
    case 'condition':
      return variables.conditionColor;
    case 'atomic event':
    case 'complex event':
    case 'aggregation event':
      return variables.eventColor;
  }
};

export const getShape = (type) => {
  switch (type) {
    case 'action':
      return 'box';
    case 'condition':
      return 'diamond';
    case 'atomic event':
    case 'complex event':
    case 'aggregation event':
      return 'ellipse';
  }
};

function randomColor() {
  const red = Math.floor(Math.random() *
256).toString(16).padStart(2, '0');
  const green = Math.floor(Math.random() *
256).toString(16).padStart(2, '0');
  const blue = Math.floor(Math.random() *
256).toString(16).padStart(2, '0');
  return `#${red}${green}${blue}`;
}

export const onOk = (data, setGraphData) => {
  setGraphData(({graph: {nodes, edges}, counter, ...rest}) => {
    const id = counter + 1;
    return {
      graph: {
        nodes: [
          ...nodes,
          {
            id,
            title: id,
            label: `${data}\n Id: ${id}`,
            font: {
              color:
data.toLowerCase().includes('condition') ? 'transparent' :
variables.whiteColor,

```

```

        },
        color: {background: getColor(data)},
        shape: getShape(data),
        x: Math.random() * 600 - 300,
        y: Math.random() * 600 - 300
      }
    ],
    edges: [
      ...edges,
    ]
  },
  counter: id,
  ...rest
};
});
};

export const createNode = (setGraphData, currentNode, currentEdge,
x, y) => {
  const color = randomColor();
  setGraphData(({graph: {nodes, edges}, counter, ...rest}) => {
    const id = counter + 1;
    return {
      graph: {
        nodes: [
          ...nodes,
          {id, title: `Code ${id}`, label: `Node ${id}`,
color, x, y}
        ],
        edges: [
          ...edges,
          {from: currentNode[0], to: id}
        ]
      },
      counter: id,
      ...rest
    };
  });
};

export const onEdit = (data, graphData, setGraphData, currentNode,
currentPointer) => {
  setGraphData(({graph: {edges}, counter, ...rest}) => {
    const temp = graphData.graph.nodes.filter(node => node.id
!== currentNode.id);
    return {
      ...graphData,
      graph: {
        nodes: [
          ...temp,
          {...currentNode, label: `${data.label} - A`,

```

```

x: currentPointer.x, y: currentPointer.y},
    ],
    edges: [
        ...edges
    ]
  },
  ...rest
};
});
};

export const onRemoveNode = (graphData, setGraphData, currentNode)
=> {
  const temp = graphData.graph.nodes.filter(node => node?.id !==
currentNode?.id);
  setGraphData(({graph: {edges}, ...rest})) => {
    return {
      graph: {
        nodes: [
          ...temp,
        ],
        edges: [
          ...edges,
        ]
      },
      ...rest
    };
  });
};

export const onRemoveEdge = (graphData, setGraphData, currentEdge)
=> {
  const temp = graphData.graph.edges.filter(edge => edge?.id !==
currentEdge?.id);
  setGraphData(({graph: {nodes}, ...rest})) => {
    return {
      graph: {
        nodes: [
          ...nodes,
        ],
        edges: [
          ...temp,
        ]
      },
      ...rest
    };
  });
};

export const onAddEdge = (data, setGraphData) => {
  setGraphData(({graph: {nodes, edges}, ...rest})) => {
    return {

```

```

graph: {
  nodes: [
    ...nodes
  ],
  edges: data.condition ? [
    ...edges,
    data.event ? {from: data?.event, to:
data.condition} : null,
    data.action ? {from: data?.condition, to:
data.action} : null
  ] : [
    ...edges,
    data.event && data.action ? {from:
data?.event, to: data.action} : null
  ]
},
...rest
};
});
};

```

Листинг Б.8 – Файл EditNodeModal/index.js

```

import React, {useEffect} from 'react';
import {Button, Input, Modal} from 'antd';
import styles from './styles.module.scss';
import {useForm, Controller} from 'react-hook-form';
import PropTypes from 'prop-types';

const EditNodeModal = ({isVisible, onCancel, onEdit, node}) => {

  const {reset, control, handleSubmit} = useForm();

  useEffect(() => {
    if (node?.label) {
      reset({
        // eslint-disable-next-line react/prop-types
        label: node?.label.slice(0, node?.label.indexOf('
- A'))
      });
    }
  }, [node?.label]);

  return <Modal title="Редактирование ноды" open={isVisible}
onCancel={onCancel} footer={false}>
    <form onSubmit={handleSubmit(onEdit)}>
      <div className={styles.formItem}>
        <div className={styles.formItemTittle}>Лейбл</div>
        <Controller
          render={({field}) => <Input {...field}/>}
          name="label"
          control={control}

```

```

        rules={{required: true}}
        defaultValue={node?.label.slice(0,
node?.label.indexOf(' - A'))}
      />
    </div>
    <div className={styles.footer}>
      <Button type="primary" htmlType="submit">
        Сохранить
      </Button>
    </div>
  </form>
</Modal>;
};

EditNodeModal.propTypes = {
  isVisible: PropTypes.bool,
  onCancel: PropTypes.func,
  onEdit: PropTypes.func,
  node: PropTypes.object
};

export default EditNodeModal;

```

Листинг Б.9 – Файл AddEdgeModal/index.js

```

import React, {useEffect, useMemo} from 'react';
import {Button, Modal, Select} from 'antd';
import styles from './styles.module.scss';
import commonStyles from
'../../../../../../../../common/styles/styles.module.scss';
import {Controller, useForm} from 'react-hook-form';
import PropTypes from 'prop-types';
import {useAppDispatch, useTypedSelector} from
'../../../../../../../../hooks/useTypedSelector';
import {useAction} from '../../../../../../../../hooks/useAction';

const AddEdgeModal = ({isVisible, onCancel, onOk, nodesState}) =>
{
  const {reset, control, handleSubmit, register, watch} =
useForm({
    action: '',
    condition: '',
    event: ''
  });

  const watchEvent = watch('event');
  const watchAction = watch('action');
  const {events} = useTypedSelector(state =>
state.eventsValues);
  const dispatch = useAppDispatch();
  const {fetchEventsBySystemCode} = useAction();
  const currentSystemCode =

```

```

localStorage.getItem('currentSystemCode');

useEffect(() => {
  reset({
    action: '',
    condition: '',
    event: ''
  });
}, []);

useEffect(() => {
  if(!events || events.length < 1) {
    dispatch(() =>
fetchEventsBySystemCode(currentSystemCode || currentSystemCode));
  }
}, []);

const eventOptions = useMemo(() => events
  .map(ev => {
    return {
      label: `${ev.description} -
${ev.categoryEvent[0].toUpperCase()}E`,
      value: ev.code
    };
  }), [events]);

const conditionOptions = nodesState
  .filter(node => node.id.toLowerCase().includes('c'))
  .map(conditionNode => {
    return {
      label: conditionNode.label,
      value: conditionNode.id
    };
  });

const actionOptions = nodesState
  .filter(node => node.label.toLowerCase().includes('a') &&
!node.label.toLowerCase().includes('e'))
  .map(actionNode => {
    return {
      label: actionNode.label,
      value: actionNode.id
    };
  });

return <Modal title="Добавление связи" open={isVisible}
onCancel={onCancel} footer={false}>
  <form onSubmit={handleSubmit(onOk)}
className={styles.form__modal}>
    <div className={styles.form__item}>
      <div className={styles.form__tittle}>Событие</div>
      <Controller

```

```

        render={({field}) =>
            <Select {...register('event', {
                required: 'Выберите событие',
            })} className={commonStyles.select}
options={eventOptions} {...field}/>
            name="event"
            control={control}
            defaultValue=""
        />

        <div className={styles.form__tittle}>Условие</div>
        <Controller
            render={({field}) =>
                <Select {...register('condition')}
className={commonStyles.select}
                    options={[...conditionOptions, {
                        label: 'Не выбрано',
                        value: null
                    }]} {...field}/>
                name="condition"
                control={control}
                defaultValue=""
            />

        <div
className={styles.form__tittle}>Действие</div>
        <Controller
            render={({field}) =>
                <Select {...register('action', {
                    required: 'Выберите действие',
                })}
                    className={commonStyles.select}
                    options={actionOptions}
                    {...field}
                />
                name="action"
                control={control}
                defaultValue=""
            />
        </div>

        <div className={styles.form__footer}>
            <Button disabled={!watchEvent || watchEvent ===
''} || (!watchAction || watchAction === '')}
                type="primary" htmlType="submit">
                Создать
            </Button>
        </div>
    </form>
</Modal>;
};

```

```
AddEdgeModal.propTypes = {
  isVisible: PropTypes.bool,
  nodesState: PropTypes.object,
  onCancel: PropTypes.func,
  onOk: PropTypes.func
};
```

```
export default AddEdgeModal;
```

Листинг Б.10 – Файл AddActionModal/index.js

```
import React, {useMemo} from 'react';
import {Button, Form, Input, Modal, Select} from 'antd';
import commonStyles from
'../../../../../common/styles/styles.module.scss';
import PropTypes from 'prop-types';
import {useTypedSelector} from
'../../../../../hooks/useTypedSelector';
import {useDispatch} from 'react-redux';
import {useAction} from ' ../../../../../hooks/useAction';

const AddActionModal = ({isVisible, onCancel, currentActiveRule,
currentSystemCode}) => {
  const currentActiveRuleObject =
JSON.parse(localStorage.getItem('currentActiveRuleObject') ||
'{}');
  const {events} = useTypedSelector(state =>
state.eventsValues);
  const conditions =
JSON.parse(currentActiveRuleObject?.condition);
  const dispatch = useDispatch();
  const {updateActiveRule} = useAction();

  const eventOptions = useMemo(() => events
    .map(ev => {
      return {
        label: `${ev.description} -
${ev.categoryEvent[0].toUpperCase()}E`,
        value: ev.code
      };
    }), [events]);

  const conditionOptions = conditions.data?.map(cond => {
    return {
      label: cond.description,
      value: cond.code
    };
  });

  const onOk = async (data) => {
    const actionCode = 'a' + Math.round(Math.random() * 100);
    const requestAction =
```

```

JSON.parse(currentActiveRule.action);
    requestAction.data.push({
        category: 'action',
        description: data.description,
        code: actionCode
    });

    requestAction.edges.push({
        from: data.condition,
        to: actionCode
    });

    const requestBody = {
        description: currentActiveRule.description,
        condition: currentActiveRule.condition,
        action: JSON.stringify({data: requestAction.data,
edges: requestAction.edges}),
        code: currentActiveRule.code
    };

    await dispatch(() => updateActiveRule(requestBody,
currentSystemCode));
    onCancel();
};

return <Modal
    title="Добавление действия"
    open={isVisible}
    onCancel={onCancel}
    footer=[
        <Button type="primary" form="addActionForm"
key="submit" htmlType="submit">
            Сохранить
        </Button>
    ]
>
    <Form
        name="addActionForm"
        labelCol={{span: 6}}
        wrapperCol={{span: 16}}
        onFinish={onOk}
    >
        <Form.Item
            label="Название"
            name="description"
            rules={[{required: true, message: 'Заполните
поле'}}]
        >
            <Input/>
        </Form.Item>
        <Form.Item
            label="Условие"

```

```

        name="condition"
        rules={[{required: true, message: 'Заполните
поле'}}]}
      >
        <Select className={commonStyles.select}
          options={conditionOptions}/>
        </Form.Item>
      </Form>
    </Modal>;
  };

AddActionModal.propTypes = {
  isVisible: PropTypes.bool,
  onCancel: PropTypes.func,
  currentActiveRule: PropTypes.object,
  currentSystemCode: PropTypes.string
};

export default AddActionModal;

```

Листинг Б.11 – Файл store/activeRules.ts

```

import {Dispatch} from 'redux';
import {
  ActiveRulesAction,
  ActiveRulesActionTypes,
  ActiveRulesInterface, AddActiveRuleInterface,
  DeleteActiveRuleInterface,
  RulesPerformance, UpdateActiveRuleInterface
} from '../types/activeRulesTypes';
import {ActiveRuleService} from
'../../api/services/ActiveRuleService';

export const setCurrentPerformance = (performance:
RulesPerformance): ActiveRulesAction => {
  return {type: ActiveRulesActionTypes.SET_CURRENT_PERFORMANCE,
payload: performance};
};

export const setCurrentActiveRule = (activeRule:
ActiveRulesInterface | null): any => {
  return (dispatch: Dispatch<ActiveRulesAction>) => {
    dispatch({type:
ActiveRulesActionTypes.SET_CURRENT_ACTIVE_RULE, payload:
activeRule});
  };
};

export const fetchActiveRuleBySystemCode = (code: string): any =>
{
  return async (dispatch: Dispatch<ActiveRulesAction>) => {
    try {
      dispatch({type:

```

```

ActiveRulesActionTypes.FETCH_RULES_SYSTEM_CODE));
    const response = await
ActiveRuleService.fetchActiveRuleBySystemCode(code);
    if (response) {
        dispatch({type:
ActiveRulesActionTypes.FETCH_RULES_SYSTEM_CODE_SUCCESS, payload:
response.data});
    }
    } catch (e) {
        dispatch({
            type:
ActiveRulesActionTypes.FETCH_RULES_SYSTEM_CODE_ERROR,
            payload: 'Произошла ошибка загрузки систем: ' + e
        });
    }
};

};

export const addActiveRule = (body: AddActiveRuleInterface,
eventData: { codeEvent: string, typeBind: string },
currentSystemCode: string) => {
    return async (dispatch: Dispatch<ActiveRulesAction>) => {
        try {
            const activeRule = await
ActiveRuleService.addActiveRule(body);
            if (activeRule) {
                await ActiveRuleService.bindActiveRuleEvent({
                    //@ts-ignore
                    codeRule: activeRule.data.code,
                    codeEvent: eventData.codeEvent, typeBind:
eventData.typeBind
                });
            }

            dispatch(fetchActiveRuleBySystemCode(currentSystemCode));
        }

        } catch (e) {
            console.error('Произошла ошибка добавления активного
правила: ', e);
        }
    };
};

export const updateActiveRule = (body: UpdateActiveRuleInterface,
currentSystemCode: string) => {
    return async (dispatch: Dispatch<ActiveRulesAction>) => {
        try {
            const res = await
ActiveRuleService.updateActiveRule(body);
            if(res) {

                localStorage.removeItem('currentActiveRuleObject');

```

```

        localStorage.setItem('currentActiveRuleObject',
JSON.stringify(res.data));

dispatch(fetchActiveRuleBySystemCode(currentSystemCode));
    }
    } catch (e) {
        console.error('Произошла ошибка добавления активного
правила: ', e);
    }
};

};

export const deleteActiveRule = (body: DeleteActiveRuleInterface,
currentSystemCode: string) => {
    return async (dispatch: Dispatch<ActiveRulesAction>) => {
        try {
            await ActiveRuleService.deleteActiveRule(body);

dispatch(fetchActiveRuleBySystemCode(currentSystemCode));
        } catch (e) {
            console.error('Произошла ошибка удаления активного
правила: ', e);
        }
    };
};
};

```

Листинг Б.12 – Файл store/editorActiveRules.ts

```

import {
    EditorActiveRulesAction,
    EditorActiveRulesActionTypes,
    EditorRulesPerformance
} from '../types/editorActiveRulesTypes';

export const setCurrentEditorRulePerformance = (performance:
EditorRulesPerformance): EditorActiveRulesAction => {
    return {type:
EditorActiveRulesActionTypes.SET_CURRENT_EDITOR_PERFORMANCE,
payload: performance};
};

```

Листинг Б.12 – Файл ActiveRuleService.ts

```

import {AxiosResponse} from 'axios';
import api from '../http';
import {
    ActiveRulesInterface,
    AddActiveRuleInterface,
    AddBindInterface,
    DeleteActiveRuleInterface, UpdateActiveRuleInterface
} from '../../store/types/activeRulesTypes';

export class ActiveRuleService {

```

```

    static async fetchActiveRuleBySystemCode(code: string =
'00016'): Promise<AxiosResponse<ActiveRulesInterface[]>> {
        return api.get<ActiveRulesInterface[]>(`/active-
rule/system/${code}`);
    }

    static async addActiveRule(body: AddActiveRuleInterface):
Promise<AxiosResponse<any[]>> {
        return api.post<any[]>(`/active-rule`, body);
    }

    static async deleteActiveRule(data:
DeleteActiveRuleInterface): Promise<AxiosResponse<any[]>> {
        return api.delete<any[]>(`/active-rule`, {data});
    }

    static async bindActiveRuleEvent(data: AddBindInterface):
Promise<AxiosResponse<any[]>> {
        return api.post<any[]>(`/active-rule/event`, data);
    }

    static async updateActiveRule(data:
UpdateActiveRuleInterface): Promise<AxiosResponse<any[]>> {
        return api.patch<any[]>(`/active-rule`, data);
    }
}
import React, {useMemo, useState} from 'react';
import styles from '../../../styles.module.scss';
import {BlockType} from '../../../types';
import classNames from 'classnames/bind';
import Icon from '../../../common/components/Icon';
import AddActionModal from
'../../../EditorRuleGraph/components/AddActionModal';
import {onRemoveNodeFromAR} from '../../../helpers';

const cn = classNames.bind(styles);

const Actions = () => {
    const currentActiveRuleObject =
JSON.parse(localStorage.getItem('currentActiveRuleObject') ||
'{}');
    const actions = JSON.parse(currentActiveRuleObject?.action);

```

```

    const currentSystemCode =
localStorage.getItem('currentSystemCode');

    const currentSystemName =
localStorage.getItem('currentSystemName');

    const [isAddActionModalVisible, setIsAddActionModalVisible] =
useState<boolean>(false);

    const actionsArray = useMemo(() => actions?.data?.map((cond:
BlockType) => {
        return {
            code: cond.code,
            description: cond.description,
            category: cond.category
        };
    }), [currentActiveRuleObject]);

return <div className={styles.block}>
    <div className={styles.block__title}>Действия</div>
    <div className={styles.block__wrapper}>
        <div className={styles.block__left_stroke}>/>
        <div className={styles.card}>
            {actionsArray?.map((act: BlockType) => {
                return <div id={act.code} key={act.code}
className={styles.card__content}>
                    <div
className={cn(styles.card__withDelete, {card__deleteBlock:
actions.data.length<2})}>
                        <div>
                            {act.category &&
                                <div
className={cn(styles.card__category,
styles.card__category_action)}>
                                    {act.category.at(0)?.toUpperCase()}
                                </div>
                        }
                    }
                )}
            )}
        </div>
    </div>

```

```

        {act.description} {act.code}
      </div>
      <Icon name="korzina" onClick={() =>
actions.data.length>1 && onRemoveNodeFromAR(act.code)}/>
    </div>
    <div className={styles.card__tags}>
      Исполняемая система:
      <div
className={cn(styles.card__category,
styles.card__category_systemTag)}>
        {currentSystemName}
      </div>
    </div>
  </div>;
  )))
  <div onClick={() =>
setIsAddActionModalVisible(true)}
        className={cn(styles.card__content,
styles.card__content_add)}>
    <Icon name="plus"/>Добавить
  </div>
</div>
<AddActionModal
  currentSystemCode={currentSystemCode}
  currentActiveRule={currentActiveRuleObject}
  isVisible={isAddActionModalVisible}
  onCancel={() => setIsAddActionModalVisible(false)}
/>
</div>;
};

export default Actions;

```

```

import React, {useMemo} from 'react';
import styles from '../..//styles.module.scss';
import {BlockType} from '../..//types';
import classNames from 'classnames/bind';

const cn = classNames.bind(styles);

const Conditions = () => {
  const currentActiveRuleObject =
    JSON.parse(localStorage.getItem('currentActiveRuleObject') ||
    '{}');

  const actions = JSON.parse(currentActiveRuleObject?.action);
  const conditions =
    JSON.parse(currentActiveRuleObject?.condition);

  const conditionsArray = useMemo(() =>
    conditions?.data?.map((cond: BlockType) => {
      return {
        code: cond.code,
        description: cond.description,
        category: cond.category
      };
    }), [currentActiveRuleObject]);

  const eventEdges = useMemo(() =>
    currentActiveRuleObject?.event.map((ev: any) => {
      if (ev.association?.typeBind === 'Event to Rule') {
        return {
          from: ev.code,
          to: conditions.data?.length > 0 ?
            conditions.data[0]?.code : actions.data?.length > 0 ?
            actions.data[0]?.code : ''
        };
      }
    })
  );

```

```

        return {
            from: conditions.data?.length > 0 ?
conditions.data[0]?.code : actions.data?.length > 0 ?
actions.data[0]?.code : '',
            to: ev.code
        };
    }
    }), [currentActiveRuleObject]);

return <div className={styles.block}>
    <div className={styles.block__title}>Условия</div>
    <div className={styles.block__wrapper}>
        <div className={styles.block__left_stroke}>/>
        <div className={styles.card}>
            {conditionsArray?.map((cond: BlockType) => {
                const filteredEventEdges = eventEdges
                    .filter((event: { from: string, to: string
})) => [event.to, event.from]
                    .includes(cond.code))
                .map((event: { from: string, to: string })
=> {
                    if (event.from[0] !== 'c') {
                        return
currentActiveRuleObject.event.find((ev: { code: string }) =>
ev.code === event.from).description;
                    } else if (event.to[0] !== 'c') {
                        return
currentActiveRuleObject.event.find((ev: { code: string }) =>
ev.code === event.to).description;
                    }
                });
            })

            const filteredActionEdges = actions.edges
                .filter((act: { from: string, to: string
})) => [act.to, act.from].includes(cond.code))

```

```

        map((act: { from: string, to: string }) => {
            if (act.from[0] !== 'c') {
                return actions.data.find((action:
{ code: string }) => action.code === act.from)?.description;
            } else if (act.to[0] !== 'c') {
                return actions.data.find((action:
{ code: string }) => action.code === act.to)?.description;
            }
        });
    });

import React, {useMemo} from 'react';
import styles from '../..//styles.module.scss';
import {BlockType} from '../..//types';
import classNames from 'classnames/bind';
import {useTypedSelector} from
'../../../../../hooks/useTypedSelector';

const cn = classNames.bind(styles);

const Events = () => {
    const currentActiveRuleObject =
JSON.parse(localStorage.getItem('currentActiveRuleObject') ||
'{}');

    const {events: eventsValues} = useTypedSelector(store =>
store.eventsValues);

    const events = useMemo(() =>
currentActiveRuleObject?.event.map((ev: BlockType) => {
        const findEvent = eventsValues.find(event => event.code
=== ev.code);

        return {
            code: ev.code,
            description: ev.description,
            categoryEvent: ev.categoryEvent,
            component: ev?.component,

```

```

        contextParam: findEvent?.contextParam
    };
    }), [currentActiveRuleObject]);

return <div className={styles.block}>
    <div className={styles.block__title}>События</div>
    <div className={styles.block__wrapper}>
        <div className={styles.block__left_stroke}/>
        <div className={styles.card}>
            {events.map((ev: BlockType) =>
                <div key={ev.code}
className={styles.card__content}>
                    <div>
                        {ev.categoryEvent &&
                            <div
className={cn(styles.card__category,
styles.card__category_event)}>
{ev.categoryEvent.at(0)?.toUpperCase()}E
                            </div>
                        }
                        {ev.description} {ev.code}
                    </div>
                    <div>
                        {ev?.component && <div
className={styles.card__tags}>
                            Группа датчика:
                            {ev.component.map(comp => <div
                                key={comp.description}
className={cn(styles.card__category,
styles.card__category_component)}>

```

Приложение В
ПРИМЕР СИСТЕМЫ ДЛЯ ТЕСТИРОВАНИЯ ПРИЛОЖЕНИЯ
(рекомендуемое)

Приложение под названием «Система мониторинга и контроля безопасности и обеспечения жизнедеятельности предприятия».

Назначением приложения является мониторинг и контроль пропускного режима сотрудников предприятия, поддержка системы жизнеобеспечения предприятия (температурный режим, обогрев и кондиционирование помещений, контроль электропотребления).

Предприятие состоит из нескольких зданий (корпусов). В корпусах расположены рабочие помещения сотрудников (кабинеты, лаборатории, офисы).

На предприятии поддерживается система пропусков для сотрудников. Имеется несколько внешних проходных, через которые сотрудники попадают на предприятие, предоставляя пропуск. В ночное время (например, с 22.00 до 7.00) несколько внешних проходных закрываются. Вход и выход сотрудников осуществляется через оставшимися открытыми проходные.

В ночное время вход в корпус и рабочее помещение разрешен сотрудникам только по пропускам. В ночное время сотрудник может попасть только в тот корпус и помещение, где находится его рабочее место.

На предприятии осуществляется мониторинг и контроль температурного режима на проходных, входных группах корпусов и в рабочих помещениях. Для этого установлены датчики температуры, системы обогрева и кондиционирования, датчики контроля потребляемой мощности и силы тока, системы включения и отключения приборов освещения, электроприборов и оборудования.

Осуществляется мониторинг и контроль пропускного режима:

1) вход и выход сотрудников через проходные в любое время суток только по пропускам;

2) в ночное время дополнительно контролируется перемещения сотрудников по предприятию – при входе на предприятие сотрудник обязательно должен попасть в свое рабочее помещение, при выходе из своего рабочего помещения сотрудник должен обязательно покинуть предприятие;

3) в ночное время блокируются определенные внешние проходные, перемещение через них сотрудников становится невозможным;

4) после окончания ночного времени все внешние проходные становятся доступными для перемещения сотрудников;

5) все перемещения сотрудников через проходные, а в ночное время перемещения в корпусах и рабочих помещениях, фиксируются в электронном журнале с указанием табельного номера сотрудника. даты/времени и вида перемещения (вход или выход).

6) отдельным образом фиксируются нештатные ситуации (например, сотрудник в ночное время вошел на предприятие через проходную, но за заданное время не появился в корпусе или в рабочем помещении и др.), о возникновении нештатных ситуаций информируются ответственные лица.

Осуществляется мониторинг и контроль жизнедеятельности предприятия:

1) мониторинг температурного режима на проходных, входных группах корпусов и в рабочих помещениях с помощью датчиков температуры, которые периодически в потоковом режиме отправляют данные в систему;

2) контроль допустимого диапазона температур на проходных, входных группах корпусов и в рабочих помещениях на основе данных, поступающих от датчиков контролируется допустимый диапазон температур;

3) в случае выхода температурных показателей за допустимые пределы включаются обогреватели (при понижении температуры) или кондиционеры (при повышении температуры);

4) контролируется не только факт выхода температуры за допустимые пределы, но и ярко выраженный тренд на повышение или понижение температуры, в этом случае включение обогревателей или кондиционеров происходит на основе прогнозных показателей.

Осуществляется мониторинг и контроль электропотребления предприятия:

1) с целью экономии электроэнергии общее освещение помещений отключается после того, как рабочее помещение покидают все сотрудники;

2) в вечернее время, после того как рабочее помещение покидают все сотрудники, отключаются электроприборы и оборудование;

3) мониторинг и контроль потребляемой мощности и силы тока в каждом корпусе – при возрастании нагрузки происходит постепенное отключение электроприборов в помещении и корпусе, при резком возрастании нагрузки полностью отключаются электроприборы и оборудование в помещении и корпусе.

Бизнес-логика приложения мониторинга и контроля строится на основе потоковой обработки событий и механизма активных правил. Мониторинг осуществляется на основе обработки поступающих событий. Управление заключается в выполнении активных правил в ответ на появление соответствующего события.

Все типы событий делятся на четыре категории:

1) временные события – события, инициируемые системой по заданному расписанию;

2) примитивные события – простые события, данные о которых поступают от датчиков и систем контроля;

3) агрегированные события – сложные события, данные о которых вычисляются с помощью непрерывных запросов к входным потокам;

4) составные события – сложные события, которые задаются с помощью операций алгебры событий над составляющими событиями.

Для обработки событий и реагирования на их возникновение в системе определяется набор типов активных правил.

Приложение Г
РЕЗУЛЬТАТЫ ФУНКЦИОНАЛЬНОГО ТЕСТИРОВАНИЯ
(рекомендуемое)

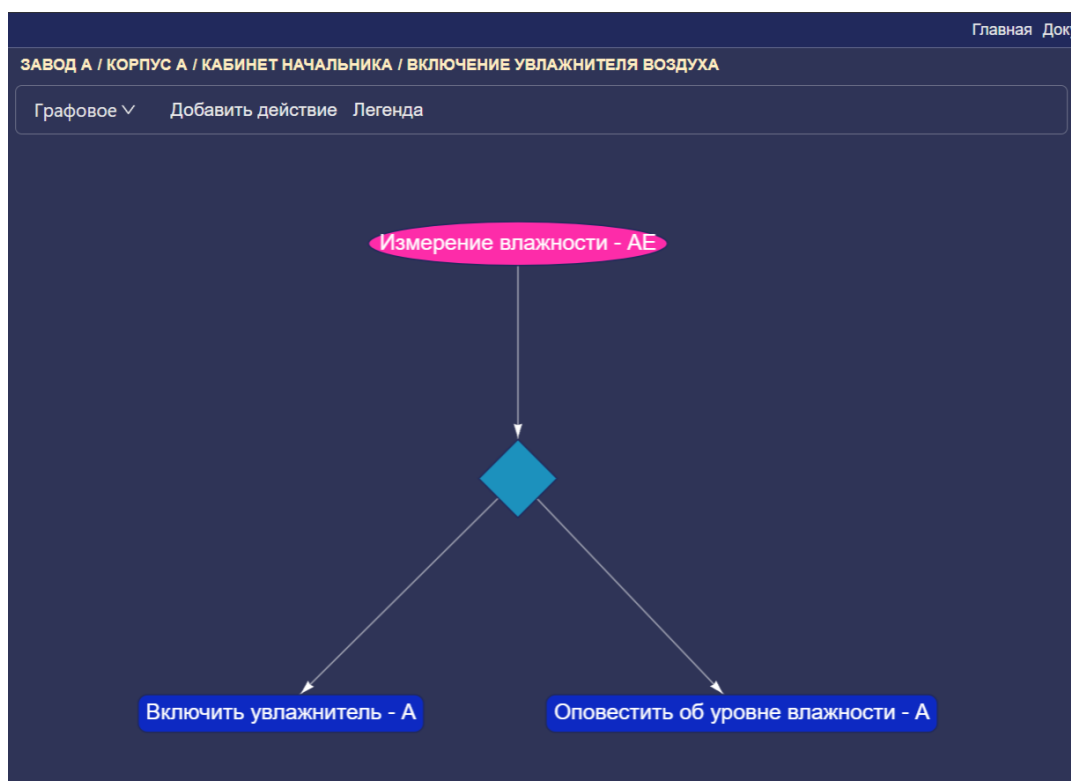


Рисунок Г.1 – Результат теста на добавление нового действия

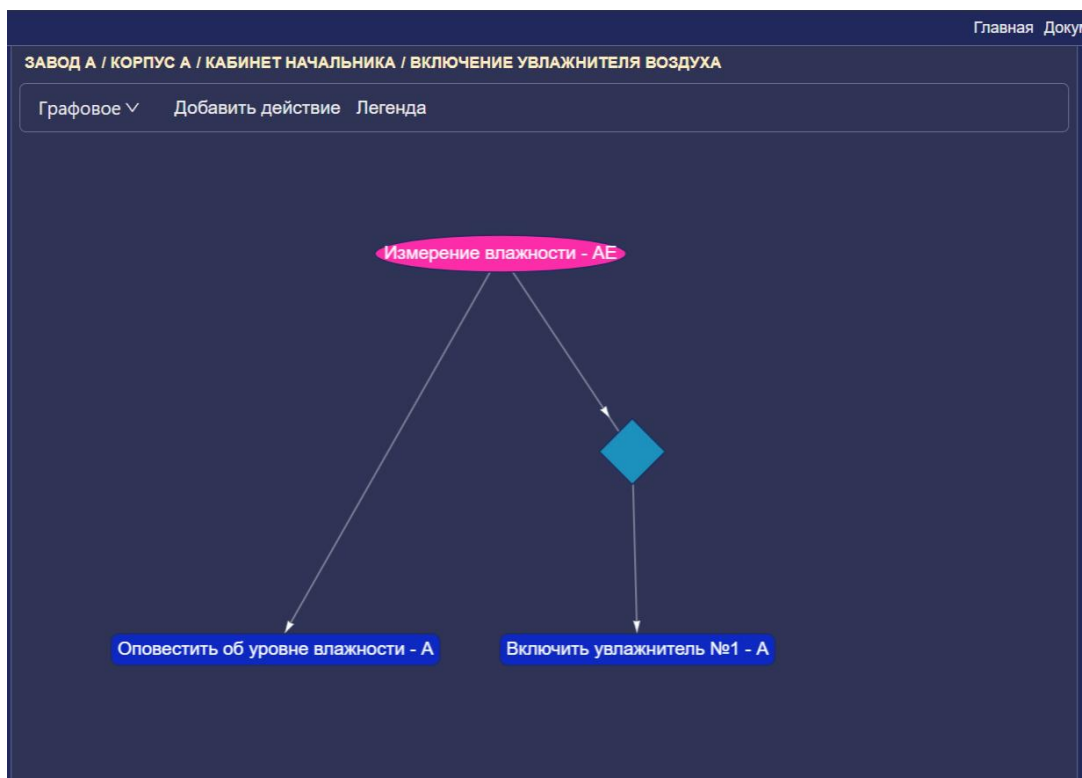


Рисунок Г.2 – Результат теста на добавление связи события и действия без условия

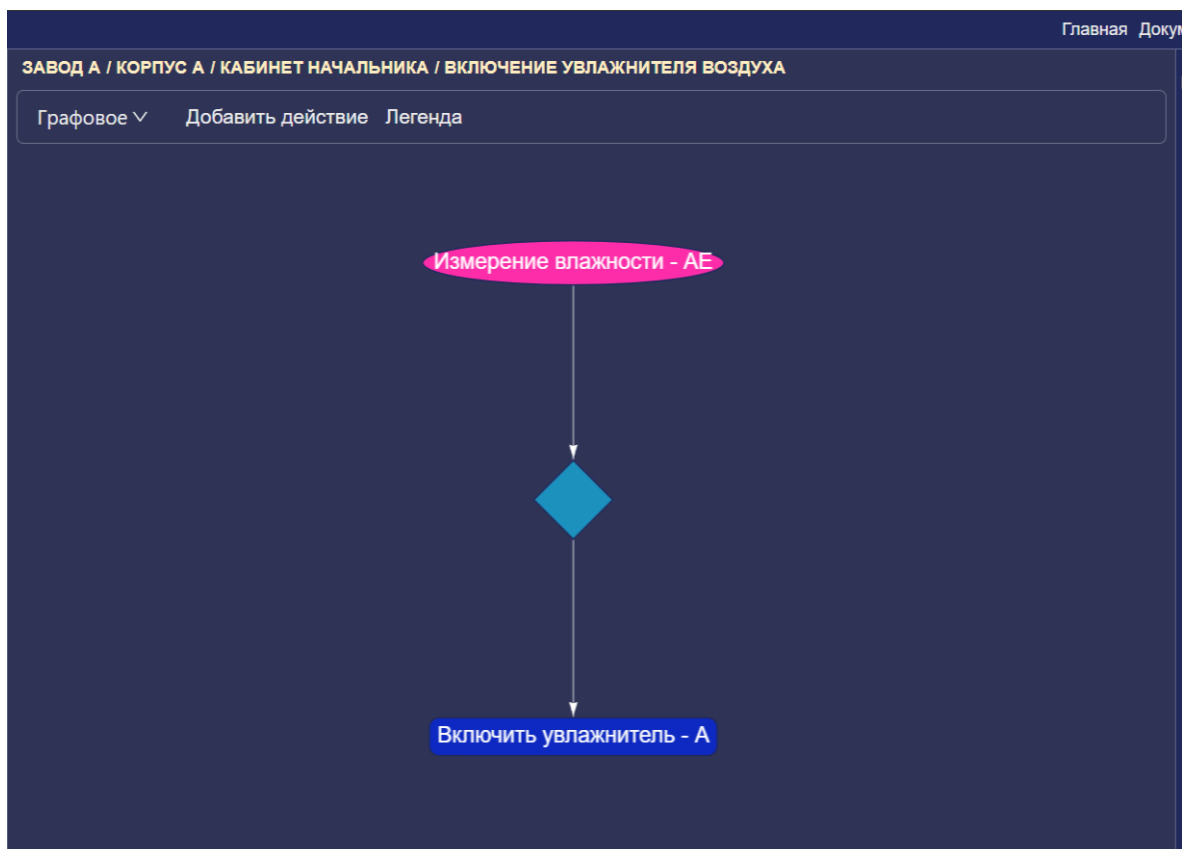


Рисунок Г.3 – Результат теста на удаление созданного в первом тесте действия

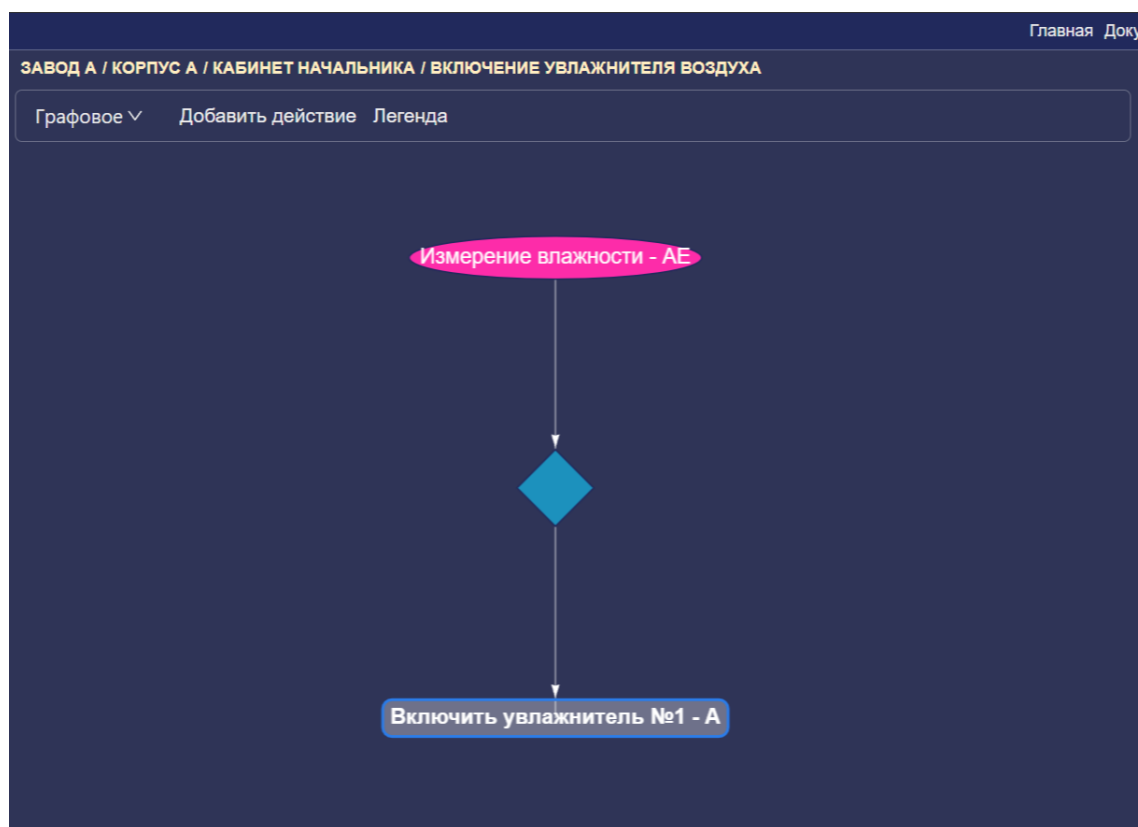


Рисунок Г.4 – Результат теста на редактирование названия действия

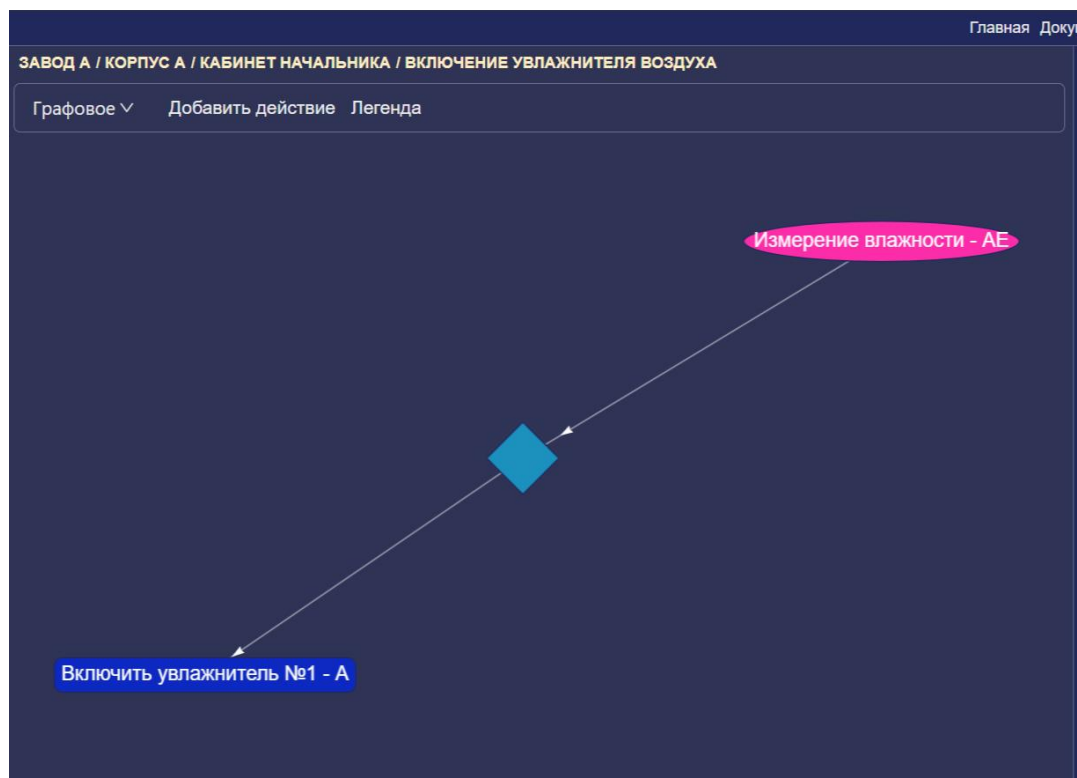


Рисунок Г.5 – Результат теста на перемещение элементов по рабочей области в рамках структуры графа

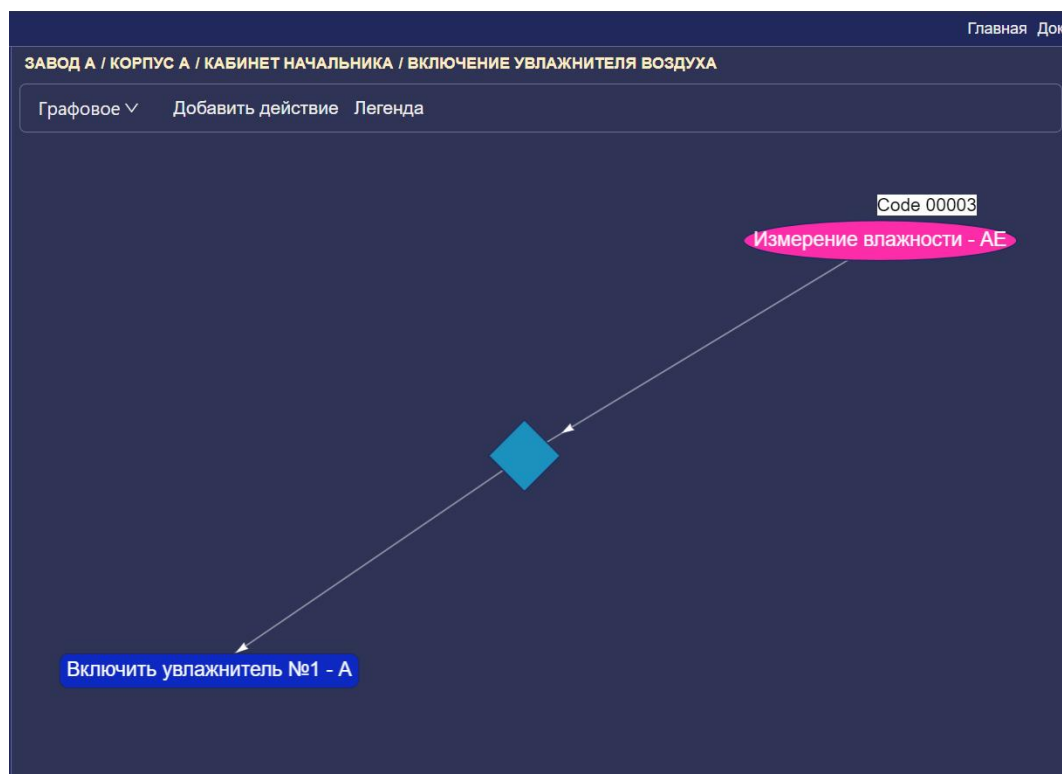


Рисунок Г.6 – Результат теста на отображение дополнительных данных об элементе графа

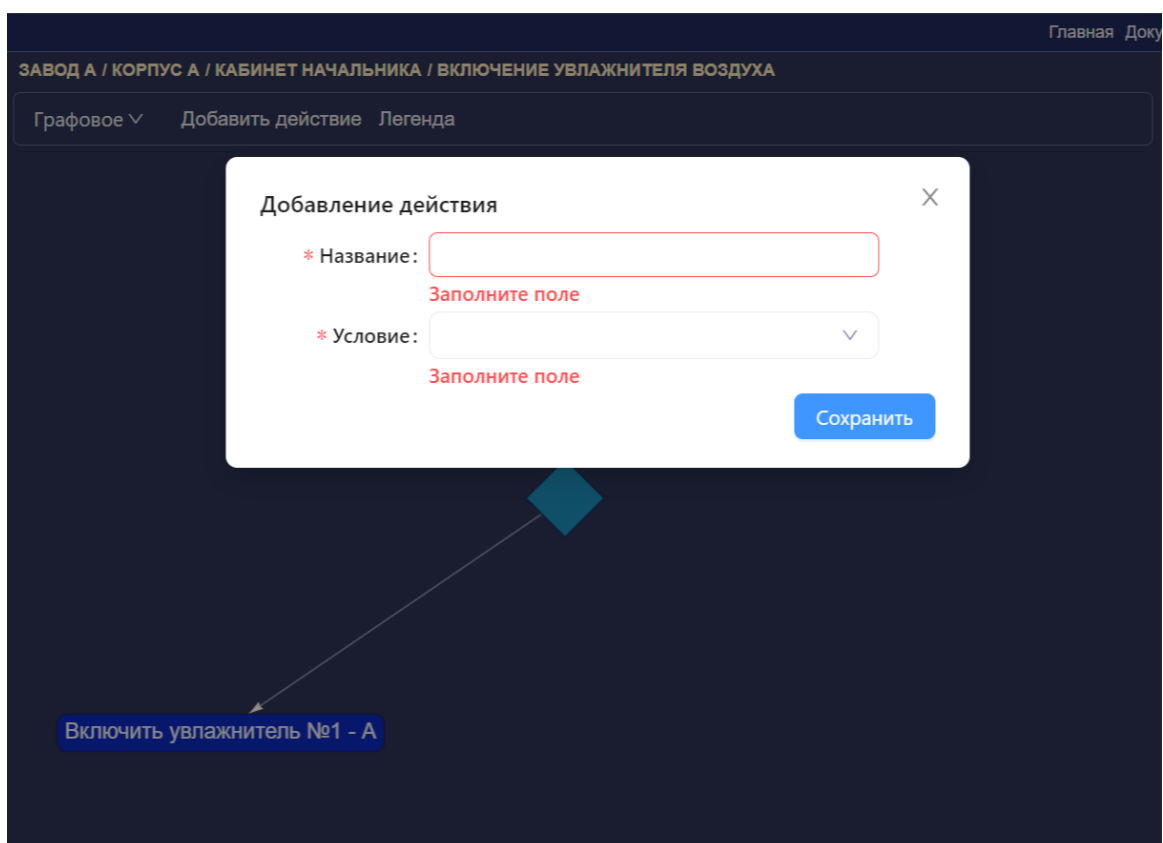


Рисунок Г.7 – Результат теста на проверку валидности полей добавления действия

Приложение Д
РЕЗУЛЬТАТЫ КОНФИГУРАЦИОННОГО ТЕСТИРОВАНИЯ
(рекомендуемое)

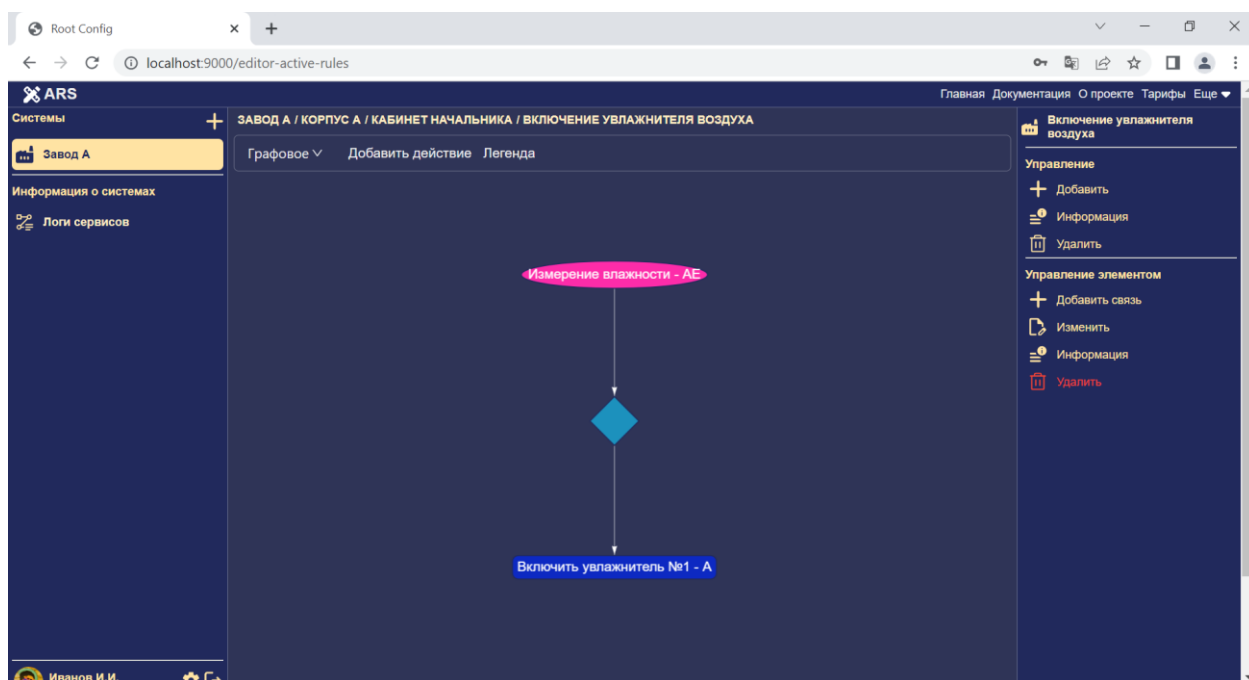


Рисунок Д.1 - Отображение страницы графового представления в браузере Google Chrome

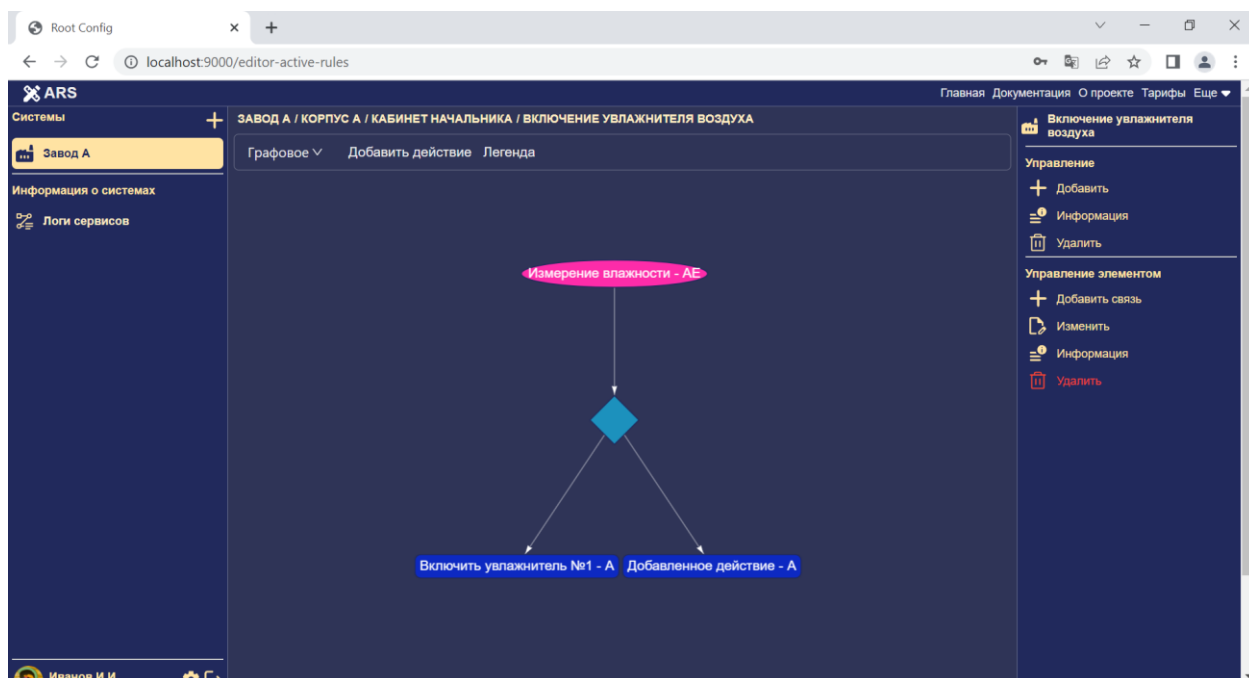


Рисунок Д.2 - Отображение созданного объекта метаданных в браузере Google Chrome

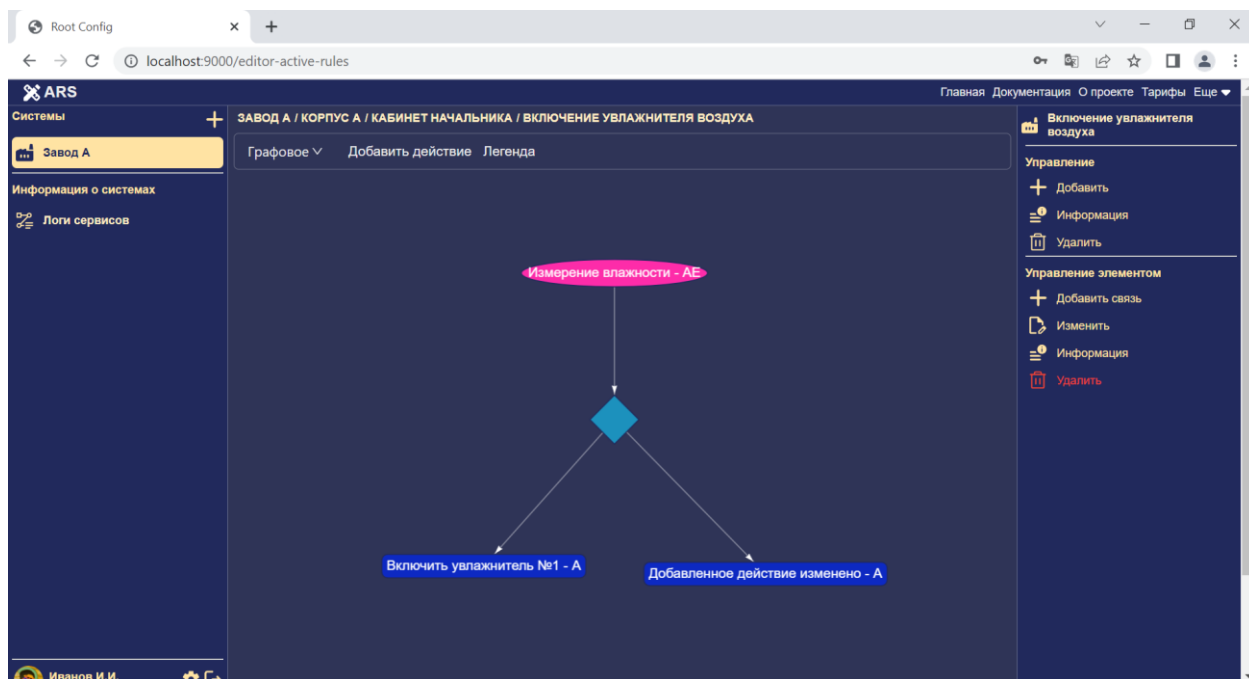


Рисунок Д.3 - Сохранение и отображение редактируемых данных элемента графа в браузере Google Chrome

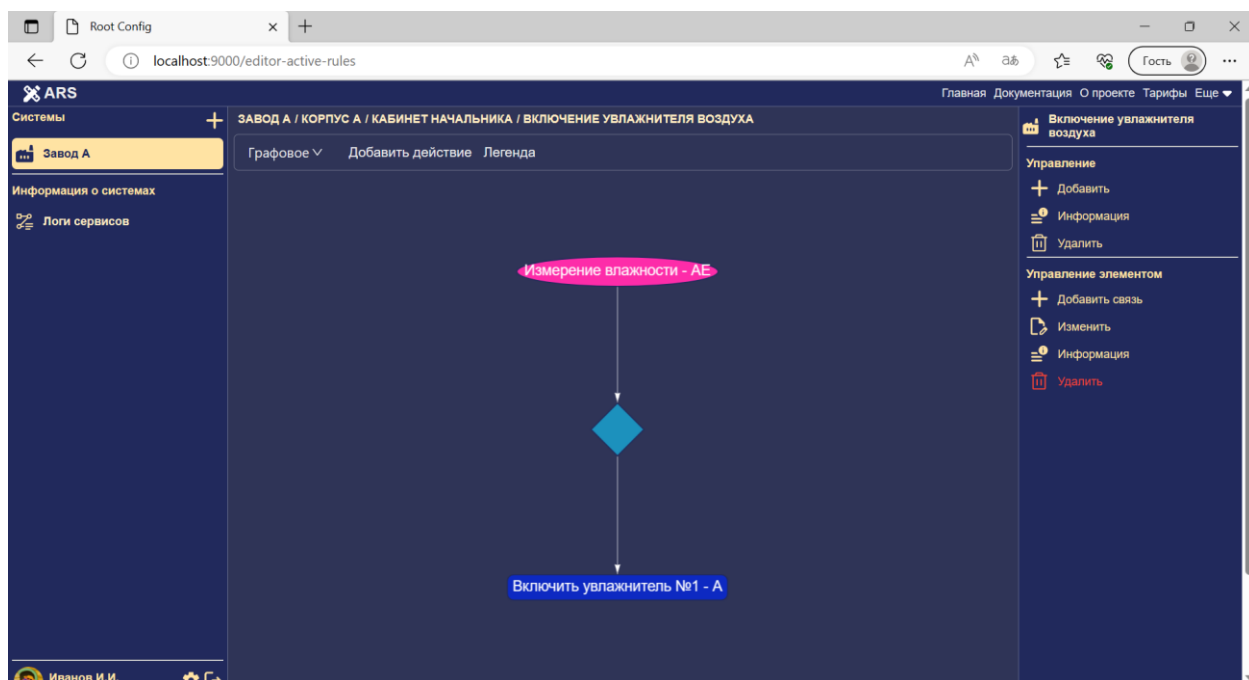


Рисунок Д.4 - Отображение страницы графового представления в браузере Microsoft Edge

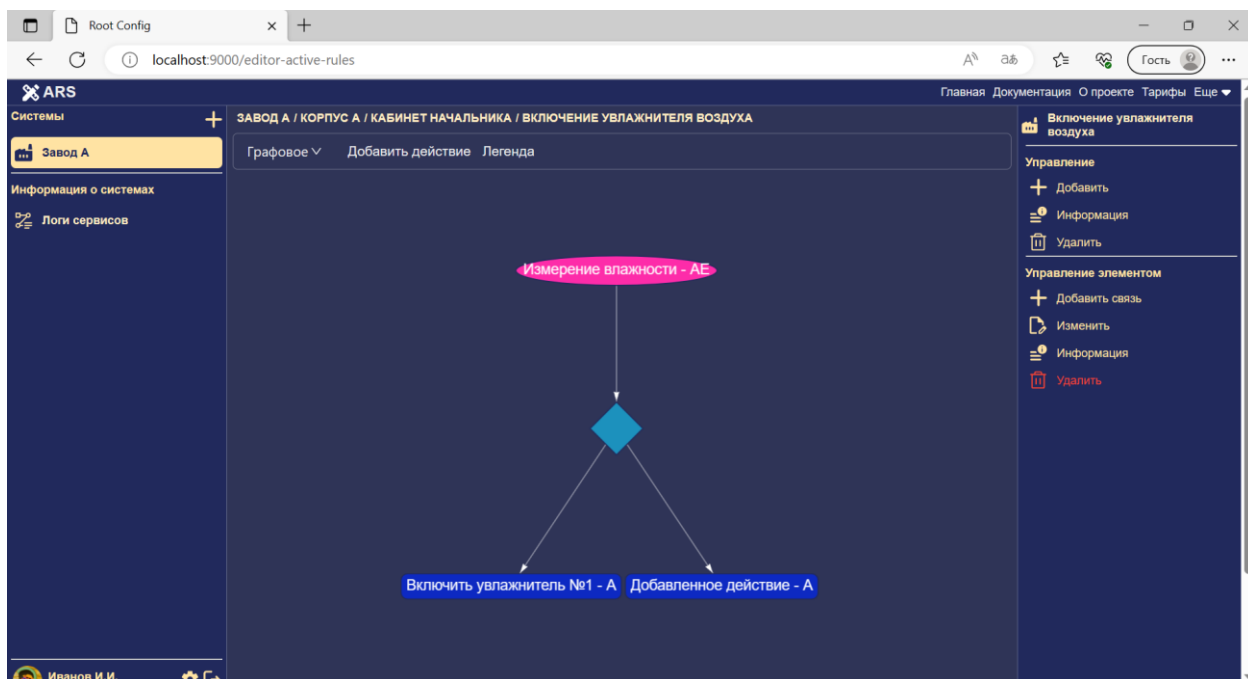


Рисунок Д.5 - Отображение созданного объекта метаданных в браузере Microsoft Edge

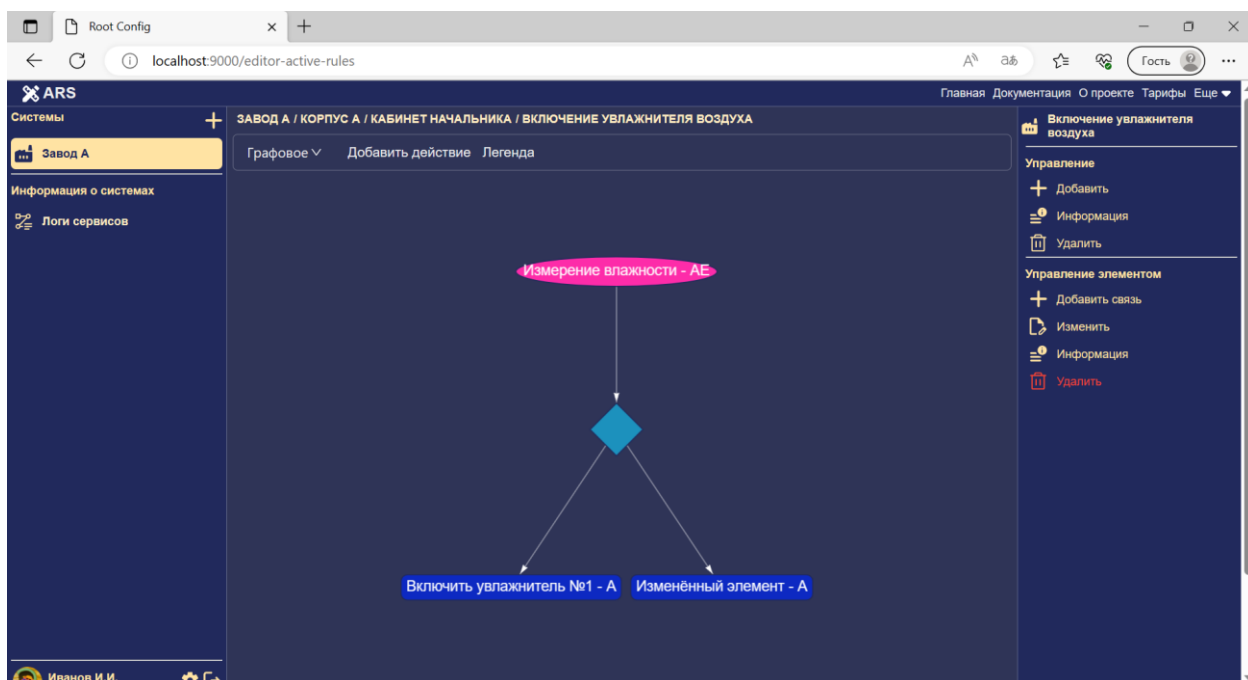


Рисунок Д.6 - Сохранение и отображение редактируемых данных элемента графа в браузере Microsoft Edge

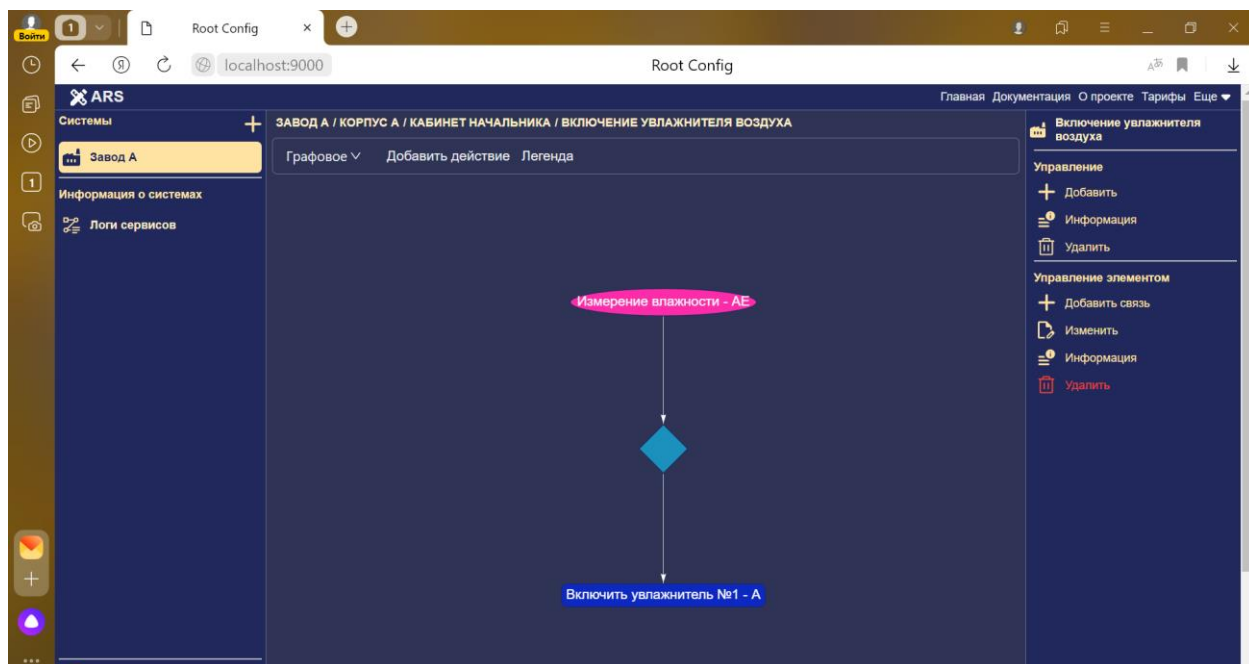


Рисунок Д.7 - Отображение страницы графового представления в браузере Яндекс.Браузер

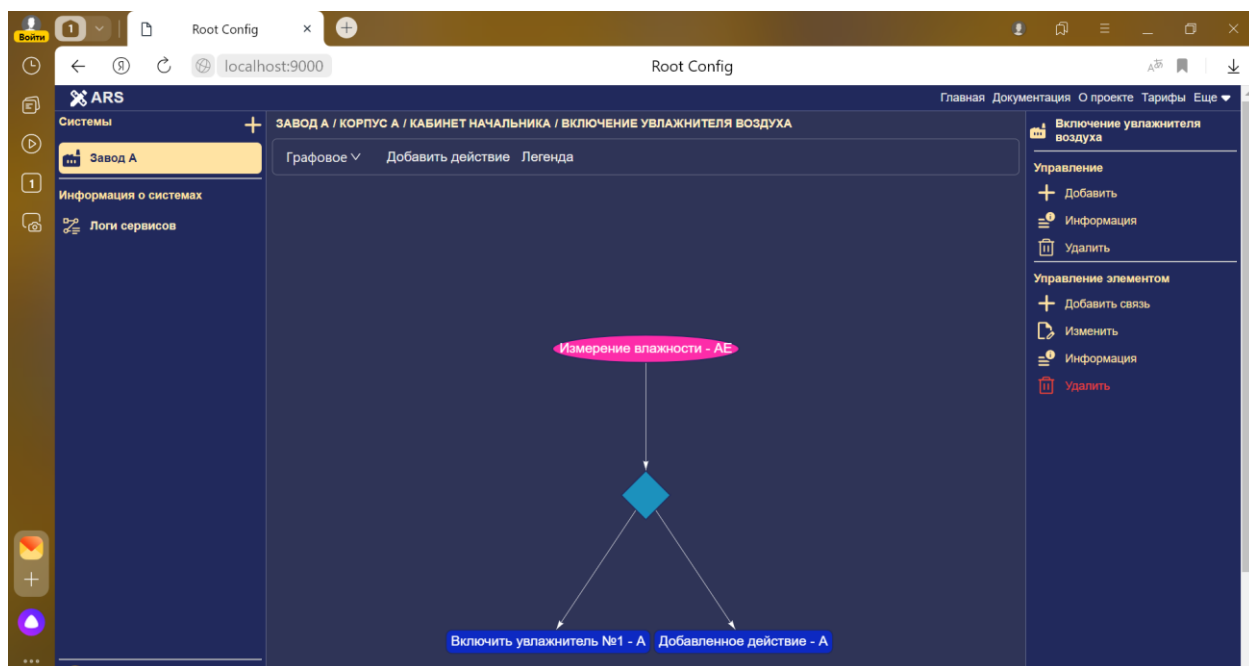


Рисунок Д.8 - Отображение созданного объекта метаданных в браузере Яндекс.Браузер

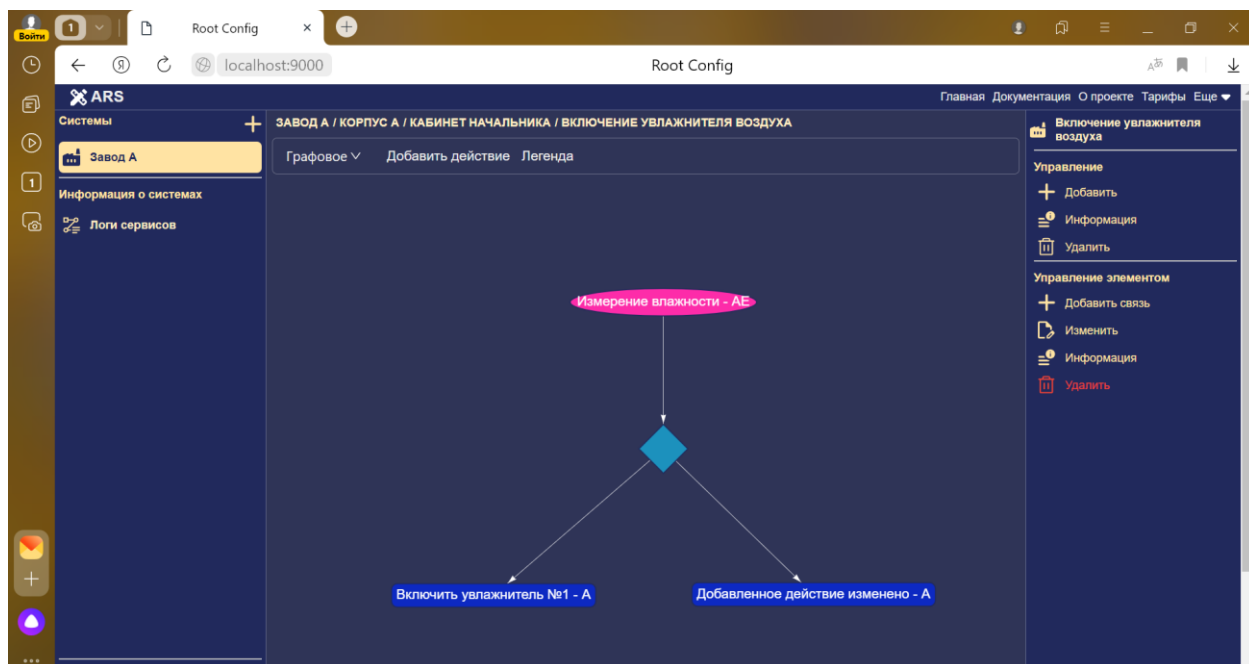


Рисунок Д.9 - Сохранение и отображение редактируемых данных элемента графа в браузере Яндекс.Браузер

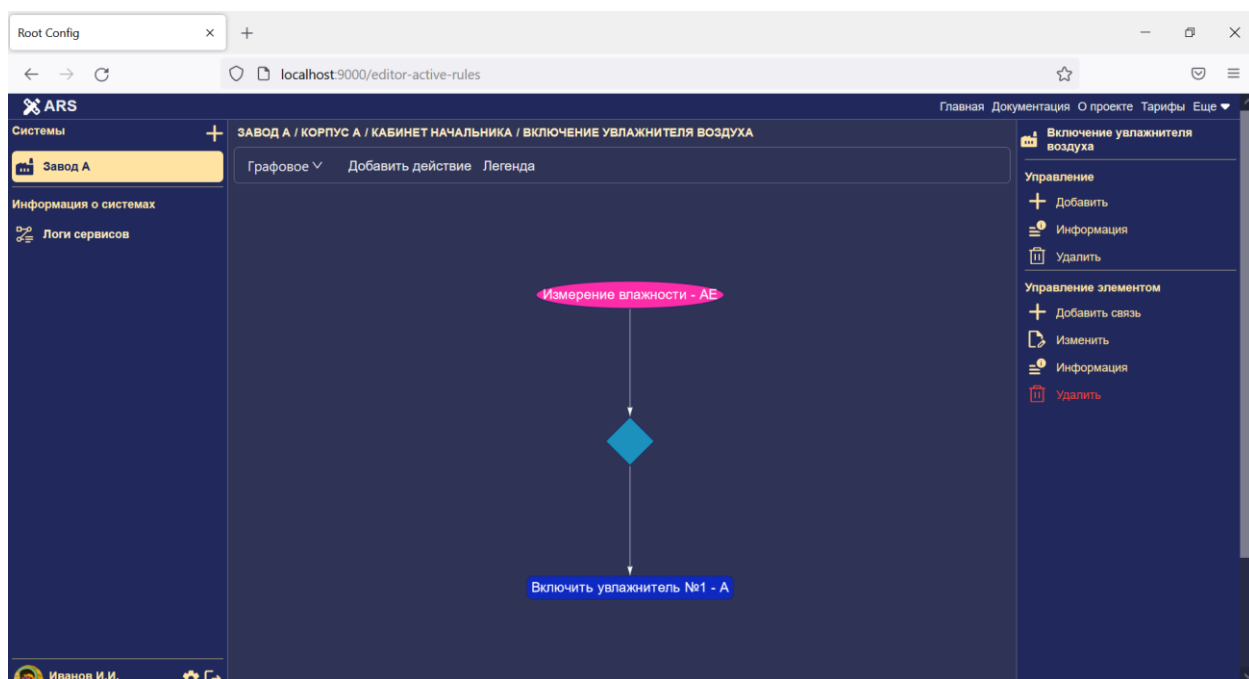


Рисунок Д.10 - Отображение страницы графового представления в браузере Firefox Browser

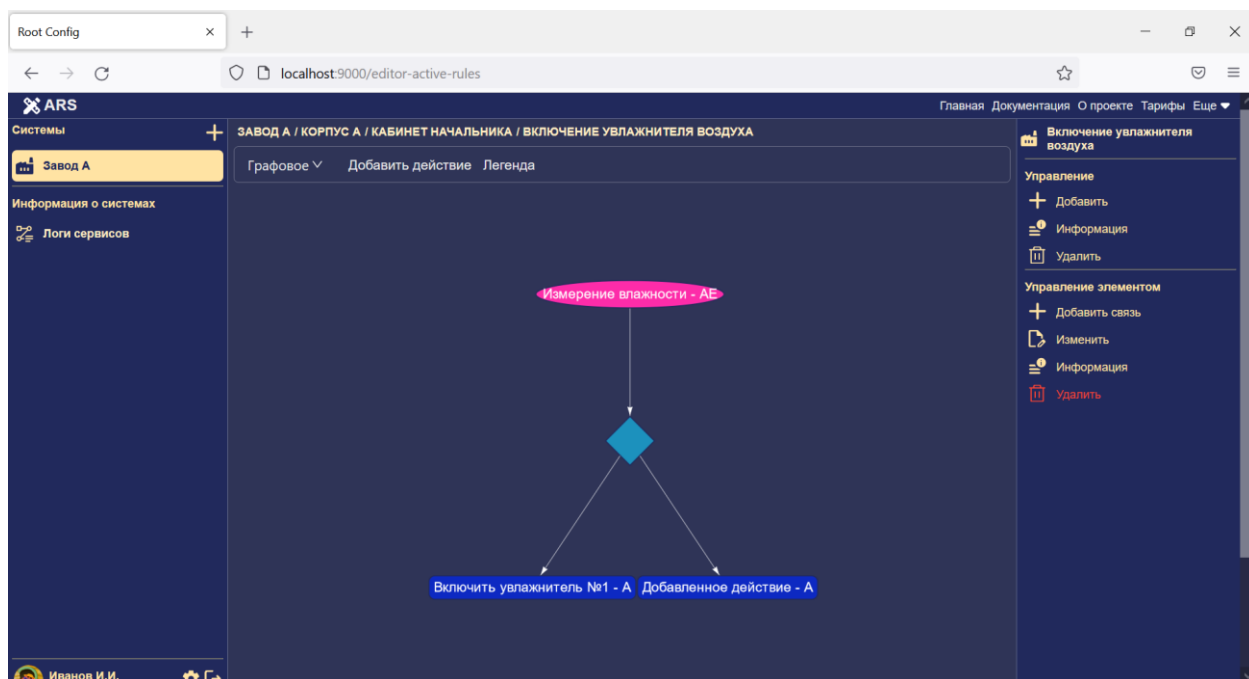


Рисунок Д.11 - Отображение созданного объекта метаданных в браузере Firefox Browser

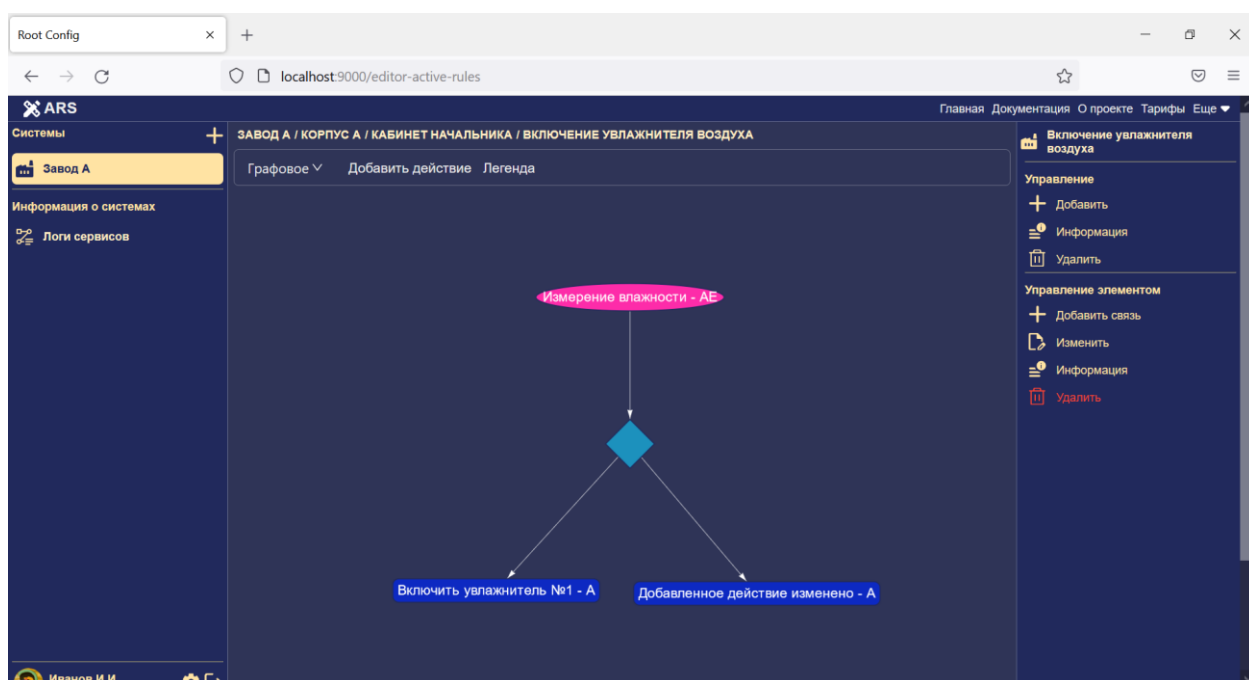


Рисунок Д.12 - Сохранение и отображение редактируемых данных элемента графа в браузере Firefox Browser

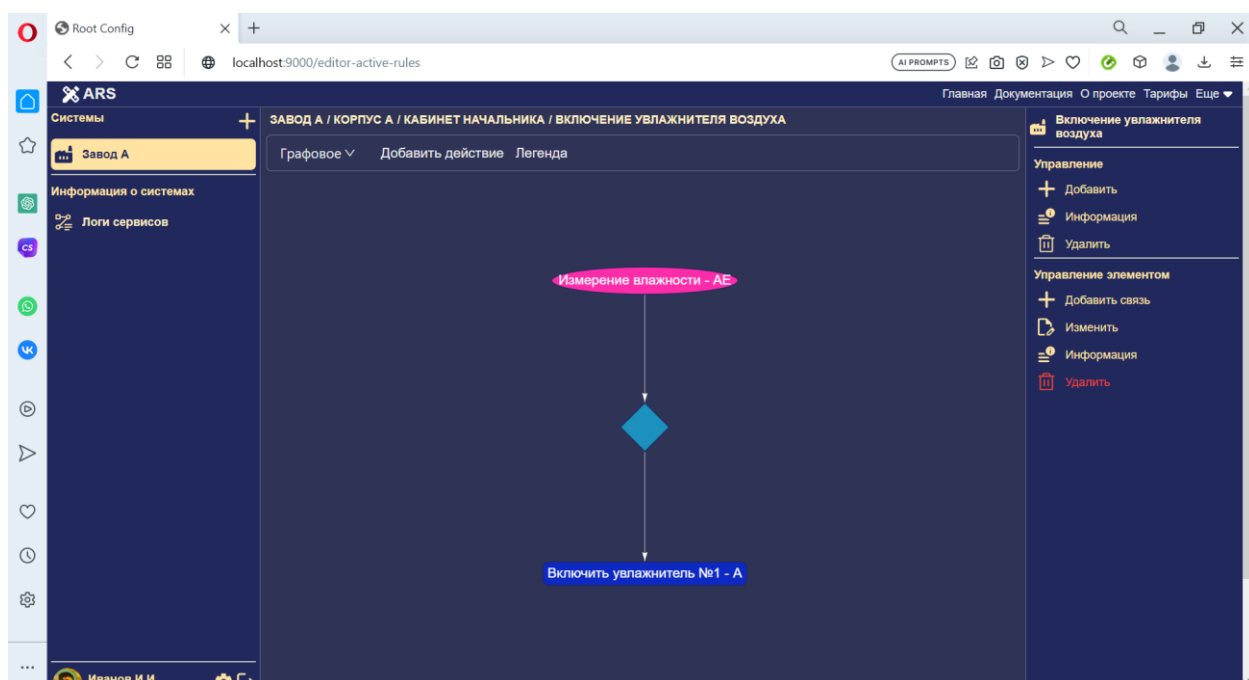


Рисунок Д.13 - Отображение страницы графового представления в браузере Opera

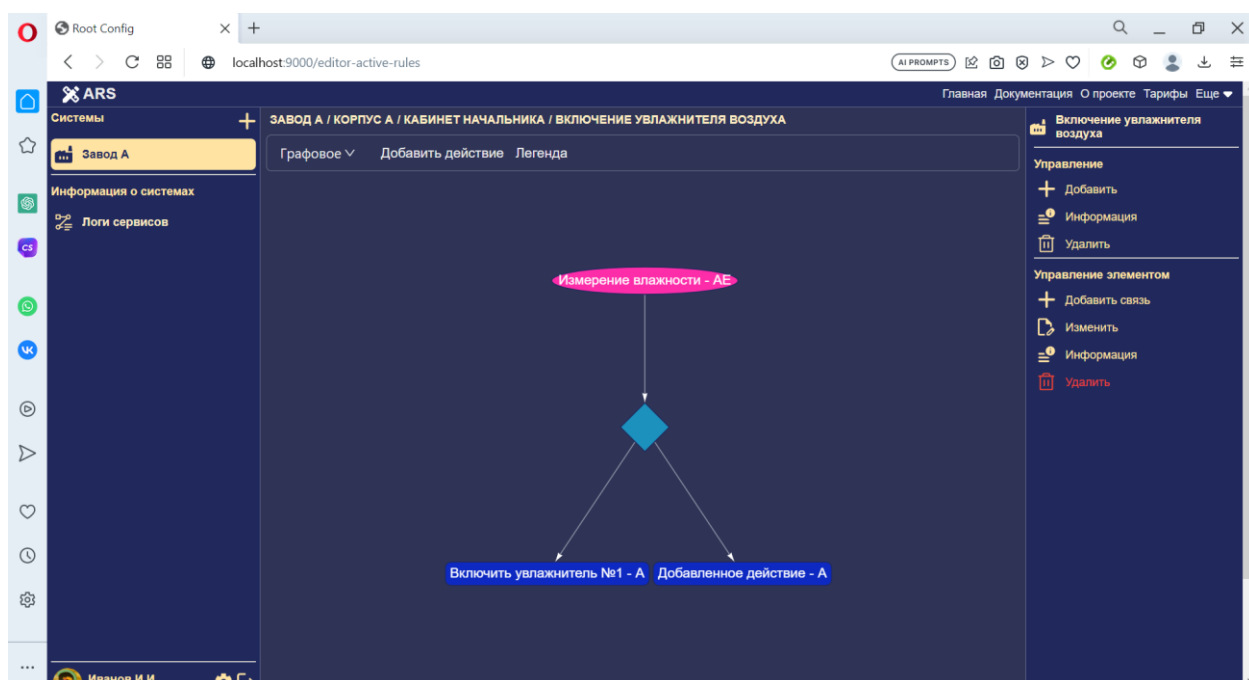


Рисунок Д.14 - Отображение созданного объекта метаданных в браузере Орега

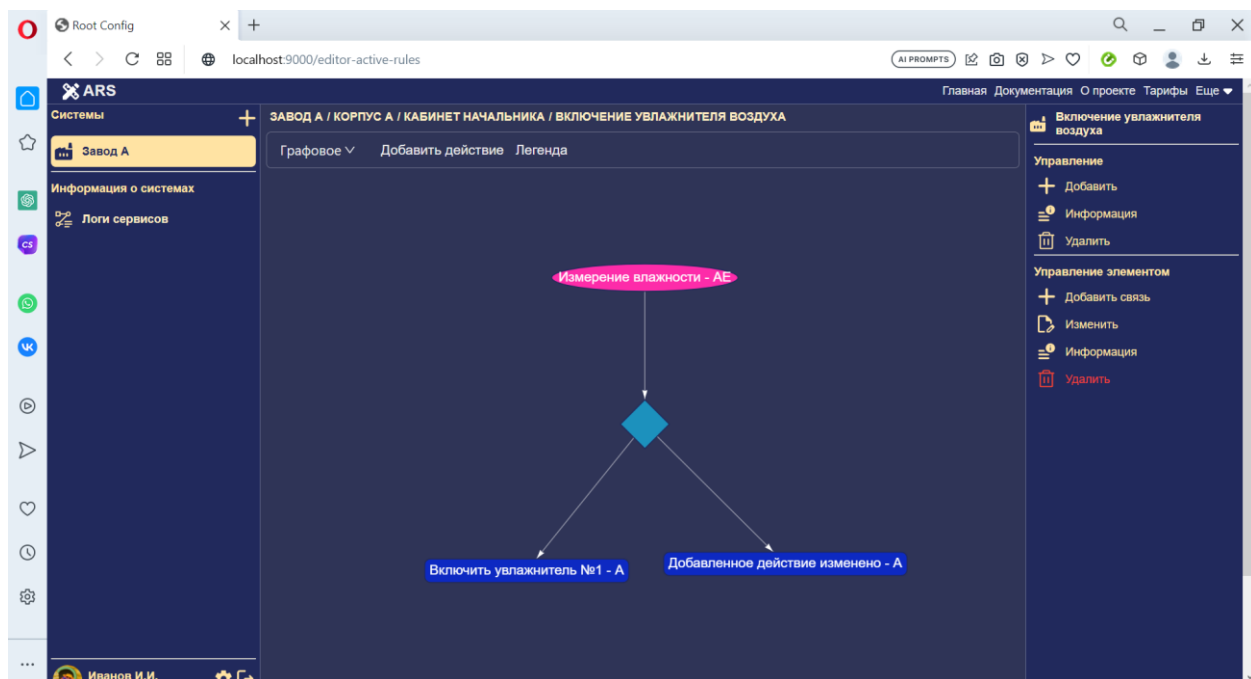


Рисунок Д.15 - Сохранение и отображение редактируемых данных элемента графа в браузере Opera

Приложение Е
РЕЗУЛЬТАТЫ МОДУЛЬНОГО ТЕСТИРОВАНИЯ
(рекомендуемое)

```

-----
[chrome 114.0.5735.106 mac os x #0-0] Running: chrome (v114.0.5735.106) on mac os x
[chrome 114.0.5735.106 mac os x #0-0] Session ID: 3dd8c2792e840065776403033aafd399
[chrome 114.0.5735.106 mac os x #0-0]
[chrome 114.0.5735.106 mac os x #0-0] » /tests/e2e/activerule.e2e.js
[chrome 114.0.5735.106 mac os x #0-0] Active rules page test
[chrome 114.0.5735.106 mac os x #0-0]     ✓ login
[chrome 114.0.5735.106 mac os x #0-0]     ✓ get subsystems
[chrome 114.0.5735.106 mac os x #0-0]     ✓ get active rules
[chrome 114.0.5735.106 mac os x #0-0]
[chrome 114.0.5735.106 mac os x #0-0] 3 passing (10.8s)

```

Рисунок Е.1 – Результат модульного тестирования раздела активных правил

```

[chrome 114.0.5735.106 mac os x #0-1] Running: chrome (v114.0.5735.106) on mac os x
[chrome 114.0.5735.106 mac os x #0-1] Session ID: 4135326b414e9890e1b831177cf8f440
[chrome 114.0.5735.106 mac os x #0-1]
[chrome 114.0.5735.106 mac os x #0-1] » /tests/e2e/editor-activerule.e2e.js
[chrome 114.0.5735.106 mac os x #0-1] Editor active rules page test
[chrome 114.0.5735.106 mac os x #0-1]     ✓ login
[chrome 114.0.5735.106 mac os x #0-1]     ✓ get subsystems
[chrome 114.0.5735.106 mac os x #0-1]     ✓ get active rules
[chrome 114.0.5735.106 mac os x #0-1]     ✓ get editor active rule
[chrome 114.0.5735.106 mac os x #0-1]
[chrome 114.0.5735.106 mac os x #0-1] 4 passing (11.3s)

```

Рисунок Е.2 – Результат модульного тестирования раздела редактора активных правил

```

-----
[chrome 114.0.5735.133 mac os x #0-0] Running: chrome (v114.0.5735.133) on mac os x
[chrome 114.0.5735.133 mac os x #0-0] Session ID: fe066c0b4d190aa3f250f825eab9fe74
[chrome 114.0.5735.133 mac os x #0-0]
[chrome 114.0.5735.133 mac os x #0-0] » /tests/e2e/editor-activerule.e2e.js
[chrome 114.0.5735.133 mac os x #0-0] Editor active rules page test
[chrome 114.0.5735.133 mac os x #0-0]      ✓ login
[chrome 114.0.5735.133 mac os x #0-0]      ✓ get subsystems
[chrome 114.0.5735.133 mac os x #0-0]      ✓ get active rules
[chrome 114.0.5735.133 mac os x #0-0]      ✓ get editor active rule
[chrome 114.0.5735.133 mac os x #0-0]      ✓ add action
[chrome 114.0.5735.133 mac os x #0-0]      ✓ add edge
[chrome 114.0.5735.133 mac os x #0-0]
[chrome 114.0.5735.133 mac os x #0-0] 6 passing (19.5s)

Spec Files:      1 passed, 1 total (100% completed) in 00:00:23

```

Рисунок Е.3 – Результат модульного тестирования раздела редактора активных правил

Приложение Ж
ЛИСТИНГ МОДУЛЬНЫХ ТЕСТОВ
(рекомендуемое)

Листинг Ж.1 – Тесты на модуль активных правил

```

const SystemsPage = require('../pages/systems.page');
const LoginPage = require('../pages/login.page');
const ActiveRulesPage = require('../pages/activerule.page');

describe('Active rules page test', () => {
  it('login', async () => {
    await LoginPage.open();
    await LoginPage.login('admin@admin.com', 'admin');

    await SystemsPage.loadSystems();
  });

  it('get subsystems', async () => {
    await SystemsPage.loadSystems();
    await SystemsPage.getSubsystem($('#system-element-
00003'));

  });

  it('get active rules', async () => {
    await ActiveRulesPage.getActiveRules($('#system-element-
00016'));
    await ActiveRulesPage.loadActiveRules();
  });
});

const Page = require('../page');

class ActiveRulePage extends Page {
  get activeRuleContainer() {
    return $('#active-rules-container');
  }

  get activeRuleLink() {
    return $('#active-rule-link');
  }

  async loadActiveRules() {
    try {
      await
this.activeRuleContainer.waitForDisplayed({timeout: 2000});
    } catch (e) {
      throw new Error('Не удалось загрузить активные
правила');
    }
  }

  async getActiveRules(element) {
    await element.click();
    await this.activeRuleLink.click();
  }
}

```

```

    async addEvent(name, context, component) {

    }

    open() {
        return super.open('');
    }
}

```

```
module.exports = new ActiveRulePage();
```

Листинг Ж.2 – Тесты на модуль редактора активных правил

```

const SystemsPage = require('../pages/systems.page');
const LoginPage = require('../pages/login.page');
const ActiveRulesPage = require('../pages/activerule.page');
const EditorActiveRulesPage = require('../pages/editor-activerule.page');

describe('Editor active rules page test', () => {
    it('login', async () => {
        await LoginPage.open();
        await LoginPage.login('admin@admin.com', 'admin');

        await SystemsPage.loadSystems();
    });

    it('get subsystems', async () => {
        await SystemsPage.loadSystems();
        await SystemsPage.getSubsystem($('#system-element-00003'));
    });

    it('get active rules', async () => {
        await ActiveRulesPage.getActiveRules($('#system-element-00016'));
        await ActiveRulesPage.loadActiveRules();
    });

    it('get editor active rule', async () => {
        await
        EditorActiveRulesPage.getEditorActiveRule($('#active-rule-element-00032'));
        await EditorActiveRulesPage.loadEditorActiveRule();
    });
});

const Page = require('../page');

class EditorActiveRulePage extends Page {
    get editorActiveRuleContainer() {

```

```

        return $('#editor-active-rules-container');
    }

    async loadEditorActiveRule() {
        try {
            await
this.editorActiveRuleContainer.waitForDisplayed({timeout: 2000});
        } catch (e) {
            throw new Error('Не удалось загрузить редакторы
активных правил');
        }
    }

    async getEditorActiveRule(element) {
        await element.doubleClick();
    }

    open() {
        return super.open('');
    }
}

module.exports = new EditorActiveRulePage();

```

Листинг Ж.3 – Тесты на создание элемента графа и связи

```

const Page = require('./page');
class EditorActiveRulePage extends Page {
    get editorActiveRuleContainer() {
        return $('#editor-active-rules-container');
    }
    get addButton() {
        return $('#add-action');
    }
    get inputAction() {
        return $('#action-input-name');
    }
    get selectEvent() {
        return $('#action-select-event');
    }
    get saveActionButton() {
        return $('#save-action-button');
    }
}

```

```

    }
    get addEdgeButton() {
        return $('#add-edge');
    }
    get selectEventForEdge() {
        return $('#edge-select-event');
    }
    get selectConditionForEdge() {
        return $('#edge-select-condition');
    }
    get selectActionForEdge() {
        return $('#edge-select-action');
    }
    get saveEdgeButton() {
        return $('#save-edge-button');
    }
    async loadEditorActiveRule() {
        try {
            await
this.editorActiveRuleContainer.waitForDisplayed({timeout: 2000});
        } catch (e) {
            throw new Error('Не удалось загрузить редакторы
активных правил');
        }
    }
    async getEditorActiveRule(element) {
        await element.doubleClick();
    }
    async addAction(name, condition) {
        await this.addButton.click();
        await this.inputAction.setValue(name);
        await this.selectEvent.click();
        const conditionOption = await $(div=${condition});
        await $(conditionOption).click();
    }

```

```

        await this.saveActionButton.click();
    }
    async addEdge(event, condition, action) {
        await this.addEdgeButton.click();
        await this.selectEventForEdge.click();
        const eventOption = await $(div=${event});
        await $(eventOption).click();
        await this.selectConditionForEdge.click();
        const conditionOption = await $(div=${condition});
        await $(conditionOption).click();
        await this.selectActionForEdge.click();
        const actionOption = await $(div=${action});
        await $(actionOption).click();
        await this.saveEdgeButton.click();
    }
    open() {
        return super.open('');
    }
}

module.exports = new EditorActiveRulePage();
const SystemsPage = require('../pages/systems.page');
const LoginPage = require('../pages/login.page');
const ActiveRulesPage = require('../pages/activerule.page');
const EditorActiveRulesPage = require('../pages/editor-activerule.page');
describe('Editor active rules page test', () => {
    it('login', async () => {
        await LoginPage.open();
        await LoginPage.login('admin@admin.com', 'admin');
        await SystemsPage.loadSystems();
    });
    it('get subsystems', async () => {
        await SystemsPage.loadSystems();
    });
});

```

```

        await SystemsPage.getSubsystem($('#system-element-
00003'));
    });
    it('get active rules', async () => {
        await ActiveRulesPage.getActiveRules($('#system-element-
00016'));
        await ActiveRulesPage.loadActiveRules();
    });
    it('get editor active rule', async () => {
        await
EditorActiveRulesPage.getEditorActiveRule($('#active-rule-element-
00047'));
        await EditorActiveRulesPage.loadEditorActiveRule();
    });
    it('add action', async () => {
        await EditorActiveRulesPage.addAction('Новое действие',
'Условие активного правила');
    });
    it('add edge', async () => {
        await EditorActiveRulesPage.addEdge('Измерение влажности -
АЕ', 'Не выбрано', 'Новое действие - А');
    });
});

```