

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 управления метаданными для
сервиса потоковой обработки событий**

Студент _____ Бителева Анастасия Олеговна
(подпись, дата) (ФИО полностью)

Руководитель _____ Шибанов С.В.
(подпись, дата) (фамилия, инициалы)

Нормоконтролер _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

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

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

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

Секретарь ГЭК _____ Попова Н.А.
(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 Бителева Анастасия Олеговна гр. 19ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Программные средства управления метаданными для сервиса
поточковой 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 управления метаданными для потоковой
обработки событий*

2) *Среда выполнения: Mac OS*

Языковые средства разработки: TypeScript

Инструментальные средства: WebStorm, Google Chrome, Docker.

Технологии разработки ООП, клиент-ориентированный подход.

3) *Основные функции разрабатываемого программного обеспечения:*

а) взаимодействие с системами и подсистемами, их управление;

б) отслеживания и управление событиями, привязка их к подсистемам;

в) управление метаданными;

г) анализ корректности метаданных;

д) взаимодействие с серверным приложениями;

е) управление активными правилами, их отслеживание, создание, удаление;

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 152 листа, 54 рисунка, 10 таблиц, 15 использованных источников и 8 приложений.

ПРОГРАММНЫЕ СРЕДСТВА УПРАВЛЕНИЯ МЕТАДААННЫМИ ДЛЯ СЕРВИСА ПОТОКОВОЙ ОБРАБОТКИ СОБЫТИЙ, АКТИВНЫЕ ПРАВИЛА, СОБЫТИЯ, WEB-ПРИЛОЖЕНИЕ, JAVASCRIPT, TYPESCRIPT, HTML, REACT JS, UML, SINGLE PAGE APPLICATION, WEBDRIVER IO.

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

Цель выпускной квалификационной работы – разработать программные средства управления метаданными для сервиса потоковой обработки событий.

Технологии разработки – разработка проводилась на языках программирования JavaScript и TypeScript с использованием фреймворка React JS. В качестве среды разработки использовался редактор программного кода Webstorm.

Среда исполнения – macOS, web-браузер.

Результаты работы – в ходе выполнения выпускной квалификационной работы были разработаны программные средства управления метаданными для сервиса потоковой обработки событий.

Все требования к разработке приложения соблюдены.

					ПГУ 09.03.04 – 08BP191.04 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Программные средства управления метаданными для сервиса потоковой обработки событий Пояснительная записка	Лит.	Лист	Листов
Разраб.		Бителева А.О.						
Пров.		Шибанов С.В.					3	152
Н. Контр.		Такташкин Д.В.				Группа 19ВП1		
Утв.		Козлов А.Ю.						

Содержание

Введение	7
1 Анализ предметной области и постановка задачи на разработку.....	8
1.1 Сервис потоковой обработки событий и исполнения активных правил	8
1.1.1 Назначение и функциональные возможности сервиса потоковой обработки событий и исполнения активных правил.....	8
1.1.2 Архитектура сервиса потоковой обработки событий и исполнения активных правил.....	12
1.1.3 Схема работы сервиса потоковой обработки событий и исполнения активных правил.....	15
1.2 Метаданные сервиса потоковой обработки событий и исполнения активных правил	16
1.2.1 Структура метаданных сервиса.....	16
1.2.2 Жизненный цикл метаданных сервиса	18
1.2.3 Задачи управления метаданными сервиса	19
1.3 Постановка задачи на разработку программных средств.....	21
1.3.1 Цели разработки и назначение программных средств.....	21
1.3.2 Классы пользователей программных средств	22
1.3.3 Перечень автоматизируемых функций.....	24
1.4 Планирование и функционально-стоимостной анализ разработки программных средств	26
2 Анализ требований на разработку программных средств.....	34
2.1 Функциональные требования к программным средствам.....	34
2.2 Требования к архитектуре программных средств	35
2.3 Требования к пользовательскому интерфейсу программных средств	38
2.4 Обоснование выбора технологий и средств разработки.....	39
2.5 Требования к аппаратно-программному обеспечению.....	42
2.5.1 Требования к аппаратно-техническому обеспечению	42
2.5.2 Требования к системному программному обеспечению	42

2.5.3	Требования к прикладному программному обеспечению.....	43
3	Разработка программных средств	44
3.1	Разработка архитектуры развертывания программных средств управления метаданными для сервиса потоковой обработки событий	44
3.2	Разработка пользовательского интерфейса.....	48
3.3	Разработка программных средств регистрации и авторизации пользователей.....	54
3.4	Разработка программных средств просмотра систем, подсистем и событий выбранной системы.....	55
3.5	Разработка программных средств управления активных правил	58
3.6	Разработка программных средств списочного конструирования.....	62
4	Контроль качества программных средств.....	64
4.1	Планирование процесса контроля качества программных средств	64
4.1.1	Мероприятия по контролю качества программных средств	64
4.1.2	Обоснование выбора модели тестирования.....	65
4.1.3	Обоснование выбора метрик программного кода	66
4.2	Тестирование программных средств	67
4.2.1	Функциональное тестирование	67
4.2.2	Конфигурационное тестирование	68
4.2.3	Модульное тестирование	70
4.3	Оценка качества программного кода.....	71
4.3.1	Метрики устойчивости и абстрактности.....	71
4.3.2	Метрика Холстеда	75
4.3.3	Оценка качества пользовательского интерфейса	80
	Заключение	83
	Список используемых источников	85
	Приложение А. Глоссарий	87
	Приложение Б. Листинг моделей систем и событий	89
	Приложение В. Листинг модели активных правил	106
	Приложение Г. Пример системы для тестирования приложения.....	124

Приложение Д. Результаты функционального тестирования	128
Приложение Е. Результаты конфигурационного тестирования.....	133
Приложение Ж. Результаты модульного тестирования.....	142
Приложение И. Листинг модульных тестов	145

Введение

Современные информационные технологии неразрывно связаны с обработкой огромных объемов данных и эффективным управлением информационными потоками. Данные могут поступать из множества различных источников, таких как социальные сети, устройства Интернета вещей, датчиков на производственных предприятиях и прочего. Эти данные, зачастую, генерируются в режиме реального времени, что создает необходимость в их мгновенной обработке и анализе. Именно здесь на помощь приходит потоковая обработка данных, позволяющая проводить анализ и обработку информации по мере ее поступления и обнаруживать важные события в реальном времени. Для оперативной реакции на эти события используются механизмы активных правил.

Сервис потоковой обработки событий получает данные, соответствующие элементарным событиям, от внешних наблюдаемых систем. На основе полученных данных могут с применением соответствующих алгоритмов обнаруживаются сложные, агрегированные и составные, события. Сервис может помочь предприятиям и организациям управлять огромными объемами данных, которые они генерируют, управлять бизнес-процессами, своевременно обнаруживать неисправности и собирать статистические данные о работе конкретных элементов инфраструктуры.

Сервис потоковой обработки событий является эффективным инструментом для обработки, анализа и представления данных. Однако, в контексте развития сервиса, возникает необходимость эффективного управления метаданными, которые описывают структуру и свойства получаемых данных.

Цель данной дипломной работы заключается в создании клиентского приложения для управления метаданными сервиса потоковой обработки событий.

Разработка этого инструментария включает в себя анализ требований, проектирование, реализацию и тестирование программного обеспечения.

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

Пропорционально ежегодному увеличению объемов данных растет и сложность их обработки. От множества сторонних наблюдаемых систем, называемых также промышленным интернетом вещей (IIoT), постоянно поступает поток событий. Информация может генерироваться базой данных или различными техническими объектами. Эти данные получает сервис потоковой обработки событий, основанный на активных правилах. Для обработки используется механизм активных правил, позволяющий корректно реагировать на события и их контекст.

Модель Event-Condition-Action (ECA) является основой работы сервиса потоковой обработки событий и позволяет системе анализировать и реагировать на события в режиме реального времени в соответствии с заранее определенными условиями и действиями. Модель состоит из трех основных составляющих [1]:

- событийная часть: в этой части система принимает сигналы, поступающие от сторонних наблюдаемых систем, события которых являются входными данными для сервиса и содержат специализированный контекст, необходимый для последующей обработки;
- часть условия: этот логический блок оценивает условия и критерии, определенные заранее, при выполнении которых запускаются предопределенные действия, и позволяет определить, когда и какие действия должны быть выполнены на основе полученных событий;
- часть действий: здесь определены различные операции и действия, которые выполняются взаимодействием с наблюдаемой системой, где

в зависимости от условий и контекста событий, сервис может инициировать различные операции, изменения состояния или передавать команды другим системам для выполнения соответствующих действий.

Таким образом, сервис потоковой обработки событий, благодаря основе на модели ЕСА, предоставляет из себя эффективный инструмент для обработки потоков данных и реагирования на них в реальном времени. Благодаря этому сервис применим в различных отраслях, включая промышленность, здравоохранение, розничную торговлю, финансы и не только. Сервис открывает возможности для создания масштабных систем, включая цифровые дубликаты сложного промышленного оборудования или целых предприятий.

Сервис потоковой обработки событий находит применение в различных сферах, включая:

- промышленность: на промышленных предприятиях сервис используется для контроля производственных линий и позволяет наблюдать и анализировать данные с множества источников, а затем принимать соответствующие решения и реагировать на возникающие события в режиме реального времени;

- здравоохранение: в медицинских учреждениях сервис применяется для предупреждения врачей о состоянии пациентов и способствует мониторить и обрабатывать медицинские данные, сигнализируя о возможных проблемах или изменениях в здоровье пациентов, что позволяет медицинскому персоналу принимать своевременные меры;

- системы «Умный дом»: в таких системах сервис активно используется для создания комфортной и безопасной обстановки, когда соответствующие датчики обнаруживают события, сервис выполняет предварительно заданные действия, упрощающие повседневную жизнь или обеспечивающие безопасность помещения, например, при обнаружении активности у входа в помещение, сервис анализирует метаданные события и может автоматически включить свет.

Таким образом, сервис потоковой обработки событий демонстрирует свою эффективность и востребованность в различных областях, где требуется оперативная обработка и анализ данных, а также реагирование на события в режиме реального времени.

Функциональные возможности сервиса направлены на эффективное распределение потока информации между бизнес-доменами или организациями и динамическое сотрудничество нескольких служб. При этом механизм активных правил обеспечивает не только быстрое реагирование, но и снижает негативное влияние на существующие прикладные системы.

Сервис способен извлекать значимые события для наблюдаемой системы и обеспечивать динамическую реакцию на бизнес-приложения или службы в режиме реального времени. В рамках сервиса используются три группы репозитория метаданных [2]:

а) глобальные репозитории содержат информацию о всех наблюдаемых системах, событиях, активных правилах, пользователях и конфигурационных данных компонентов сервиса и обеспечивают централизованное хранение и доступ к основной информации, необходимой для работы сервиса;

б) оперативные хранилища компонентов содержат информацию, специфичную для каждого отдельного компонента системы и содержат данные, необходимые для непосредственной работы компонента, и обеспечивают быстрый доступ к соответствующей информации;

в) репозиторий журнализации предназначен для хранения информации о работе сервиса, включает журналы, которые записывают и хранят сведения о выполненных операциях, событиях и других важных аспектах функционирования сервиса, а также позволяет осуществлять отслеживание и анализ работы сервиса, а также восстановление информации в случае необходимости.

Благодаря данным репозиториям сервис потоковой обработки событий предоставляет не только мощные функциональные возможности, но и обладает

надежной архитектурой, обеспечивающей эффективное управление и использование метаданных в рамках работы с системой.

На рисунке 1 представлена диаграмма вариантов использования сервиса потоковой обработки событий [3].

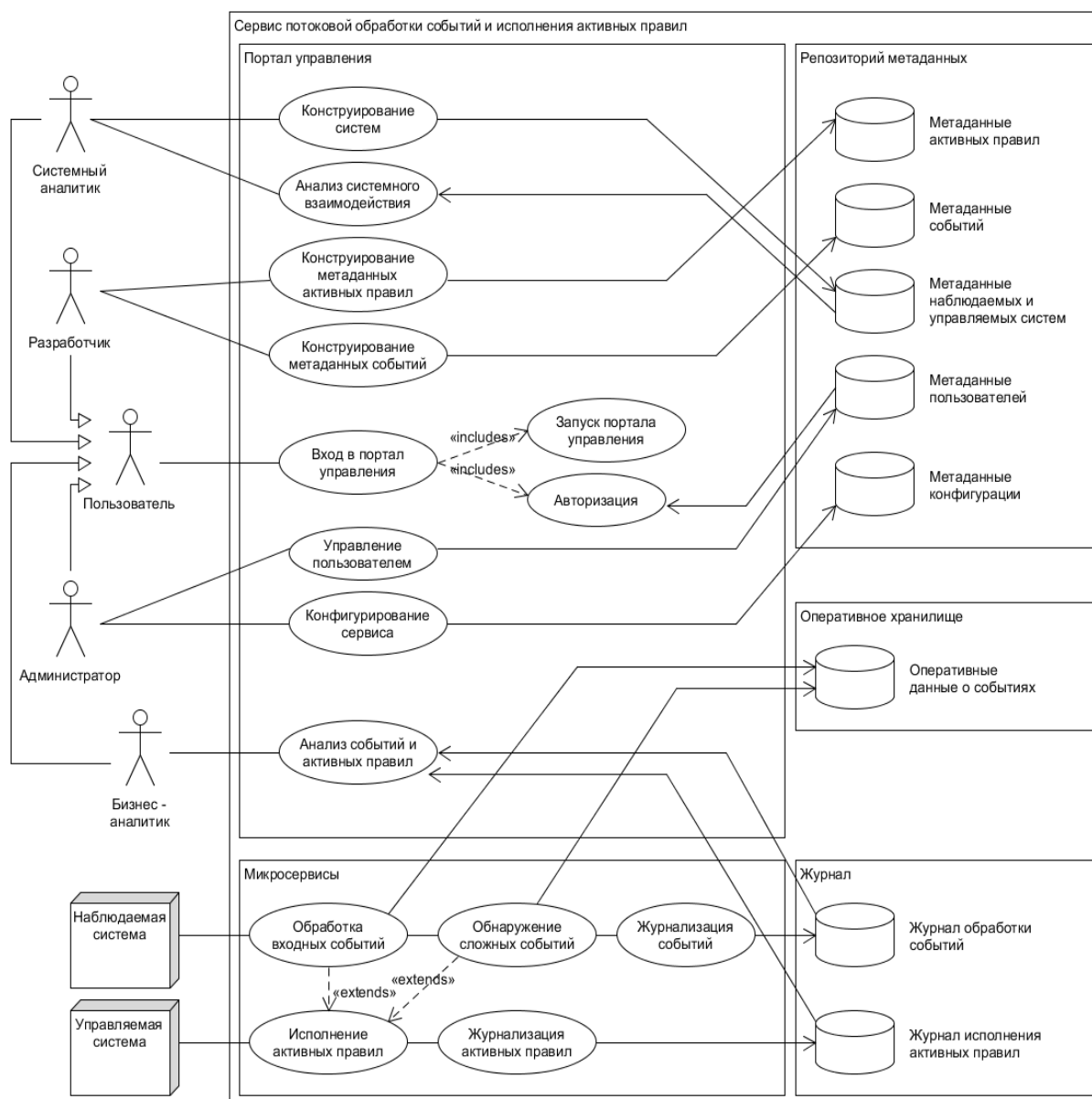


Рисунок 1 – Диаграмма вариантов использования сервиса потоковой обработки событий

Функциональные возможности сервиса потоковой обработки событий включают следующее [4]:

- мониторинг и обработка входных событий от объектов, обнаружение сложных и агрегированных событий на основе заданных шаблонов, доставка событий соответствующим компонентам сервиса;
- создание активных правил через управляющий портал, настройка условий активации и входящих событий;
- управление созданными активными правилами, проверка их корректности, отключение и подключение к системам управления;
- исполнение статических и динамических алгоритмов анализа активных правил и событий, поиск цикличности исполнения, выявление коллизий, моделирование новых событий, на основе проанализированных данных;
- вычисление условий активного правила и дальнейшее исполнение активных правил, в зависимости от поступивших событий;
- журнализация обрабатываемых событий и исполнения активных правил, последующее предоставление данных для анализа;
- поддержание актуальности метаданных во всех компонентах сервиса, синхронизация глобального и локальных репозиторий метаданных;
- масштабирование и настройка сервиса с учётом данных журнализации и результатов анализа.

Для правильного функционирования требуется создание и управление учетными записями пользователей, а также осуществление процессов регистрации и аутентификации. Предоставление доступа к операциям сервиса осуществляется на основе назначенных ролей пользователям.

1.1.2 Архитектура сервиса потоковой обработки событий и исполнения активных правил

Архитектура данного сервиса потоковой обработки событий основана на принципах микросервисной архитектуры и событийно-ориентированном подходе. Это обеспечивает ряд преимуществ, включая простую и удобную

масштабируемость, возможность работы в режиме реального времени, отказоустойчивость и высокую доступность.

Для доставки событий в сервис используются очереди и брокер сообщений, который выполняет сбор и маршрутизацию сообщений в соответствии с предопределенной логикой. Брокер сообщений не анализирует содержимое передаваемых данных, а служит в качестве посредника между компонентами системы.

Архитектура позволяет обеспечить автономность каждого компонента системы, разделяя логику независимых элементов друг от друга. Благодаря этому, компоненты могут быть написаны на разных языках программирования и работать на различных платформах, а взаимодействие между ними настраивается с помощью брокера сообщений. Такой подход также обеспечивает независимую масштабируемость микросервисов, позволяя масштабировать только те сервисы, которые испытывают наибольшую нагрузку. Выход из строя одного компонента не оказывает критического влияния на работу всего сервиса.

Для взаимодействия с клиентскими приложениями, настройки сервиса и управления метаданными предусмотрен REST-сервер. Он обеспечивает отправку и прием сообщений от каждого микросервиса, а также управление глобальным репозиторием метаданных.

Для обеспечения функционирования сервиса необходим центральный репозиторий, в котором хранятся метаданные. В состав набора метаданных входят различные элементы, такие как спецификации событий и активных правил, информация о наблюдаемых системах и объектах, данные о системах управления, промежуточные результаты работы, правила обработки событий, правила формирования управляющих воздействий, результаты анализа событий и правил, а также множество других информационных элементов. Центральный репозиторий играет важную роль в хранении и управлении этими метаданными в процессе работы сервиса [5].

Архитектура сервиса потоковой обработки событий представлена на рисунке 2.

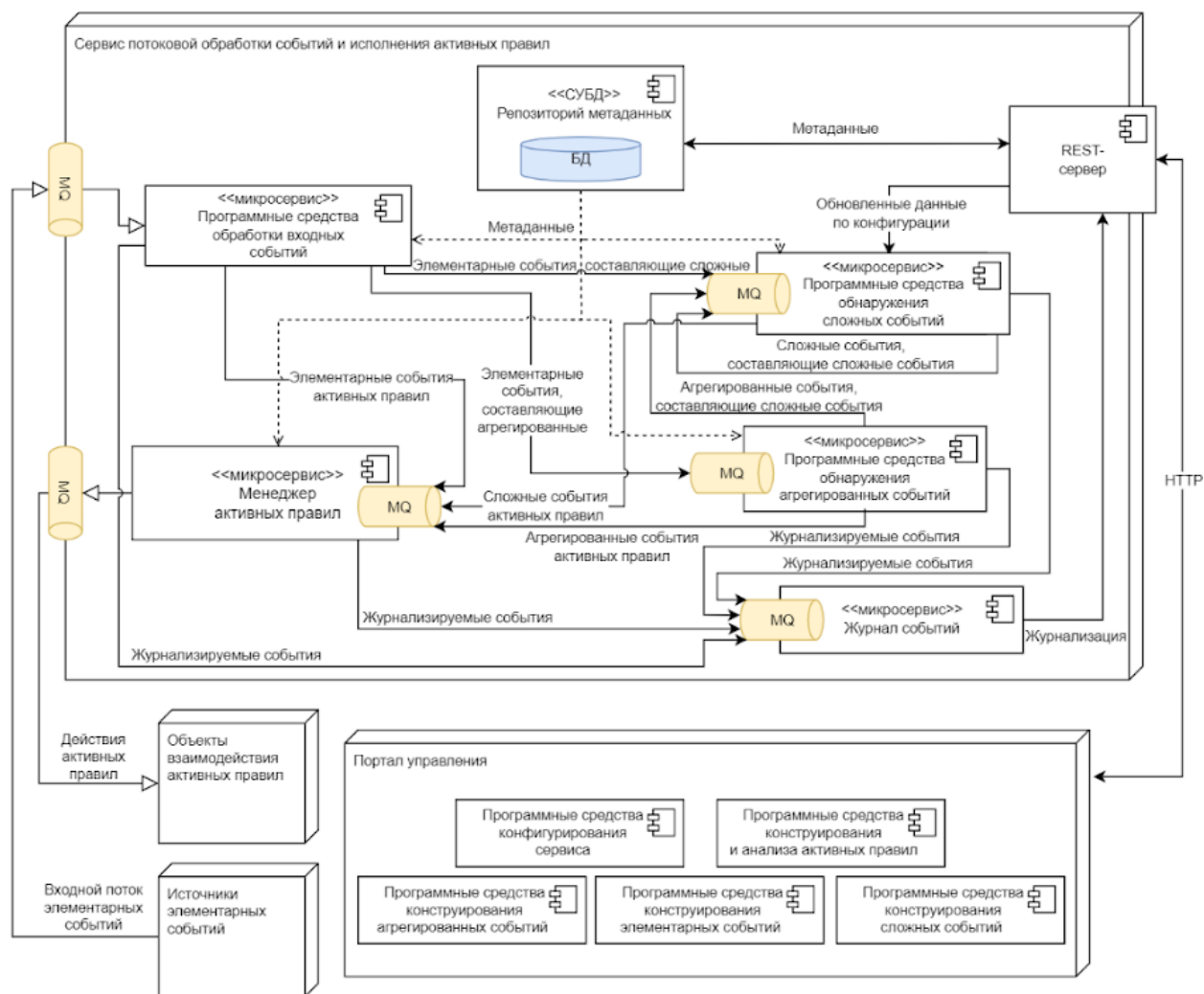


Рисунок 2 – Архитектура сервиса потоковой обработки событий

Микросервисы внедряют концепцию «Database Per Service» («База данных на сервис»), при которой у каждого сервиса имеется свой собственный локальный репозиторий. Это повышает отказоустойчивость и обеспечивает автономность каждого сервиса в отдельности. Синхронизация локальных репозиториях с глобальным решается с помощью специального сценария, который обеспечивает минимальное время ожидания запросов к метаданным без ущерба для их актуальности и с минимальной задержкой в работе сервиса. Это позволяет достигнуть оптимальной скорости работы сервиса при обеспечении соответствующей согласованности данных между локальными и глобальными репозиториями [6].

1.1.3 Схема работы сервиса потоковой обработки событий и исполнения активных правил

Сервис потоковой обработки событий и исполнения активных правил непрерывно обрабатывает поток событий в реальном времени. Главная задача состоит в оперативном определении событий и реакции на них для обеспечения постоянного и своевременного взаимодействия с окружением. Для поддержания требуемых характеристик системы применяются механизмы активных правил и промежуточные сервисы пересылки данных.

В системе размещены микросервисы, которые используют созданные пользователями метаданные для работы с потоками событий от наблюдаемых систем. Они обрабатывают входящие события, определяют появление событийных шаблонов и активируют соответствующие активные правила. Результаты работы микросервисов журналируются и предоставляются аналитикам для последующего анализа.

Сгенерированный поток событий от компонентов-источников поступает на вход сервису. Микросервис по обработке входных событий проверяет корректность данных и с помощью механизма обнаружения сложных событий ищет совпадения с заранее заданными событийными шаблонами. При обнаружении шаблона генерируется сложное событие, которое также направляется в поток для дальнейшей обработки. Затем все события передаются на вход менеджера активных правил, который активирует соответствующие правила при выполнении заданных условий. При активации активного правила система реагирует на появление события в потоке, отправляя управляющие сигналы управляемым объектам [7].

Разработка разнообразных алгоритмов, в том числе в контексте предприятий, обеспечивает возможность удаленного контроля над процессом производства, предварительное выявление неисправностей и сбор статистических данных о работе конкретных узлов. Этот инструмент эффективно управляет жизненным циклом системы.

Для обработки событий используется технология ESP (Event Stream Processing), которая позволяет обнаруживать часто встречающиеся шаблоны поведения системы, выявлять циклические сценарии, коллизии, иерархию событий и другие проблемные ситуации [8]. В ходе потоковой обработки событий сервис автоматически реагирует на заданные ситуации, такие как отказы оборудования или угрозы пожарной безопасности. Активное внедрение подобных технологий на предприятиях позволило бы осуществлять обслуживание оборудования на основе его фактического состояния, вместо привязки к жестко регламентированным срокам ремонтных работ, что привело бы к существенному сокращению расходов.

1.2 Метаданные сервиса потоковой обработки событий и исполнения активных правил

1.2.1 Структура метаданных сервиса

Информация об активных правилах, событиях и компонентах, от которых они поступают, формируют основу обрабатываемого набора метаданных. Единый репозиторий является хранилищем для метаданных, связанных с активными правилами и событиями.

Также имеются метаданные, связанные с конкретной системой, в которой автоматизировано управление процессами и жизненным циклом. Основными метаданными такой системы могут быть следующие:

- общая структура ответов для моделей поведения;
- возможности доступа тех или иных пользователей к функционалу просмотра и управления;
- иерархические отношения с прочими системами.

События во все системы поступают из набора компонентов-источников. Метаданные компонентов включают в себя:

- список ролей, которые раскрывают функциональные возможности компонента;

- уровень компонента, отражающий приоритетность поступающих событий от этого компонента;
- разнообразные статистические данные;
- связь с другими системами, отслеживающими события.

В сервисе события представляются с помощью трех кортежей: <ID, данные, временная метка>. Каждый кортеж содержит уникальный идентификатор устройства (ID), данные, относящиеся к наблюдаемым значениям разных типов, а также временную метку, указывающую на время генерации события соответствующим устройством.

Сервис работает с тремя разными видами событий:

- простые (атомарные);
- агрегированные – такие события имеют определенный шаблон агрегации;
- сложные (комплексные) – события этого типа состоят из набора других событий.

Необходимо обязательно устанавливать ассоциации между активными правилами и событиями. Эти связи могут быть двух видов: «От события к активному правилу» и наоборот «От активного правила к событию». В целях правильной и эффективной идентификации взаимодействия, сервису нужно иметь данные о том, какое событие порождает активное правило и наоборот.

Практически во всех сервисах с большим потоком пользователей содержатся метаданные этих пользователей. Разрабатываемый программный продукт не стал исключением. В сервисе тоже имеется информация о данных учетных записей и ролей доступа.

В связи с этим, стоит отметить важность обеспечения безопасности личных данных пользователей, особенно, паролей. Для этого их хранение осуществляется в зашифрованном виде. Процесс шифрования может происходить с использованием криптографических хеш-функций для создания

ключей. Такой подход обеспечивает безопасное хранение пользовательских данных и их аутентификацию во время взаимодействия с сервисом.

1.2.2 Жизненный цикл метаданных сервиса

У разрабатываемого сервиса есть четыре основных уровня функциональности:

- управления метаданными;
- разработки алгоритмов и систем взаимодействия;
- обработки событий и исполнения активных правил;
- публикации/подписки бизнес-приложений.

С помощью REST-сервера реализуется управление метаданными. Благодаря серверу вовремя происходит передача новых или обновленных метаданных микросервисам обработки событий и выполнения активных правил. REST-сервер также предоставляет средства для создания новых метаданных. Процесс создания может инициировать администратор, или же он может происходить автоматически при публикации или подписке новых бизнес-приложений.

Алгоритмы обработки событий и исполнения активных правил разрабатываются с использованием портала управления. Этот портал взаимодействует с REST-сервером по протоколу HTTP с целью получения необходимых метаданных для создания активных правил и объявления новых событий. Портал управления предоставляет возможность создавать и настраивать и сложные, и агрегированные события. Кроме того, есть инструменты объявления новых правил, редактирования уже существующих, а также определения разных связей между событиями, правилами, системами.

Все микросервисы в архитектуре сервиса потоковой обработки событий и исполнения активных правил выполняют индивидуальные задачи и оперируют собственными наборами метаданных. Такие уникальные наборы метаданных доставляются элементам архитектуры при помощи брокера сообщений. В процессе обработки событий и исполнения активных правил, микросервисы

могут отправлять сообщения бизнес-приложениям с использованием механизма подписчик/читатель. Команды управления могут быть использованы как автоматически генерируемые события от самого сервиса и направлены на соответствующие датчики или устройства.

В конечной стадии обработки событий соответствующие сервисы осуществляют журнализацию. Наличие этого процесса обеспечивает предоставления статистических данных и выявление проблемных участков в текущих алгоритмах. Критичными могут оказаться, например, коллизии или заикливания.

1.2.3 Задачи управления метаданными сервиса

Сервис потоковой обработки событий и исполнения активных правил взаимодействует с многопользовательским порталом управления с помощью REST API. Такая архитектура идеальна для многопользовательских систем. Информацию о пользователях, их аккаунтах и ролях хранится в единой базе данных [9].

В работе сервиса активно применяется JWT. JSON Web Token является открытым стандартом (RFC 7519) для создания токенов доступа, основанным на формате JavaScript Object Notation (JSON). JWT применяется в данном случае для аутентификации и передачи метаданных пользователя. Токены создаются и подписываются секретным ключом на сервере, после чего передаются клиенту. В работе используются два токена: Access и Refresh [10].

В процессе управления метаданными выделяются такие ключевые аспекты, как формирование, внесение изменений, обеспечение контроля данных, а также определение оптимальных наборов метаданных, необходимых для эффективной работы каждого из микросервисов.

На рисунке 3 показан процесс управления метаданными в виде диаграммы деятельности.

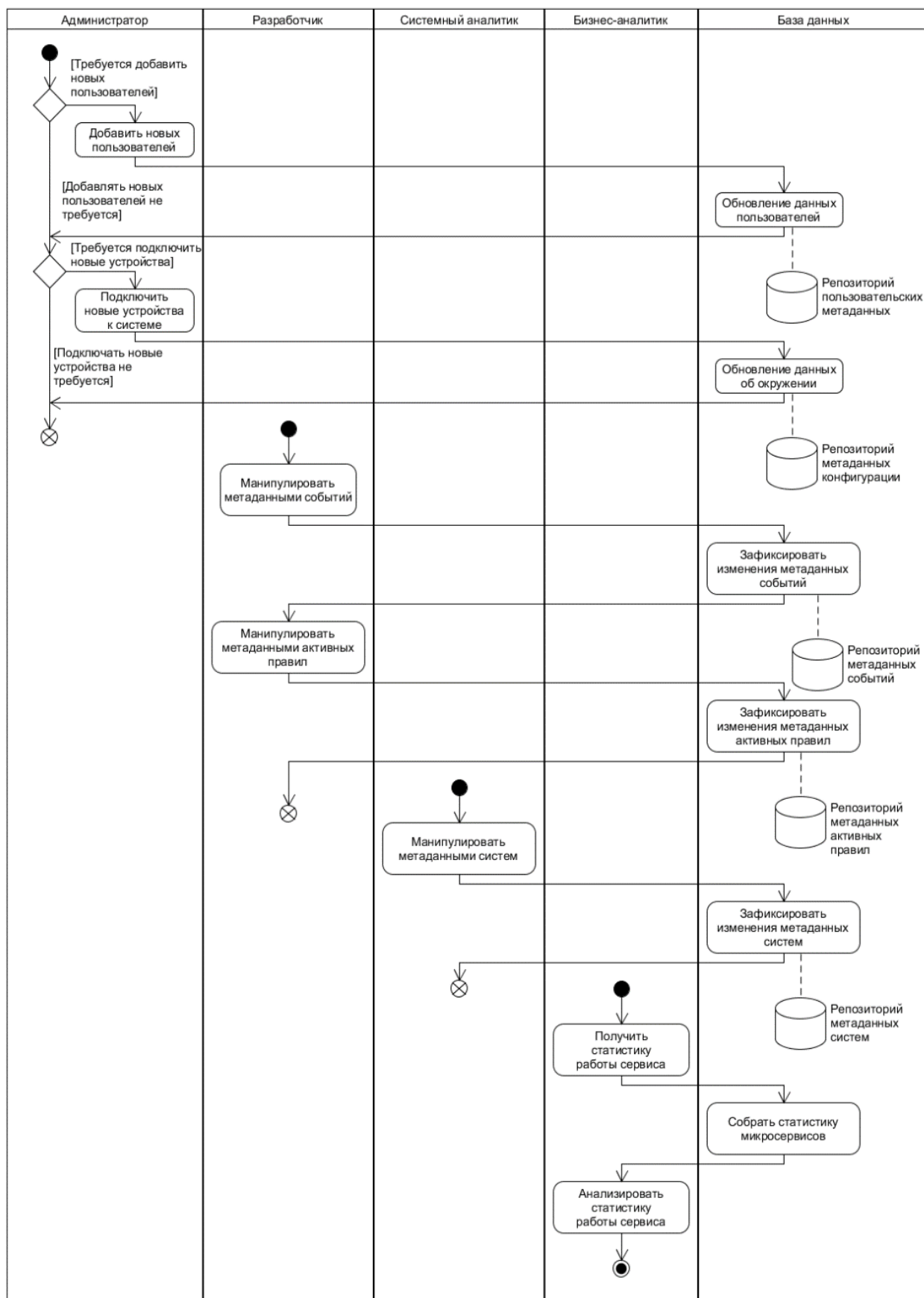


Рисунок 3 – Процесс управления метаданными в виде диаграммы деятельности

Реализация данных процессов осуществляется при помощи инструментов серверного программного обеспечения. Управление метаданными представляет собой комплекс операций, основанных на принципе CRUD (создание, чтение, обновление и удаление), и направлено на обеспечение целостности и надежности хранимых данных.

Работа с метаданными требует использование алгоритмов для первичной проверки данных при их создании и редактировании. Запрещено создание повторяющихся событий или активных правил, так как это может негативно сказаться на агрегации запросов и привести к нежелательным последствиям, например, к дублированию данных, полученных от источника.

Для обеспечения контроля над логикой исполнения активных правил, сервис использует механизмы управления метаданными и запускает алгоритмы обнаружения нарушений. Например, в случае, если метаданные были изменены или удалены из системы, могут возникнуть проблемы в обработке событий и исполнении правил.

1.3 Постановка задачи на разработку программных средств

1.3.1 Цели разработки и назначение программных средств

Прежде чем сервис можно будет использовать напрямую, физически подключать устройства или генерировать поток событий, его необходимо сначала сконфигурировать и настроить. Этот процесс требует обширной подготовки его компонентов, включая отдельные микросервисы, связывающие программные средства и репозитории метаданных.

Помимо этого, при развертывании сервиса для конкретной системы необходимо заполнить ряд баз данных, содержащих управляющие и информационные данные, с которыми впоследствии будут взаимодействовать различные пользователи со знаниями в конкретных областях.

Стоит отметить, что на разработке и запуске сервиса процесс не заканчивается. После этих этапов важно иметь возможность при необходимости

расширять сервис с помощью новых элементов. Также требуется осуществлять мониторинг и анализ состояния и метаданных системы для обнаружения дефектов или ошибок в проектировании, интеграции или использовании сервиса.

Также задачи по работе с сервисом выполняют специалисты, которым не нужно отвлекаться на технические детали реализации функционала. Внедрение отдельного инструментария предоставит такие преимущества как:

- автоматизация многих функций сервиса;
- верификация доступа пользователей к инструментарию сервиса;
- упрощенная модель взаимодействия с сервисом через пользовательский интерфейс;
- упрощенный контроль и мониторинг созданного окружения.

Учитывая возможность использования указанных выше преимуществ, разработка программных средств управления метаданными направлена на создание и предоставление пользователям набора удобных инструментов для манипулирования метаданными системы, а также программных средств с возможностями комплексного мониторинга, контроля и управления созданным окружением.

1.3.2 Классы пользователей программных средств

Для всех процессов управления в сервисе требуются аутентификация и авторизация пользователя. Такой подход реализован для разграничения прав пользователей.

Системному аналитику и разработчику доступны различные функции для работы с метаданными, который может пересекаться и быть в чем-то общим. В то время как у обычного пользователя, не владеющего никакой ролью, функционала не так много и в его возможности включена только авторизация.

На рисунке 4 продемонстрированы основные процессы управления и роли доступа пользователей к данным процессам.

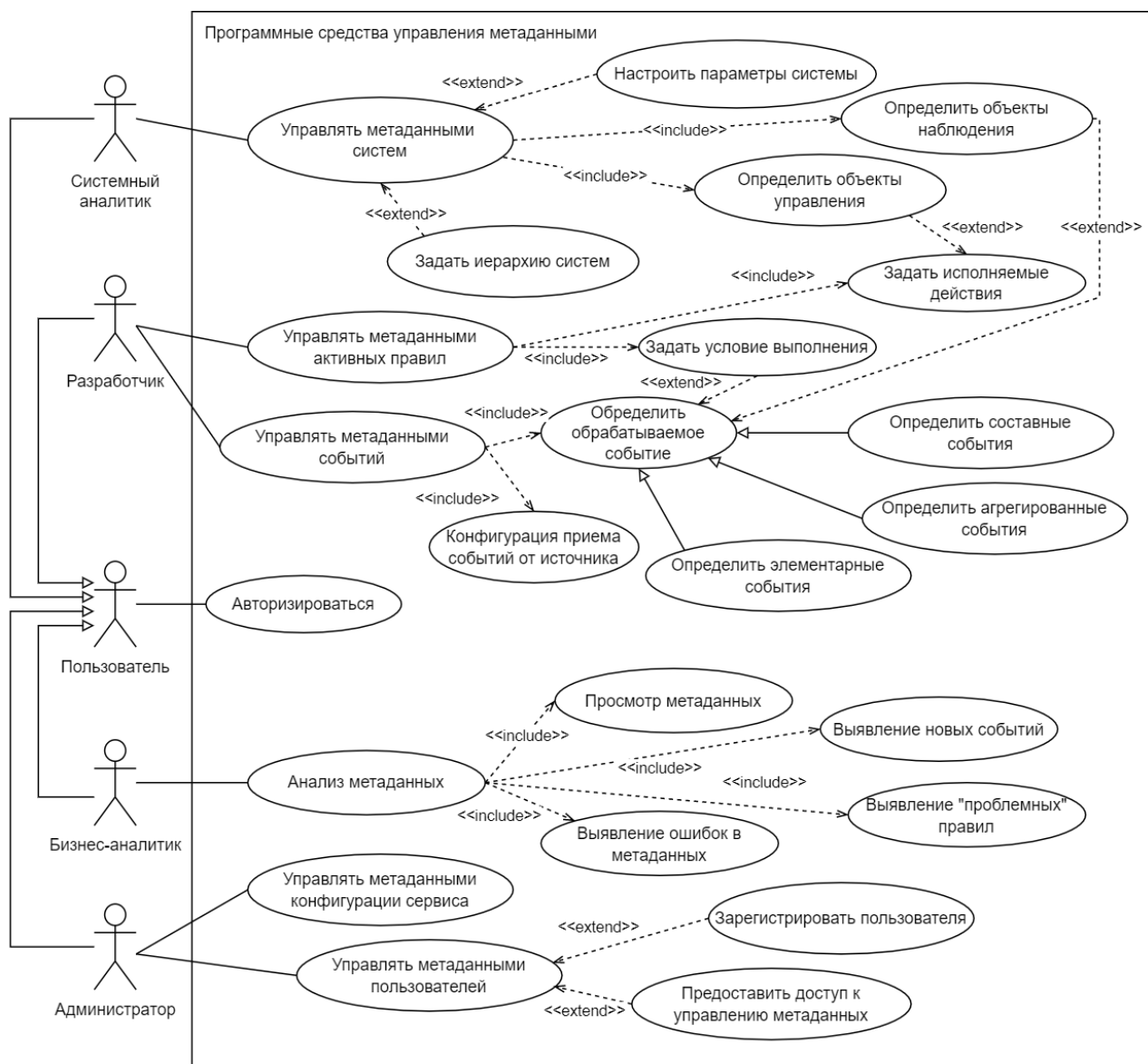


Рисунок 4 – Основные процессы управления и роли доступа пользователей к данным процессам

Можно выделить четыре пользовательских роли:

- а) разработчик: к его возможностям относятся не только конструирование активных правил и событий, но и управление связанных с ними метаданными;
- б) бизнес-аналитик: пользователь с данной ролью может работать со статическими данными касательно произошедших событий и активированных правил, полученных от микросервисов, и на основе полученной информации бизнес-аналитик выявляет ошибки, новые события и «проблемные» правила;

в) системный аналитик: эта роль позволяет настраивать наблюдаемые системы посредством изменения их параметров и определения объектов наблюдения и управления;

г) администратор: во-первых, пользователи с данной ролью могут конфигурировать все компоненты сервиса потоковой обработки событий и исполнения активных правил, а во-вторых, у него есть права на работу с метаданными пользователей, их регистрацию и предоставление доступа к различным интерфейсам управления.

1.3.3 Перечень автоматизируемых функций

На основе анализа назначения инструментария управления метаданными в системе обработки событий был выявлен перечень автоматизируемых функций, реализуемый разрабатываемым приложением. Рассмотрим некоторые из этих функций.

Во-первых, автоматическая генерация запросов к репозиторию метаданных основывается на требованиях, предъявляемых к сервису. Это позволяет эффективно извлекать необходимые данные.

Во-вторых, важным аспектом является контроль безопасности и управление доступом пользователей к механизмам управления метаданными. Предоставление правильного уровня доступа обеспечивает безопасность системы и предотвращает несанкционированный доступ.

Для обеспечения надежного исполнения активных правил и обновления метаданных в системе используется автоматическое оповещение бизнес-приложений и микросервисов о происходящих изменениях. Это позволяет системе оперативно реагировать на изменения и обновления.

Контроль актуальности метаданных в локальных репозиториях каждого элемента архитектуры также является важным аспектом. Это обеспечивает согласованность данных и предотвращает возникновение проблем, связанных с устаревшей или некорректной информацией.

Важным элементом является журнализация операций управления метаданными и ошибок, которые могут возникнуть во время эксплуатации системы. Это позволяет отслеживать выполнение операций и обнаруживать проблемы для последующего анализа и устранения.

Помимо этого, серверные программные средства должны обладать набором защитных механизмов для предотвращения атак злонамеренных сторонних сервисов и программ, а также для обеспечения защиты от несанкционированного доступа.

Кроме функций, связанных с управлением метаданными, рассматриваемые инструментальные средства предоставляют ряд автоматизированных функций на стороне клиента. Например, авторизация пользователя, расшифровка JWT для авторизации, предоставление функционала в зависимости от роли пользователя и сбор информации с полей пользовательского приложения.

Также инструментальные средства позволяют автоматически валидировать данные, упаковывать их в JSON-объекты и обращаться к REST-серверу для выполнения требуемых запросов. Кроме того, происходит анализ ответов от сервера, систематизация и представление информации. Важным аспектом является также автоматическое кеширование запрошенных данных.

Кроме указанных функций, инструментальные средства обладают возможностями по работе с данными на стороне клиента, включая сортировку, фильтрацию, сохранение в локальные хранилища и архивацию части метаданных.

Таким образом, для эффективного управления метаданными в системе обработки событий необходимо обеспечить автоматизированные функции как на серверной, так и на клиентской стороне. Это включает генерацию запросов, контроль доступа, оповещение о изменениях, актуализацию данных, журнализацию операций, обработку ошибок, а также функции авторизации, сбора, валидации и обработки данных на стороне клиента.

1.4 Планирование и функционально-стоимостной анализ разработки программных средств

Планирование играет решающую роль в процессе разработки и последующей поддержки программного продукта, и от его качества зависит эффективность всего проекта. Без тщательного планирования невозможно выполнить работы в оговоренные сроки с требуемым уровнем качества и минимальными затратами.

Основная цель эффективного планирования состоит в определении состава работ и их взаимосвязей, необходимых для успешной реализации проекта. На этом этапе особое внимание уделяется оценке временных рамок для выполнения будущих работ.

Кроме того, структурное планирование включает в себя анализ рисков, определение необходимых ресурсов, а также распределение задач между участниками команды. Важно учесть все факторы, которые могут повлиять на успешное завершение проекта, и принять соответствующие меры заранее.

Индивидуальный подход к каждому проекту позволяет обеспечить оптимальное использование ресурсов, минимизировать риски и достичь высокого уровня эффективности в процессе разработки и поддержки программного продукта.

Таким образом, качественное планирование является фундаментом успешного проекта, обеспечивая его выполнение в установленные сроки, с требуемым качеством и с минимальными затратами. Это включает описание работ, их взаимосвязь, оценку временных рамок и учет всех необходимых факторов для успешной реализации проекта.

На основе тщательного анализа предметной области были выявлены ключевые задачи, которые требуется решить для успешной разработки серверных программных средств. В таблице 1 представлено подробное описание данных задач, а также определены их последовательность и оценочная длительность. В процессе планирования было принято решение о работе в

шестидневной рабочей неделе, учитывая необходимость соблюдения графика и достижения поставленных целей.

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	3 дня	1
4	Изучение и сопоставление аналогичных программных средств	4 дня	1
5	Постановка задачи на разработку инструментальных средств управления метаданными	2 дня	3;4
6	Анализ предметной области и постановка задачи на разработку закончены	-	5
7	Разработка требований	-	-
8	Разработка функциональных требований	2 дня	6
9	Разработка требований к пользовательскому интерфейсу	2 дня	8
10	Разработка требований к архитектуре	4 дня	8
11	Выбор технологий и средств разработки	1 день	10;9
12	Разработка требований к аппаратно-программному обеспечению	2 дня	11
13	Разработка требований закончена	-	12
14	Разработка инструментальных средств управления метаданными	-	-
15	Разработка программных средств взаимодействия с сервером	14 дней	13
16	Разработка пользовательского интерфейса	7 дней	13
17	Разработка программных средств реализации основных автоматизируемых функций	14 дней	13
18	Разработка инструментальных средств управления метаданными закончена	-	17;16;15
19	Тестирование	-	-
20	Выбор методов тестирования	1 день	18
21	Проведение тестирования	4 дня	20

Продолжение таблицы 1

№	Название задачи	Длительность	Предшественники
22	Тестирование закончено	-	21
23	Интеграция с системой обработки событий	-	-
24	Встраивание инструментальных средств в систему обработки событий	1 день	22
25	Тестирование инструментальных средств в качестве микросервиса системы	2 дня	24
26	Интеграция с системой обработки событий закончена	-	25
27	Сдача проекта	-	-
28	Составление документации	7 дней	26
29	Ввод в эксплуатацию	2 дня	28
30	Сдача проекта закончена	-	29
31	Конец проекта	-	30

По итогам анализа, для реализации проекта потребуется выполнить 31 задачу. В таблице 2 представлена детализация затрачиваемых ресурсов.

Таблица 2 – Описание затрачиваемых ресурсов

Название ресурса	Тип	Стандартная ставка	Ставка сверхурочных
Аналитик	Трудовой	120 Р/ч	130 Р/ч
Программист1	Трудовой	140 Р/ч	150 Р/ч
Программист2	Трудовой	140 Р/ч	150 Р/ч
Дизайнер	Трудовой	120 Р/ч	130 Р/ч
Тестировщик	Трудовой	130 Р/ч	140 Р/ч
Компьютер	Материальный	30 000 Р	-
Бумага	Материальный	500 Р	-
Канцелярские принадлежности	Материальный	250 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 Электричество Интернет	50% 50 Р 100 Р
4	Изучение и сопоставление аналогичных программных средств	Аналитик Электричество Интернет	50% 200 Р 300 Р
5	Постановка задачи на разработку инструментальных средств управления метаданными	Аналитик Программист1 Программист2 Электричество Бумага Канцелярские принадлежности	100% 20% 20% 150 Р 1 1
6	Анализ предметной области и постановка задачи на разработку закончены	-	-
7	Разработка требований	-	-
8	Разработка функциональных требований	Программист1 Электричество	100% 100 Р
9	Разработка требований к пользовательскому интерфейсу	Программист2 Электричество	100% 150 Р
10	Разработка требований к архитектуре	Программист1 Электричество	100% 200
11	Выбор технологий и средств разработки	Программист1 Программист2 Электричество	50% 20% 100 Р
12	Разработка требований к аппаратно - программному обеспечению	Программист1 Электричество	100% 150 Р
13	Разработка требований закончена	-	-
14	Разработка инструментальных средств управления метаданными	-	-
15	Разработка программных средств взаимодействия с сервером	Программист1 Электричество Интернет	80% 800 Р 1000 Р

Продолжение таблицы 3

	Название задачи	Ресурс	Единицы
16	Разработка пользовательского интерфейса	Дизайнер Программист2 Электричество Интернет	100% 40% 400 Р 600 Р
17	Разработка программных средств реализации основных автоматизируемых функций	Программист2 Электричество Интернет Программист1	60% 600 Р 1000 Р 20%
18	Разработка инструментальных средств управления метаданными закончена	-	-
19	Тестирование	-	-
20	Выбор методов тестирования	Тестировщик Электричество Интернет	100% 100 Р 300 Р
21	Проведение тестирования	Тестировщик Электричество Интернет	100% 500 Р 200 Р
22	Тестирование закончено	-	-
23	Интеграция с системой обработки событий	-	-
24	Встраивание инструментальных средств в систему обработки событий	Программист1 Электричество Интернет	100% 100 Р 200 Р
25	Тестирование инструментальных средств в качестве микросервиса системы	Тестировщик Электричество Интернет	100% 150 Р 200 Р
26	Интеграция с системой обработки событий Закончена	-	-
27	Сдача проекта	-	-
28	Составление документации	Аналитик Бумага Канцелярские принадлежности Программист1 Электричество	100% 1 1 50% 300 Р
29	Ввод в эксплуатацию	Программист1 Программист2 Электричество Интернет	50% 50% 100 Р 250 Р
30	Сдача проекта закончена	-	-
31	Конец проекта	-	-

Для планирования и анализа временных и финансовых затрат разработки было использовано программное обеспечение Microsoft Project. В данном случае была применена версия Microsoft Project 2007 [11]. С помощью этого инструмента был создан план разработки в виде диаграммы Ганта. Диаграмма Ганта состоит из полос, ориентированных вдоль временной оси. Каждая полоса соответствует отдельной задаче в рамках проекта, а начало и конец работ обозначены на краях полосы.

На рисунке 5 изображена диаграмма Ганта задач с распределением ресурсов.

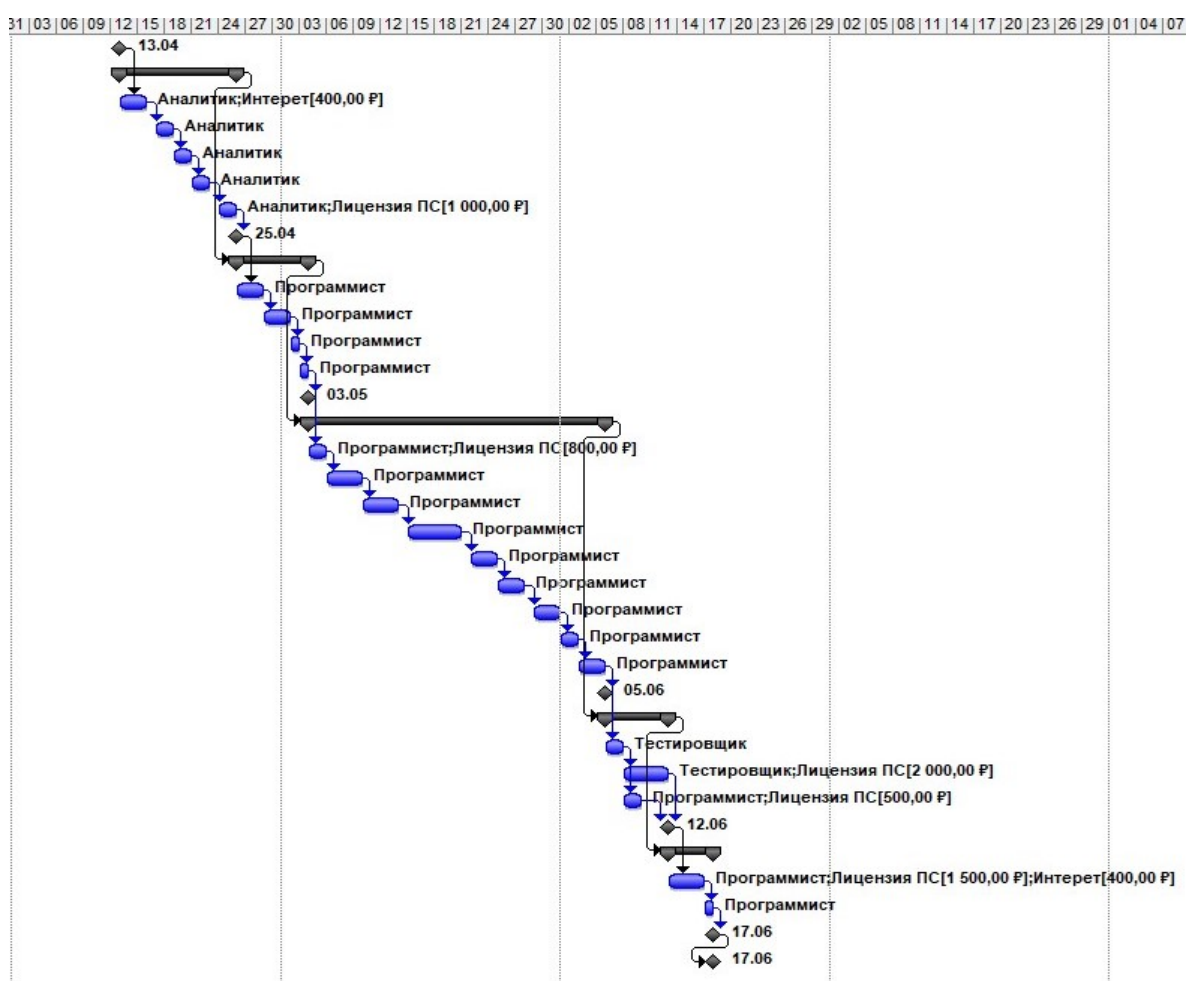


Рисунок 5 – Диаграмма Ганта задач с распределением ресурсов

На представленной диаграмме, изображенной на рисунке 5, были наглядно отражены необходимые ресурсы, основные этапы разработки, а также последовательность и сроки выполнения каждой задачи. При разработке данного

плана было решено работать в рамках шестидневной рабочей недели, учитывая необходимость оптимального использования времени и ресурсов.

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.

После определения основных задач и распределения ресурсов были подготовлены два важных отчета: «Обзор затрат» (рисунок 6), который демонстрирует реальные затраты на реализацию проекта, и «Обзор ресурсов» (рисунок 7), отражающий объем трудозатрат, необходимых для успешной разработки. Эти отчеты играют важную роль в контроле и анализе проекта, позволяя оценить эффективность использования ресурсов и прогнозировать дальнейшие этапы разработки.

Таким образом, диаграмма предоставляет комплексную информацию о необходимых ресурсах, этапах разработки и их последовательности, а также позволяет оценить затраты и трудозатраты, что способствует эффективному управлению проектом и достижению его целей.



Рисунок 6 – Отчет «Обзор затрат»

После проведения анализа и подготовки отчета «Обзор затрат» было установлено, что общая стоимость разработки серверных программных средств составляет 130 200 рублей.

ОБЗОР РЕСУРСОВ

СТАТИСТИКА РЕСУРСОВ

Состояние трудозатрат для всех трудовых ресурсов.

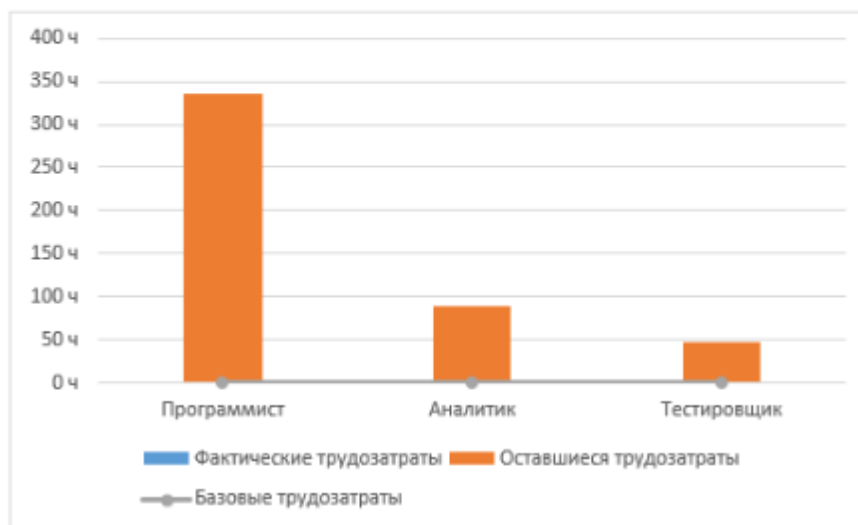


Рисунок 7 – Отчет «Обзор ресурсов»

Учитывая выходные дни, общий срок выполнения всех задач, необходимых для разработки, составил 56 дней.

2 Анализ требований на разработку программных средств

2.1 Функциональные требования к программным средствам

В дополнение к основным механизмам управления, описанным в прошлых главах, разрабатываемое программное средство должно 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;
- з) отображение активных правил:
 - 1) графовом представлении;
 - 2) языковом представлении;
 - 3) списковом представлении;
- и) редактирование активных правил:
 - 1) добавление события;
 - 2) добавление условия;
 - 3) добавление действия;
 - 4) удаление правил.

На основании вышеупомянутых требований была разработана диаграмма вариантов использования, изображенная на рисунке 8. На диаграмме показаны функциональные возможности, которые распределены между различными страницами.

упрощает процесс развертывания и обновления приложения, поскольку нет необходимости устанавливать его на каждом клиентском устройстве отдельно.

Таким образом, выбор реализации клиентской части в виде web-приложения открывает широкие возможности для эффективного использования активных правил и событий, обеспечивая удобство работы и универсальность доступа для пользователей

Клиентская часть веб-приложения должна состоять из трех модулей: модуль для Angular, модуль для React и общий Root модуль, который объединяет оба модуля в единое целое. Это решение позволяет распределить всё на три различных микрофронтенда, что способствует созданию гибкой и масштабируемой архитектуры приложения.

Для объединения трех упомянутых выше микрофронтендов используется Single App, что позволяет создать совместное взаимодействие между модулями и обеспечить единый пользовательский интерфейс. Single SPA обеспечивает согласованность данных и состояния приложения, а также упрощает развертывание и поддержку системы [12].

Серверная часть – специальный REST-сервер, промежуточное звено между запрашивающим ресурсом и базой данных. REST-сервер принимает HTTP-запросы от клиента или микросервиса, проверяет их, обращается к базе данных с помощью заранее определенных SQL-запросов. Ответ базы данных обрабатывается, упаковывается в JSON-объект и отправляется микросервису.

Модули REST-сервера включают регистрацию пользователей, авторизацию и аутентификацию, хранение метаданных пользователей и их ролей, модуль событий, компонентов, системы, активных правил, работы с базой данных, микросервисами и локальными репозиториями.

Визуальное отображение архитектуры показано на диаграмме развертывания (рисунок 9).

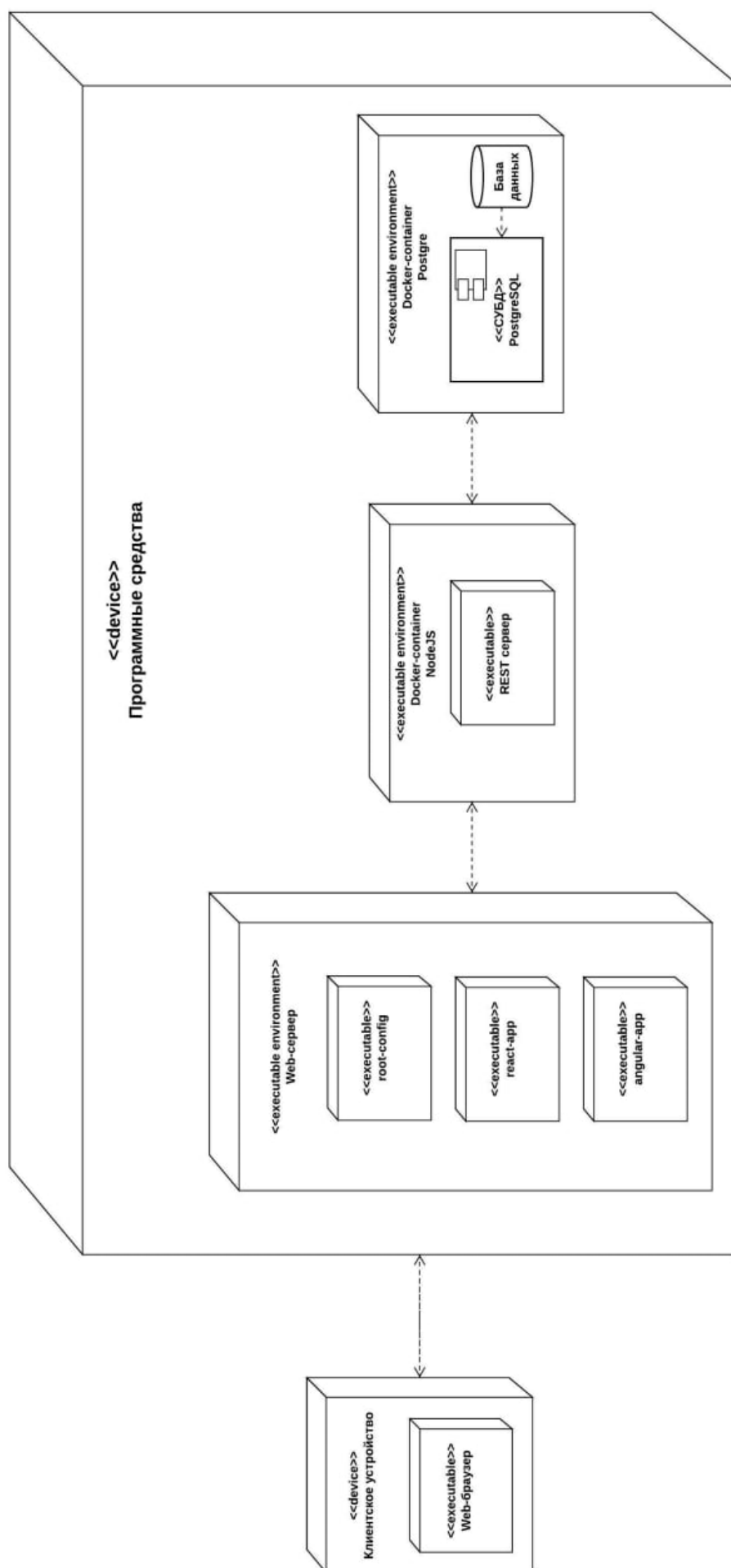


Рисунок 9 – Диаграмма развертывания

Для быстрого доступа к данным используется Redis – резидентная система управления базами данных, работающая в качестве буферного слоя. Управляющий модуль на REST-сервере, а локальные хранилища данных на микросервисах. При запросе метаданных микросервис проверяет локальный репозиторий Redis, и при отсутствии данных обращается к REST-серверу. REST-сервер обновляет локальные репозитории при отправке метаданных, обеспечивая актуализацию кэша микросервиса [13].

2.3 Требования к пользовательскому интерфейсу программных средств

Для обеспечения удобного доступа пользователей к метаданным сервиса потоковой обработки событий необходимо разработать пользовательский интерфейс, который сочетает в себе функциональность, понятность и логичную структуру. Основной задачей интерфейса является обеспечение высокой производительности работы и минимальной возможности ошибки со стороны пользователей, а также легкой усваиваемости для новых пользователей.

Для достижения этих целей интерфейс должен быть разделен на страницы, в зависимости от их функциональности и ролей пользователей, работающих с ними. Такой подход позволяет пользователям быстро и удобно находить необходимые функции и информацию, а также предоставлять им только необходимые возможности в соответствии с их ролями.

Каждая страница интерфейса должна быть функционально насыщенной и предлагать необходимые инструменты и функции для выполнения задач, связанных с обработкой событий и управлением метаданными. При этом структура интерфейса должна быть логичной и интуитивно понятной, чтобы пользователи могли быстро ориентироваться в системе и выполнять необходимые действия без лишних усилий.

Разделение интерфейса на страницы с функциональной направленностью позволит создать удобный и эффективный пользовательский интерфейс, который отвечает потребностям пользователей, обеспечивает высокую

производительность работы и облегчает процесс овладения интерфейсом для новых пользователей. На основе функциональных требований, необходимо разработать следующие страницы:

- страница авторизации должна содержать форму для ввода авторизующих пользователя данных;
- страница со всеми системами должна отражать все подключенные на данный момент к сервису системы;
- страница с подсистемами;
- страница с событиями выбранной системы;
- страница с активными правилами выбранной системы в табличном представлении;
- страница с активными правилами выбранной системы в теговом представлении;
- страница управления активными правилами в списочном представлении;
- страница управления активными правилами в графовом представлении;
- страница управления активными правилами в языковом представлении.

Разработка данных страниц позволит создать удобный и эффективный пользовательский интерфейс, который удовлетворит потребности пользователей и обеспечит эффективное взаимодействие с метаданными сервиса потоковой обработки событий.

2.4 Обоснование выбора технологий и средств разработки

При выборе технологий для разработки инструментальных средств управления метаданными учитывались функциональные требования, интерфейс и общая архитектура приложения.

Для создания пользовательского интерфейса было решено использовать HTML и CSS, так как они широко поддерживаются всеми популярными интернет-браузерами. Эти средства просты в использовании и позволяют легко создавать структурированные и стильные веб-страницы. Их гибкость и возможность интеграции с другими языками веб-программирования делают их идеальным выбором для разработки пользовательских интерфейсов.

JavaScript и TypeScript были выбраны для разработки функциональности сайта. JavaScript является популярным языком программирования, поддерживаемым на различных операционных системах и во всех известных браузерах. Он интерпретируемый, что ускоряет разработку и уменьшает размер программы. TypeScript, в свою очередь, расширяет возможности JavaScript, обеспечивая лучшую организацию кода для крупных и сложных программ и добавляя статическую типизацию для большей предсказуемости кода.

Таким образом, выбор HTML, CSS, JavaScript и TypeScript обеспечивает разработку мощного пользовательского интерфейса и функциональности приложения. Эти технологии соответствуют требованиям проекта, обеспечивают высокую производительность и упрощают разработку сложных функциональностей.

Кроме указанных выше, в разработке программного средства используются следующие технологии:

- React JS – используется для разработки мощных и масштабируемых пользовательских интерфейсов с использованием компонентного подхода;
- Single SPA – фреймворк для разработки микросервисных архитектур приложений, интегрирует различные фреймворки и библиотеки в одно приложение, обеспечивает ленивую загрузку компонентов, управление маршрутизацией и обмен данными, также улучшает масштабируемость и поддержку, создавая гибкие и отзывчивые пользовательские интерфейсы;

- CSS module – используется для работы с файлами стилей в рамках локальной области видимости, что обеспечивает изоляцию стилей между компонентами;
- БЭМ методология – используется для именования стилей, основанная на компонентном подходе, чтобы обеспечить понятность и структурированность стилей;
- GIT – система контроля версий разработки, используется для отслеживания изменений в коде, совместной работы и управления версиями проекта;
- Eslint/prettier – используются для анализа и форматирования кода, чтобы обеспечить единообразный и читабельный стиль кодирования;
- Redux – используется в качестве менеджера состояний для управления состоянием приложения и упрощения работы с данными;
- Yarn – менеджер пакетов, используется для управления зависимостями проекта и загрузки модулей и библиотек;
- SCSS – препроцессор CSS, предоставляет дополнительные возможности и удобства при работе со стилями, такие как переменные, вложенность и миксины;
- Webpack – сборщик модулей, используется для создания общего бандла и объединения множества TypeScript файлов в один, а также для управления зависимостями и оптимизации проекта;
- Ant Design – библиотека компонентов, предоставляет готовые стилизованные компоненты для быстрой и согласованной разработки пользовательского интерфейса;
- Docker – программное обеспечение для автоматизации развертывания и управления приложениями в контейнерах, что облегчает разработку, тестирование и развертывание приложений;
- Figma – редактор векторной графики, используется для разработки шаблонов пользовательского интерфейса и визуализации дизайна.

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

2.5 Требования к аппаратно-программному обеспечению

2.5.1 Требования к аппаратно-техническому обеспечению

К аппаратно-техническому обеспечению обычно относят 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 на данный раздел и нуждается в стандартных значениях.

2.5.2 Требования к системному программному обеспечению

Системное программное обеспечение описывает требования к операционным системам, 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

Исходя из описанных выше технологий для реализации был определен список поддерживаемых браузеров, на которых приложение будет отображаться и работать наиболее корректным образом:

- 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 управления метаданными для сервиса потоковой обработки событий

На основе перечня описанных в предыдущем разделе требований были сформированы основные компоненты и модули разрабатываемых программных средств.

Подробное содержание компонентов отражено на диаграмме пакетов на рисунке 10.

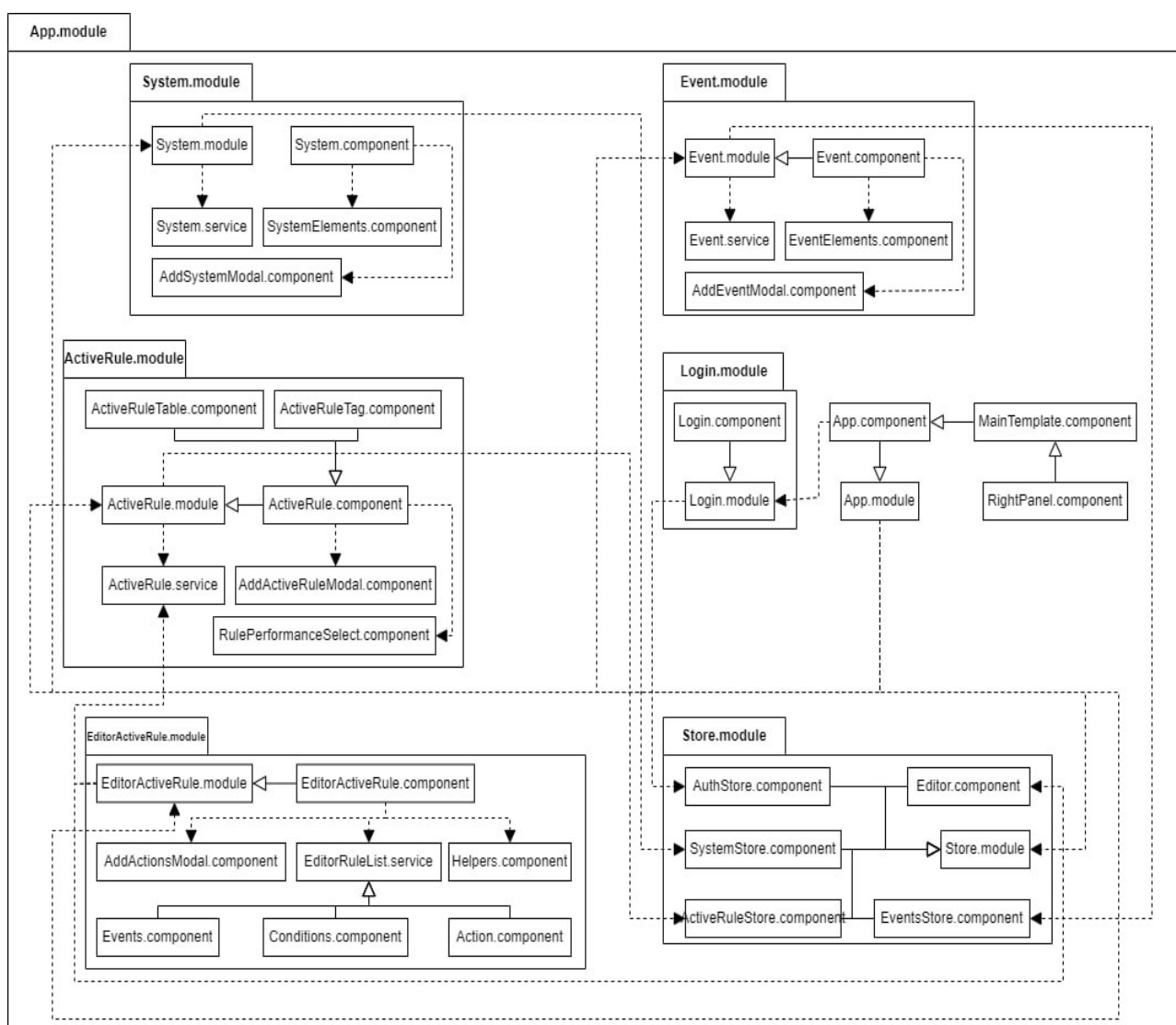


Рисунок 10 – Подробная архитектура элементов приложения в виде диаграммы пакетов

Структура этих компонентов отображена на рисунке 11 в виде диаграммы КОМПОНЕНТОВ.

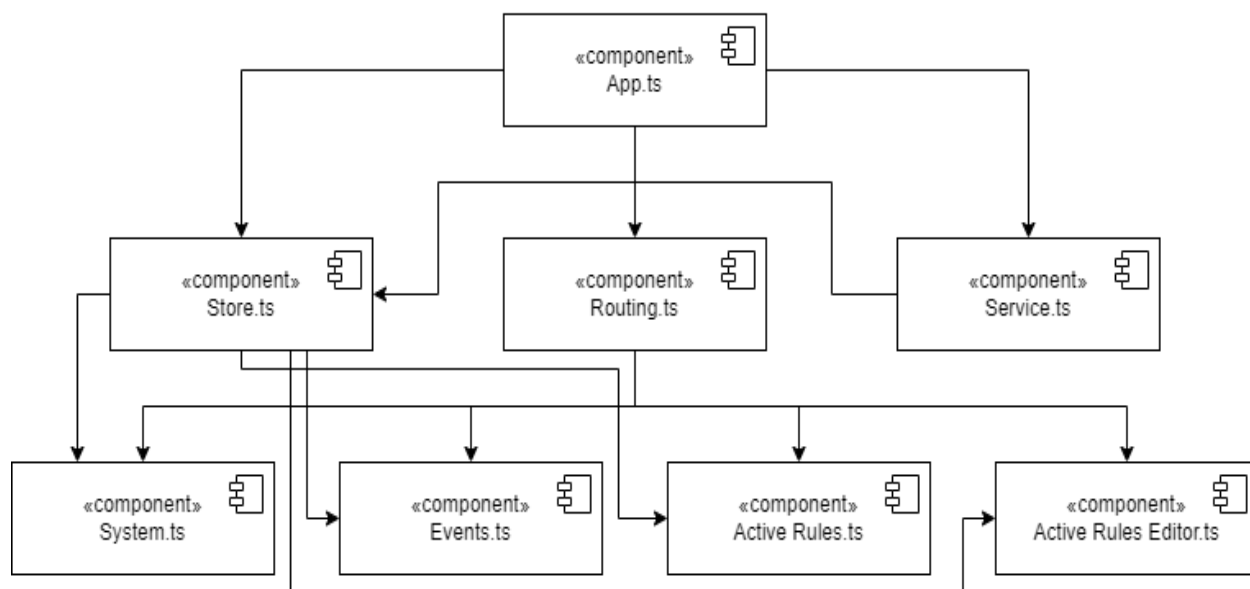


Рисунок 11 – Диаграмма основных компонентов веб-приложения

Основополагающие компоненты программных средств описаны в таблице 5.

Таблица 5 – Компоненты программных средств управления метаданными 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 состояний компонентов и глобальных переменных системы
5	Login.module	Модуль авторизации
6	System.module	Модуль для управления системами и подсистемами
7	Events.module	Модуль для управления событиями

Продолжение таблицы 5

№	Сущность	Описание
8	ActiveRules.module	Модуль для управления активными правилами
9	EditorActiveRule.module	Модуль для редактирования состояния активного правила
10	Tests.module	Модуль для хранения модульных тестов системы
11	MainTemplate.component	Компонент, отвечающий за общую обертку интерфейса системы
12	RightPanel.component	Компонент правой боковой панели для управления текущими модулями и навигации
13	AddSystemModale.component	Компонент модального окна создания системы
14	SystemElements.component	Компонент с карточками, содержащими информацию о системах
15	AddEventModal.component	Компонент модального окна добавления события
16	EventsElements.component	Компонент с элементами событий
17	ActiveRuleTable.component	Компонент таблицы с активными правилами
18	ActiveRuleTag.component	Компонент страницы с теговым представлением активных правил
19	RulePerformanceSelect.component	Компонент выпадающего списка для выбора текущего отображения активных правил
20	AddActiveRuleModal.component	Компонент модального окна для добавления активного правила
21	EditorRulePerformanceSelect.component	Компонент выпадающего списка для выбора текущего представления редактора активного правила
22	EditorRuleList.component	Компонент для списочного отображения редактора активного правила
23	Helpers.component	Компонент для работы с элементами активного правила
24	AuthStore.component	Компонент для хранения состояний для авторизации
25	ActiveRulesStore.component	Компонент для хранения состояний активных правил
26	EditorActiveRuleStore.component	Компонент для хранения состояний редактора активных правил
27	AddActionModal.component	Компонент модального окна для добавления действия
28	Actions.component	Компонент для отображения блока действий
29	Conditions.component	Компонент для отображения блока условий

Завершение таблицы 5

№	Сущность	Описание
30	Events.component	Компонент для отображения блока событий
31	Auth.service	Сервис, хранящий http-запросы для авторизации и регистрации пользователя
32	Systems.service	Сервис, хранящий http-запросы для управления системами и подсистемами
33	ActiveRules.service	Сервис, хранящий http-запросы для управления активными правилами
34	Events.service	Сервис, хранящий http-запросы для управления событиями
35	Component.service	Сервис, хранящий http-запросы для управления компонентами систем
36	Context.service	Сервис, хранящий http-запросы для управления контекстом событий

В рамках разрабатываемых программных средств можно выделить три основных типа сущностей:

- компоненты;
- сервисы;
- модули.

Компоненты представляют собой ключевые элементы приложения, отвечающие за реализацию конкретных функций и взаимодействие с пользовательским интерфейсом. Они содержат программный код, который определяет логику работы и взаимодействия с другими компонентами или сервисами. Компоненты включают в себя файлы HTML, CSS и JavaScript, которые обеспечивают визуальное представление и функциональность интерфейса.

Сервисы обеспечивают дополнительную функциональность и логику взаимодействия между компонентами, моделями или внешними системами. Они могут предоставлять методы и функции для выполнения определенных задач, обработки данных или управления ресурсами. Сервисы могут использоваться компонентами или другими сервисами в приложении для обмена информацией и выполнения необходимых операций.

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

3.2 Разработка пользовательского интерфейса

На основе требований к разрабатываемым программным средствам был разработан графический пользовательский интерфейс web-приложения. Данный интерфейс разделен на несколько страниц:

- страница просмотра систем и подсистем, а также их управления;
- страница просмотра списка событий.

Страница просмотра систем и подсистем, а также их управления представлена на рисунке 12.

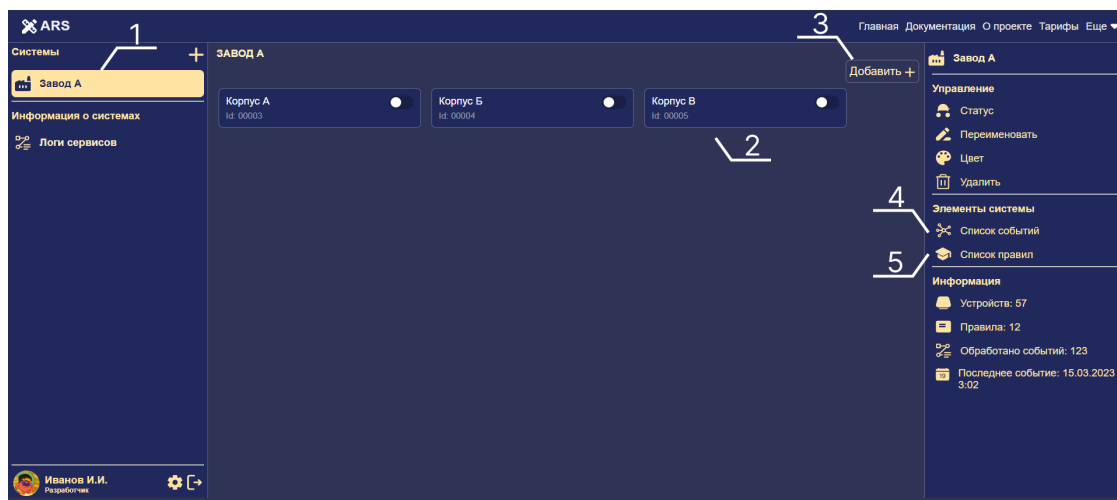


Рисунок 12 – Страница просмотра систем и подсистем, а также их управления

Список наиболее значимых элементов, расположенных на странице, отображенной на рисунке 12:

- 1 – список родительских систем;
- 2 – карточки подсистем выбранной родительской системы;
- 3 – кнопка добавления подсистем;
- 4 – ссылка на переход на страницу просмотра списка событий;
- 5 – ссылка на переход на страницу просмотра списка активных правил.

Далее рассмотрим упомянутую выше страницу просмотра списка событий, она отображена на рисунке 13.

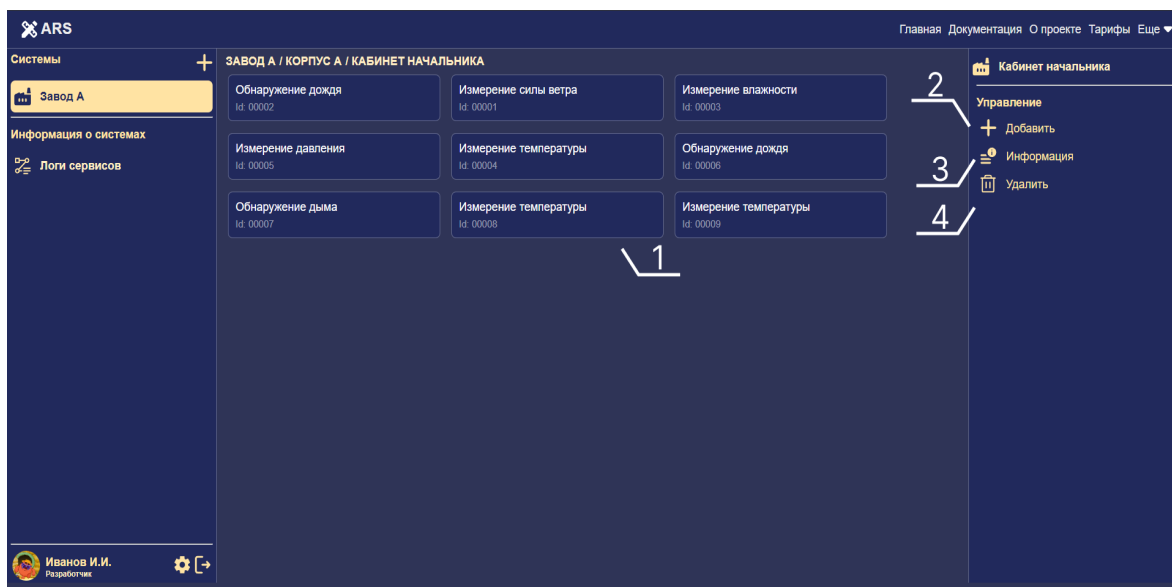


Рисунок 13 – Страница просмотра списка событий

Список наиболее значимых элементов, расположенных на странице, отображенной на рисунке 13:

- 1 – карточки событий выбранной подсистемы;
- 2 – кнопка добавления события;
- 3 – кнопка просмотра информации о событии;
- 4 – кнопка удаления события.

После нажатия на кнопку добавления события открывается модальное окно, показанное на рисунке 14.

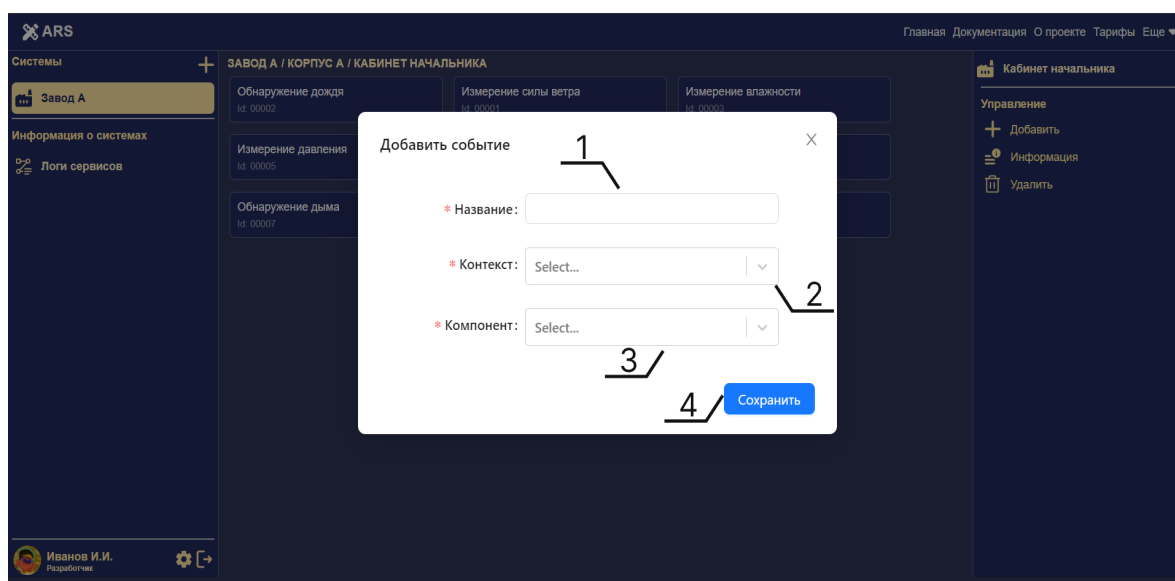


Рисунок 14 – Модальное окно добавления события

Список наиболее значимых элементов, расположенных на странице, отображенной на рисунке 14:

- 1 – поле ввода названия создаваемого события;
- 2 – выпадающий список выбора контекста;
- 3 – выпадающий список выбора компонента;
- 4 – кнопка сохранения введенных данных.

Если вернуться на страницу просмотра систем и подсистем, то с нее можно перейти на страницу просмотра активных правил, изображенную на рисунке 15.

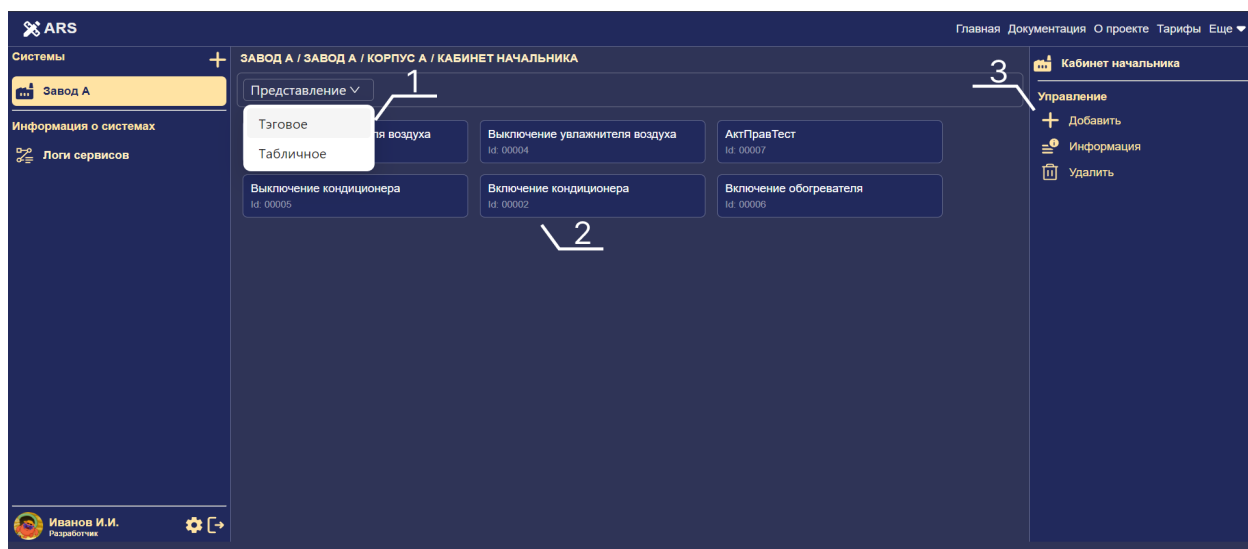


Рисунок 15 – Страница просмотра активных правил выбранной подсистемы

Список наиболее значимых элементов, расположенных на странице, отображенной на рисунке 15:

- 1 – выпадающий список вариантов отображения активных правил;
- 2 – карточки активных правил;
- 3 – панель управления с кнопками добавления, просмотра информации и удаления активных правил.

По умолчанию страница открывается в тэговом представлении активных правил, однако в выпадающем списке можно выбрать и табличное представление. Страница с табличным представлением активных правил представлена на рисунке 16.

Название	Id	Статус	Теги
Выключение увлажнителя воздуха	00004	☐	Увлажнение Кабинет начальника
Включение увлажнителя воздуха	00003	☐	Увлажнение Кабинет начальника
Выключение кондиционера	00005	☐	Увлажнение Кабинет начальника
Включение кондиционера	00002	☐	Увлажнение Кабинет начальника

Рисунок 16 – Страница табличного представления активных правил

Список наиболее значимых элементов, расположенных на странице, отображенной на рисунке 16:

- 1 – столбец название активного правила;
- 2 – столбец id активного правила;
- 3 – столбец статуса активного правила;
- 4 – столбец тэгов активного правила.

В рассмотренной выше панели управления пользователь может нажать на кнопку добавления активного правила, после чего откроется модальное окно создания правила. Модальное окно отображено на рисунке 17.

Добавить активное правило

* Название:

* Условие:

* Действие:

* Событие:

* Тип связи:

Сохранить

Рисунок 17 – Модальное окно создания активного правила

Список наиболее значимых элементов, расположенных на странице, отображенной на рисунке 17:

- 1 – поле ввода названия активного правила;
- 2 – поле ввода условия активного правила;
- 3 – поле ввода действия активного правила;
- 4 – выпадающий список для выбора события, связанного с активным правилом;
- 5 – выпадающий список для выбора типа связи события с активным правилом;
- 6 – кнопка сохранения введенных данных.

Дважды кликнув на активное правило, пользователь переходит на страницу редактирования активного правила. По умолчанию открывается списочное представление. Страница редактирования АП продемонстрирована на рисунке 18.



Рисунок 18 – Страница редактирования активного правила в списочном представлении

Список наиболее значимых элементов, расположенных на странице, отображенной на рисунке 18:

1 – выпадающий список для выбора представления элементов активного правила;

2 – блок отображения события и сопутствующей информации;

3 – блок отображения условия и сопутствующей информации;

4 – блок отображения действий и сопутствующей информации;

5 – кнопка удаления действия;

6 – кнопка добавления нового действия.

После нажатия на кнопку добавления нового действия откроется модальное окно создания действия, отображенное на рисунке 19.

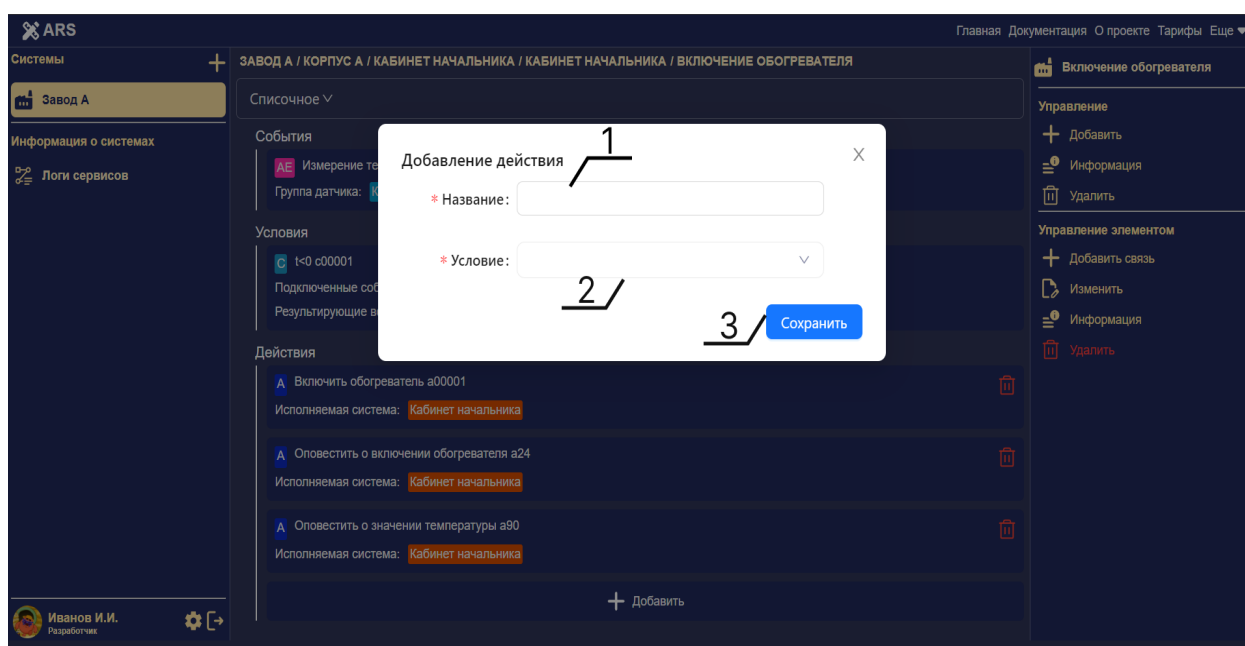


Рисунок 19 – Модальное окно создания активного правила

Список наиболее значимых элементов, расположенных на странице, отображенной на рисунке 19:

1 – поле ввода названия создаваемого действия;

2 – выпадающий список для выбора условия, с которым будет связано создаваемое действия;

3 – кнопка сохранения введенных данных.

3.3 Разработка программных средств регистрации и авторизации пользователей

Регистрация и авторизация пользователей позволяют идентифицировать пользователей и проверить их подлинность. При регистрации пользователь предоставляет уникальные учетные данные, которые в дальнейшем используются для аутентификации при каждом входе в систему. Это обеспечивает безопасность и защиту данных, позволяя системе различать пользователей и предоставлять им доступ к соответствующим ресурсам и функциональности.

Регистрация новых пользователей и авторизация уже зарегистрированных происходит на странице, отображенной на рисунке 20.

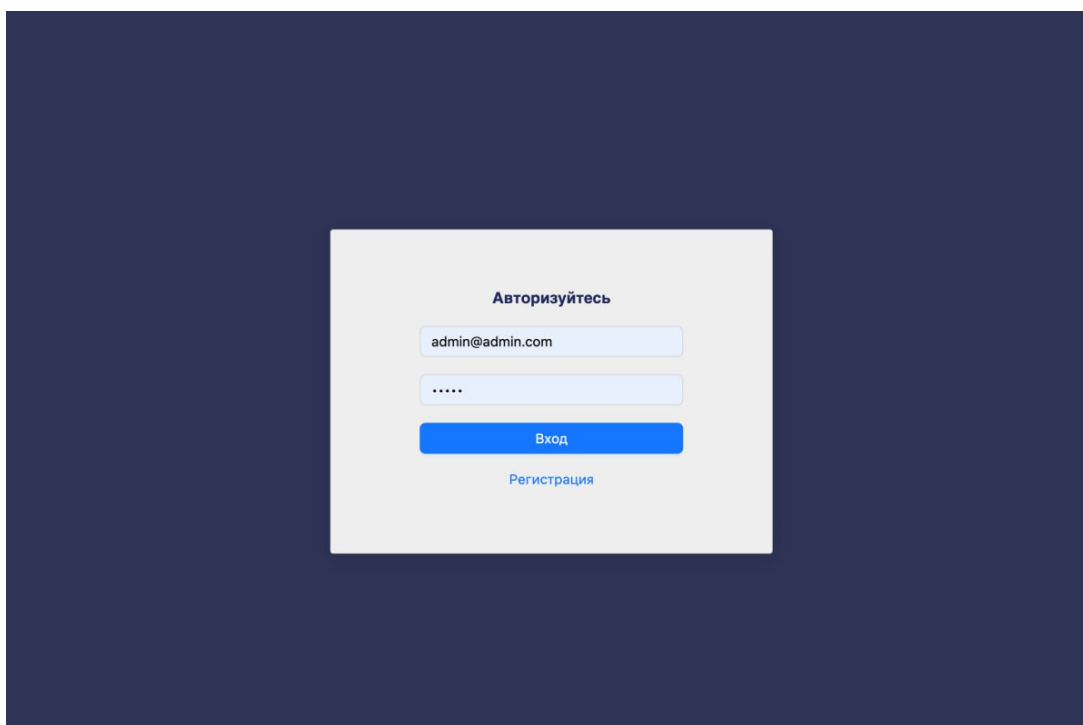


Рисунок 20 – Страница авторизации и регистрации пользователей

Для регистрации на сервер передаются введенные пользователем email и пароль. После получения сервером данных, они проверяются на наличие ранее созданных учетных записей с такими же данными.

При каждом новом сеансе работы с программными средствами управления метаданными для сервиса потоковой обработки событий пользователям

необходимо авторизоваться, введя логин и пароль. При авторизации вызывается метод, который отправляет данные на сервер, где проверяется наличие в БД учетной записи с введенными данными. Если данные есть, то метод возвращает Access token и refresh token. Далее в локальном хранилище браузера проверяется наличие этих двух токенов: если токены есть, то они перезаписываются; если нет, то добавляются.

3.4 Разработка программных средств просмотра систем, подсистем и событий выбранной системы

После успешной авторизации пользователя в веб-приложении ему становится видна страница со списком всех доступных для просмотра и управления систем.

Диаграмма деятельности создания систем и подсистем показана на рисунке 21.

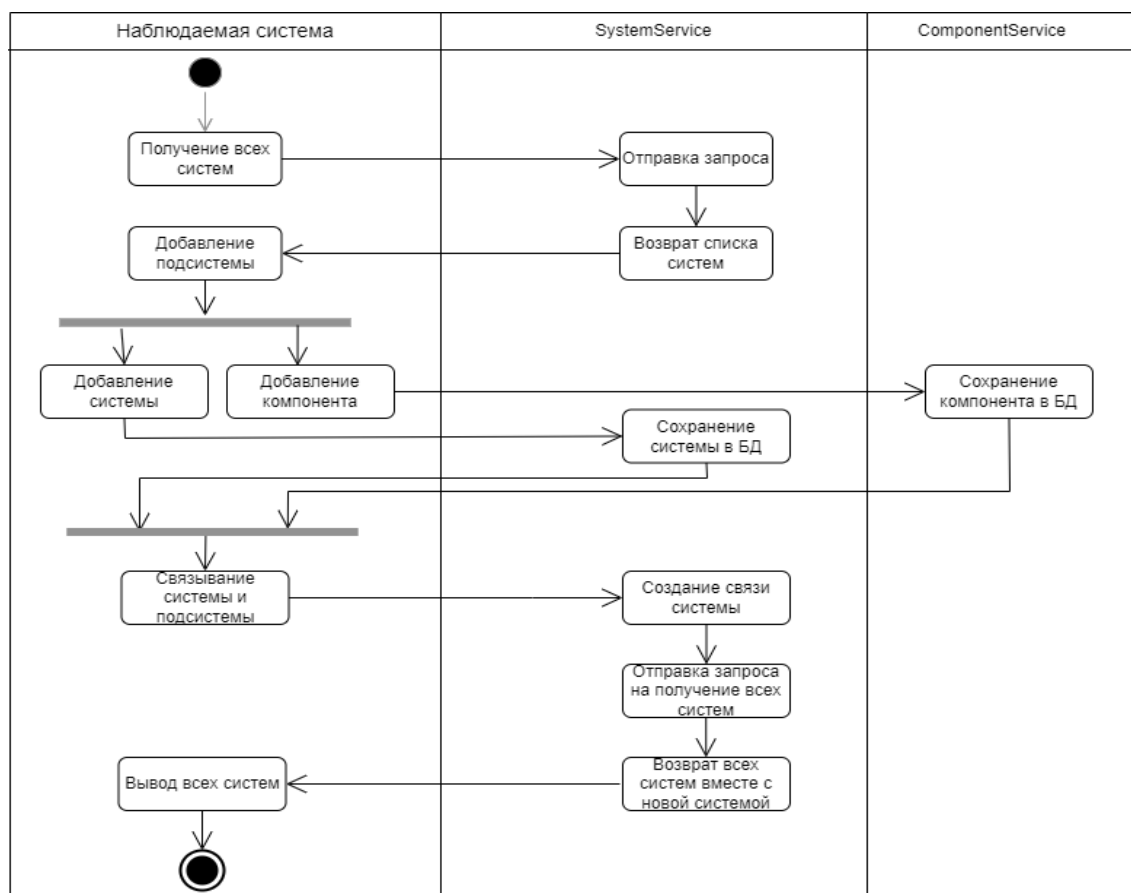


Рисунок 21 – Диаграмма деятельности создания систем и подсистем

Листинг модели систем и событий представлен в приложении Б.

В левой боковой панели отображаются родительские системы, то есть те, которые не входят в какие-либо другие 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.

Над рабочей областью отображается путь до выбранной на данный момент подсистемы. С помощью клика на определённый фрагмент «хлебных крошек» пользователь может переместиться на страницу открытых ранее систем.

В правой боковой панели находятся кнопки:

- «Добавить» для создания системы;
- «Удалить» для удаления выбранной системы.

Диаграмма классов модели систем показана на рисунке 22.

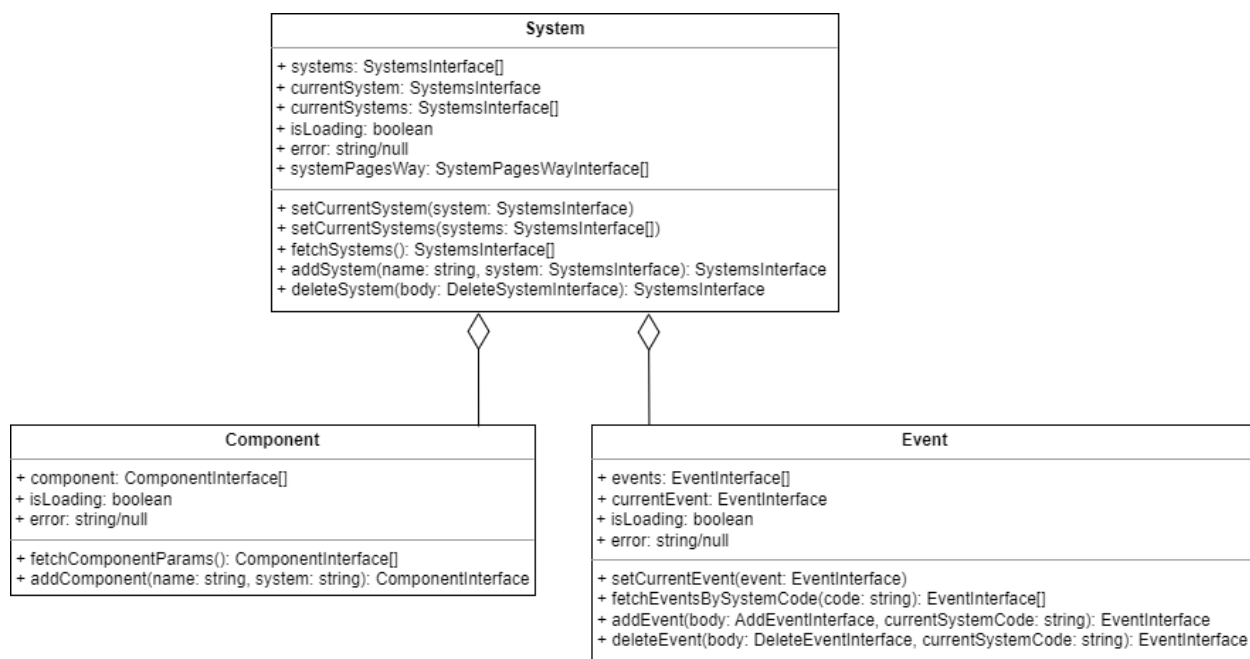


Рисунок 22 – Диаграмма классов модели систем

Получение системы происходит с помощью метода GET /system; добавление системы – POST /system; удаление системы – DELETE /system. Пример функции создания подсистемы представлен на листинге 1.

Листинг 1 – Функция создания подсистемы

```
export const addSystem = (name: string, system: SystemsInterface)
=> {
  return async (dispatch: Dispatch<SystemsAction>) => {
    try {
      const newSystem = await
SystemService.addSystem({name});
      if (newSystem) {
        await
SystemService.addSystemAttachment({codeParent: system.code,
codeAttach: newSystem.data.code});
        const updatedSystem = await
SystemService.fetchSystemByCode(system.code);
        if (updatedSystem) {

dispatch(setCurrentSystems(updatedSystem.data.children));
          dispatch(setCurrentSystem(updatedSystem.data));
          dispatch(fetchSystems());
        }
      }
    } catch (e) {
      console.error('Произошла ошибка добавления системы: ',
e);
    }
  };
};
```

По выбранной подсистеме можно посмотреть события. Нужно кликнуть на карточку подсистемы и в правой боковой панели перейти по ссылке «Список событий». На странице, на которую будет осуществлен переход, отобразится ряд событий, выбранной ранее подсистемы. Кроме того, пользователь может добавить новое событие в систему. Для этого нужно в правой управляющей панели выбрать кнопку «Добавить событие», после чего откроется модальное окно с необходимыми для создания полями. В двух из них будут выпадающие списки с выбором компонент и контекста. Данные для списков подгружаются после открытия страницы событий. Кроме создания события подсистемы также можно его удалить, для чего в правой панели найти соответствующую кнопку. Кликнув по ней событие пропадет из общего списка.

На рисунке 23 представлена диаграмма классов модели событий.

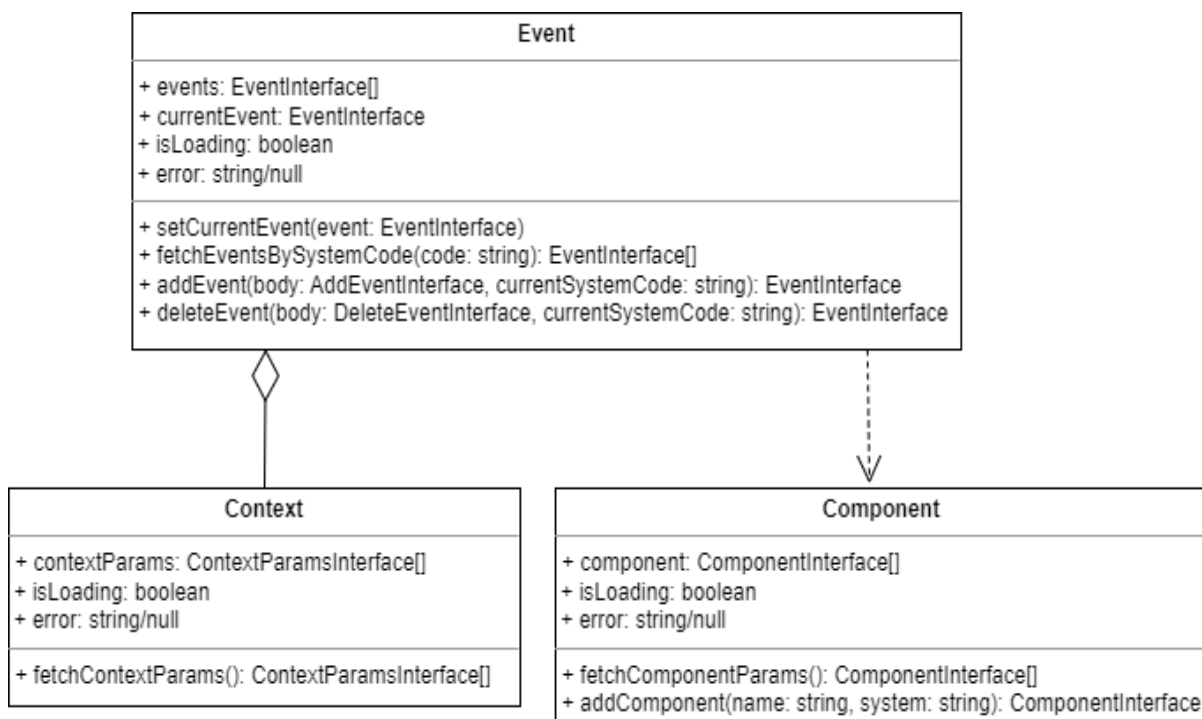


Рисунок 23 – Диаграмма классов модели событий

Получение всех событий происходит с помощью метода GET /event/system/*код выбранной подсистемы*; создание событий – POST /event; удаление событий – DELETE /event.

При добавлении события необходимо ввести его название, выбрать компонент, то есть систему, к которой нужно привязать событие, и указать контекст (единицы измерения).

3.5 Разработка программных средств управления активных правил

Выбрав подсистему, пользователь, кликнув на ссылку «Список правил» в правом боковом меню, может перейти на страницу отображения всех активных правил, относящихся к выбранной подсистеме. Активное правило можно создать, изменить, а также просмотреть его в карточном или табличном видах.

Диаграмма классов модели активных правил показана на рисунке 24.

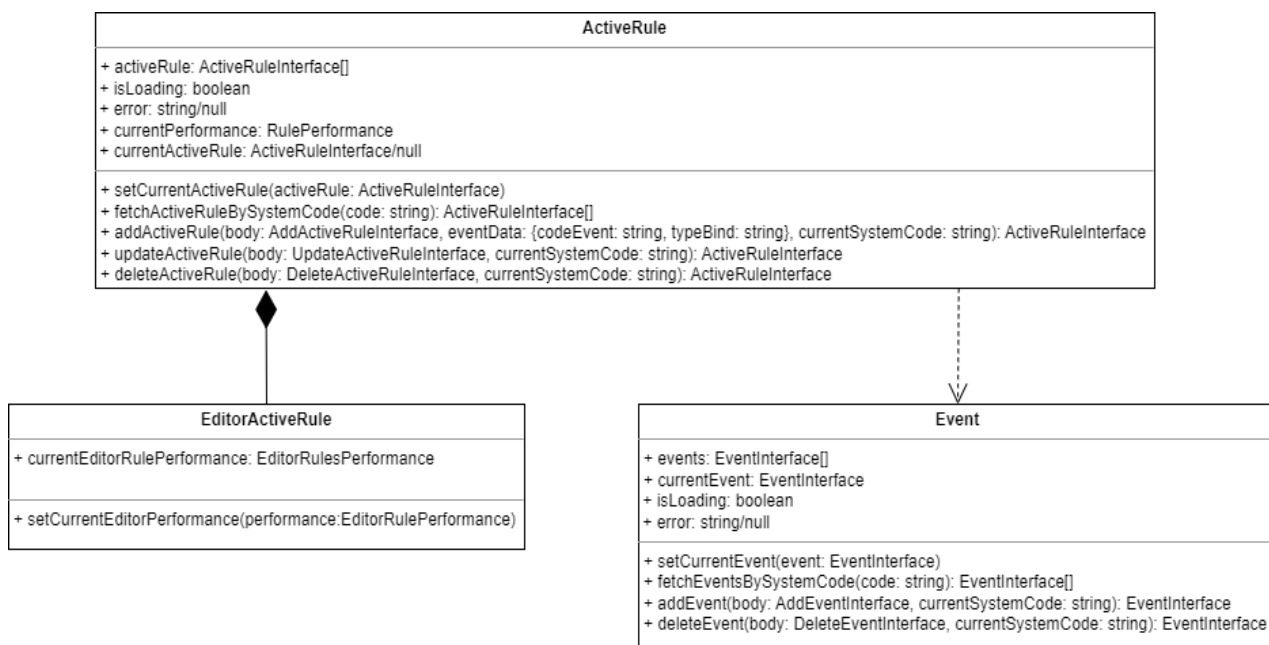


Рисунок 24 – Диаграмма классов модели активных правил

Получение правил происходит с помощью метода GET /active-rule/system/*код выбранной подсистемы*; создание правила – POST /active-rule; удаление правила – DELETE/active-rule. Создать активное правило пользователь может в модальном окне, проделав несколько шагов:

- задать название активного правила, действия и условия, введя их в соответствующие текстовые блоки;
- выбрать событие из выпадающего списка, в который входят все входящее в систему и связанное с создаваемым правилом события;
- выбрать тип связи («От события к действию» или «От действия к событию»);
- нажать на кнопку сохранения и отправить заполненную данными форму в функцию обратного вызова на событие сохранения, где данные формы будут приведены в вид, необходимый методом сервера, и отправлены на сохранение в базу данных.

Диаграмма состояний создания активного правила показана на рисунке 25.

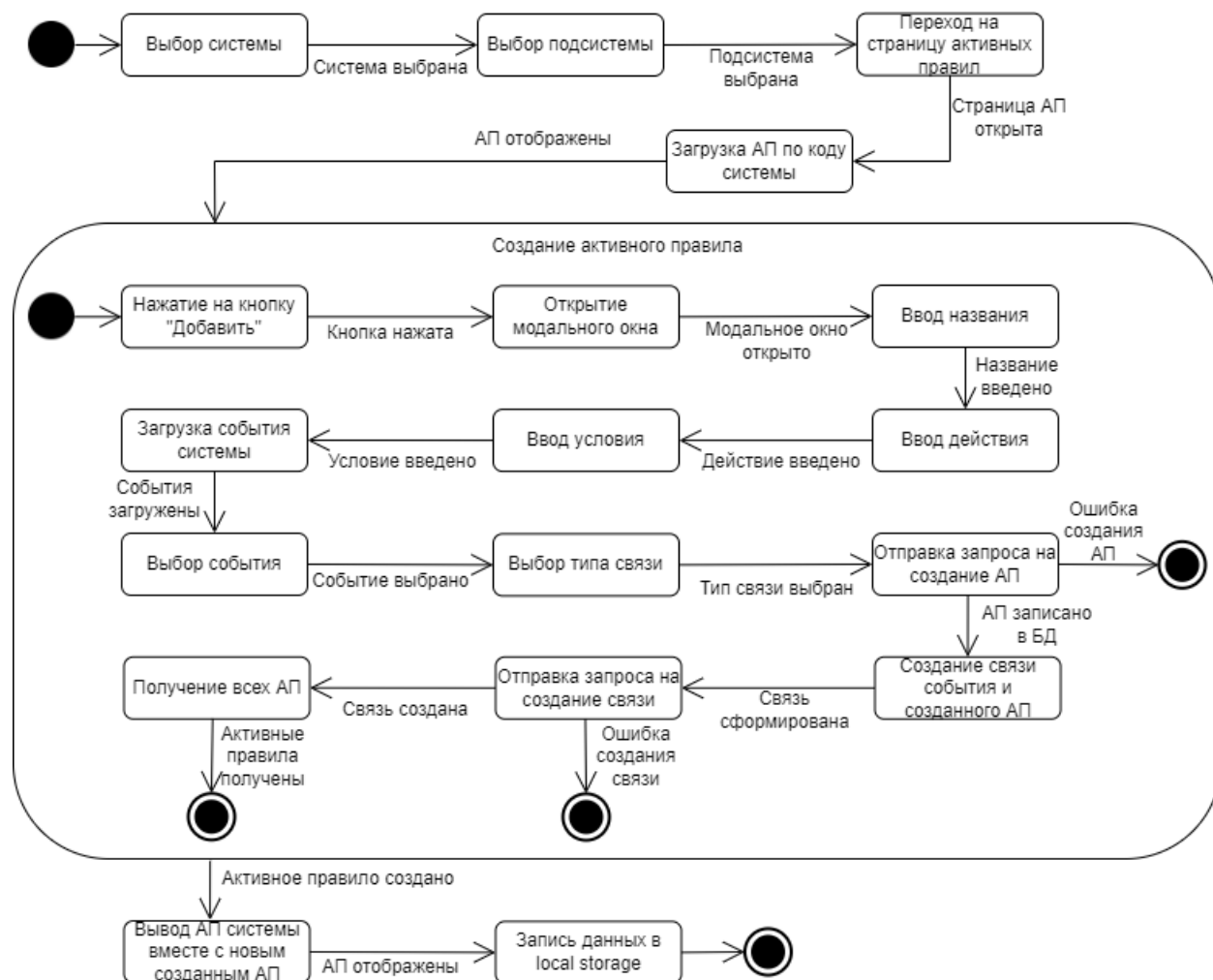


Рисунок 25 – Диаграмма состояний создания активного правила

Листинг модели активных правил представлен в приложении В.

При создании активного правила сначала вызывается метод PATCH /active-rule, куда передается название активного правила, условие и действие. Затем происходит вызов метода POST /active-rule/event, посредством которого происходит связь созданного активного правила с событием. Функции, использующиеся для создания активного правила, показаны на листинге 2.

Листинг 2 – Функция создания активного правила

```

const onFinish = async (data: {
  description: string, condition: string,
  action: string, codeEvent: string, typeBind: string
}) => {
  const requestData = {
    description: data.description,
  
```

```

        condition: data.condition ? JSON.stringify({
            data: [
                {
                    category: 'condition',
                    description: data.condition,
                    code: 'c00001'
                }
            ]
        }) : '',
        action: JSON.stringify({
            data: [
                {
                    category: 'action',
                    description: data.action,
                    code: 'a00001',
                }
            ],
            edges: [
                {
                    from: 'c00001',
                    to: 'a00001'
                }
            ]
        })
    });
    const eventData = {
        codeEvent: data.codeEvent,
        typeBind: data.typeBind
    };

    await dispatch(() => addActiveRule(requestData, eventData,
currentSystemCode));
    setIsVisible(false);
};
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({
                    codeRule: activeRule.data.code,
                    codeEvent: eventData.codeEvent, typeBind:
eventData.typeBind
                });
            }
        } catch (e) {

```

```

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

```

Активные правила отображаются в двух видах: карточки-теги и таблица. Кликнув дважды по активному правилу, пользователь переходит на страницу с редактирования. Редактировать активное правило можно в нескольких представлениях:

- списочное;
- графовое;
- языковое.

Подробнее списковое представление рассмотрено в следующем подразделе.

3.6 Разработка программных средств списочного конструирования

Редактирование активных правил может осуществляться в том числе в списочном представлении. У любого активного правила есть поля `action`, `conditions` и `events`, с помощью которых происходит получение данных для трех блоков списочного представления: событие, условие и действие. В этих блоках отражается текущая информация о составе активного правила.

В списочном представлении из всех активных правил с помощью метода `GET /active-rule/system/*код выбранной подсистемы*` получаем то активное правило, в редактор которого был осуществлен переход.

Кроме того, в списочном представлении активному правилу можно добавить новое действие, выполнив следующие шаги:

- кликнуть на кнопку «Добавить»;
- ввести название действия;
- выбрать условие.

Пример функции добавления действия в активное правило представлен в листинге 3.

Листинг 3 – Функция добавления действия

```
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();
};
```

Метод PATCH `/active-rule/` обновляет измененное значение активного правила. При добавлении действия новое добавляется в соответствующий блок.

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 являются неотъемлемой частью процесса разработки и обеспечивают высокий уровень качества конечного продукта. Они несколько важных целей и характеристик.

Мероприятия по контролю качества позволяют убедиться, что программное средство полностью соответствует функциональным и нефункциональным требованиям, установленным для него. Это включает проверку правильности реализации функций, соответствие интерфейсов, требованиям безопасности и производительности.

Одной из главных задач мероприятий по контролю качества является выявление дефектов, ошибок и неполадок в программных средствах. Это может быть достигнуто путем систематического тестирования, анализа кода и использования специализированных инструментов для обнаружения проблем.

Кроме того, мероприятия по контролю качества направлены на обеспечение высокой надежности и стабильности программного средства. Это включает проверку устойчивости к непредвиденным ситуациям, обработку ошибок, контроль доступа и сохранность данных.

Два основных подхода к мероприятиям по контролю качества ПС включают анализ кода и подсчет метрик, а также тестирование программных средств.

Подход подсчета метрик основан на анализе программного кода для выявления потенциальных проблем и нарушений стандартов разработки. Подсчет метрик позволяет оценить качество кода и выявить его уязвимости [14]. Тестирование является ключевым мероприятием для проверки работоспособности, соответствия требованиям и выявления дефектов в

программных средствах. Это включает разработку и выполнение тестовых сценариев, проверку функциональности, производительности, безопасности и других аспектов работы ПС.

4.1.2 Обоснование выбора модели тестирования

Для проверки работоспособности программного обеспечения применяют множество видов тестирования. Рассмотрим подробнее некоторые из них.

Функциональное тестирование – вид тестирования, который проверяет, насколько функциональности программного средства соответствуют его требования. Он проверяет, выполняются ли функции и основные операции программы в соответствии с ожидаемым поведением.

Конфигурационное тестирование – тестирование, которое выполняется для проверки работоспособности программного средства в различных конфигурациях и окружениях. Оно включает проверку совместимости с разными операционными системами, аппаратными платформами, базами данных и другими факторами, которые могут влиять на функционирование программы.

Интеграционное тестирование – вид тестирования, направленный на проверку взаимодействия различных компонентов программного средства. Он проверяет, как разные модули или подсистемы взаимодействуют друг с другом и работают вместе в рамках единой системы.

Нагрузочное тестирование – вид тестирования для оценки производительности и стабильности программного средства при работе с большой нагрузкой. Он проверяет, как система справляется с высокими объемами данных или пользовательским трафиком и позволяет идентифицировать ее пределы и возможные проблемы.

Регрессионное тестирование – тестирование для проверки того, что изменения в программном средстве не вызывают нежелательных побочных эффектов или не повреждают существующую функциональность. Оно выполняется после внесения изменений или исправления дефектов для убеждения в стабильности и работоспособности программы.

Автоматизированное модульное тестирование – вид тестирования, при котором модули программного средства тестируются независимо от других компонентов. Он использует автоматизированные средства и инструменты для проведения тестов и проверки отдельных функциональных блоков кода.

Выбор функционального, конфигурационного и модульного тестирования для программных средств управления метаданными в сервисе потоковой обработки событий обоснован следующим образом:

Функциональное тестирование необходимо для проверки соответствия функциональности программного средства требованиям, определенным для управления метаданными. Это позволит убедиться, что все функции и операции, связанные с метаданными, работают корректно и соответствуют ожиданиям.

Конфигурационное тестирование важно, поскольку разрабатываемые программные средства могут работать в разных конфигурациях и окружениях. Тестирование в различных конфигурациях поможет убедиться, что программное средство успешно функционирует и взаимодействует с различными операционными системами, аппаратными платформами и другими факторами.

Автоматизированное модульное тестирование применимо для тестирования отдельных функциональных блоков кода, связанных с управлением метаданными. Это поможет выявить и исправить потенциальные проблемы и ошибки в отдельных модулях, обеспечивая стабильность и надежность работы программного средства.

4.1.3 Обоснование выбора метрик программного кода

В качестве метрик программного кода были выбраны анализ абстрактности и устойчивости программного обеспечения, метрики Холстеда, а также оценка качества пользовательского интерфейса.

Анализ абстрактности и устойчивости программного обеспечения позволяет оценить уровень абстракции и универсальности кода. Метрика абстрактности показывает отношение количества абстрактных классов к общему числу классов в компоненте. Высокий уровень абстрактности может указывать

на гибкость и переиспользуемость кода, что способствует легкому внесению изменений и расширению функциональности программы. Метрика устойчивости отражает количественную меру трудозатрат, которые необходимо произвести при изменении зависимых компонентов от анализируемого компонента.

Метрики Холстеда предоставляют количественные оценки сложности программного кода, основанные на количестве операторов и операндов в коде. Метрики Холстеда также могут указывать на наличие дублирования кода, что может привести к увеличению сложности поддержки и изменений программы. Идентификация и устранение дублирования кода может повысить его читаемость, снизить вероятность ошибок и упростить процесс разработки.

Оценка качества пользовательского интерфейса позволяет оценить, насколько быстро та или иная страница отвечает на действия пользователя, что важно для работы с широким функционалом разрабатываемых программных средств.

4.2 Тестирование программных средств

4.2.1 Функциональное тестирование

Для проверки функциональности программного продукта и соответствия его работы заявленным требованиям проводится функциональное тестирование.

Подробное описание функциональных тестов представлено в таблице 6.

Таблица 6 – План функционального тестирования

Название теста	Входные данные	Ожидаемый результат	Тест пройден/не пройден
Создание новой подсистемы	Название системы – «Корпус С»	На странице отображена карточка с названием добавленной подсистемы	Тест пройден (рисунок Д.1)
Создание нового события	Название события – «Измерение уровня освещения» Контекст – «Освещение» Компонент – «»	На странице отображена карточка с названием добавленного события	Тест пройден (рисунок Д.2)

Продолжение таблицы 6

Название теста	Входные данные	Ожидаемый результат	Тест пройден/не пройден
Создание нового активного правила	Название – «Включение света» Условие – «Уровень света < 10» Действие – «Включить свет» Событие – «Измерение уровня освещения» Тип связи – «Событие к правилу»	На странице отображена карточка с названием добавленного активного правила	Тест пройден (рисунок Д.3)
Добавить действие в списочном представлении активного правила	Название – «Подать звуковой сигнал» Условие – «Включение обогревателя»	На странице в блоке действий отображено добавленное действие	Тест пройден (рисунок Д.4)
Переключение с тэгового на табличное представление активного правила		На странице в отображены все активные правила выбранной подсистемы в табличном представлении	Тест пройден (рисунок Д.5)
Удаление подсистемы «Корпус С»		На странице в не отображается удаленная подсистема	Тест пройден (рисунки Д.6 и Д.7)
Переход по хлебным крошкам		Осуществлен переход со страницы событий до корневой подсистемы	Тест пройден

Тестирование разработанного приложения проводилось на примере системы, описанной в приложении Г. Все рисунки со скриншотами пройденных тестов приведены в приложении Д.

4.2.2 Конфигурационное тестирование

В целях оценки качества программного обеспечения и проверки его работоспособности в различных исполняемых окружениях, было выполнено конфигурационное тестирование с использованием нескольких наиболее популярных веб-браузеров. В процессе тестирования осуществлялся запуск web-

приложения в различных браузерах с целью проверки его работоспособности и согласованности результатов.

Для проведения конфигурационного тестирования веб-приложения, был выполнен запуск приложения в различных веб-браузерах. Детальный план тестирования, описывающий последовательность и параметры запуска в разных окружениях, приведен в таблице 7.

Таблица 7 – План конфигурационного тестирования

Тестируемый браузер	Тестируемая ситуация	Ожидаемый результат	Полученный результат
Google Chrome (версия 114.0.5735.110)	Отображение систем и подсистем	Страница с системами и подсистемами отображена	Фактический результат совпал с ожидаемым (рисунок E.1)
	Отображение активных правил системы	Страница с активными правилами системы отображена	Фактический результат совпал с ожидаемым (рисунок E.2)
	Отображение событий системы	Страница с активными правилами системы отображена	Фактический результат совпал с ожидаемым (рисунок E.3)
Microsoft Edge (версия 114.0.1823.37)	Отображение систем и подсистем	Страница с системами и подсистемами отображена	Фактический результат совпал с ожидаемым (рисунок E.4)
	Отображение активных правил системы	Страница с активными правилами системы отображена	Фактический результат совпал с ожидаемым (рисунок E.5)
	Отображение событий системы	Страница с активными правилами системы отображена	Фактический результат совпал с ожидаемым (рисунок E.6)
Яндекс.Браузер (версия 23.5.1.721)	Отображение систем и подсистем	Страница с системами и подсистемами отображена	Фактический результат совпал с ожидаемым (рисунок E.7)
	Отображение активных правил системы	Страница с активными правилами системы отображена	Фактический результат совпал с ожидаемым (рисунок E.8)
	Отображение событий системы	Страница с активными правилами системы отображена	Фактический результат совпал с ожидаемым (рисунок E.9)

Продолжение таблицы 7

Тестируемый браузер	Тестируемая ситуация	Ожидаемый результат	Полученный результат
Firefox Browser (версия 72.0.2)	Отображение систем и подсистем	Страница с системами и подсистемами отображена	Фактический результат совпал с ожидаемым (рисунок E.10)
	Отображение активных правил системы	Страница с активными правилами системы отображена	Фактический результат совпал с ожидаемым (рисунок E.11)
	Отображение событий системы	Страница с активными правилами системы отображена	Фактический результат совпал с ожидаемым (рисунок E.12)
Opera (версия 85.0.4341.18)	Отображение систем и подсистем	Страница с системами и подсистемами отображена	Фактический результат совпал с ожидаемым (рисунок E.13)
	Отображение активных правил системы	Страница с активными правилами системы отображена	Фактический результат совпал с ожидаемым (рисунок E.14)
	Отображение событий системы	Страница с активными правилами системы отображена	Фактический результат совпал с ожидаемым (рисунок E.15)

Программные средства запустились и выдали ожидаемый результат во всех тестируемых web-браузерах. Данное мероприятие позволило убедиться в том, что приложение функционирует корректно и обеспечивает одинаковый уровень производительности и визуального представления в разных браузерах, обеспечивая таким образом оптимальный пользовательский опыт в любом выбранном окружении. Все рисунки со скриншотами пройденных тестов приведены в приложении E.

4.2.3 Модульное тестирование

Автоматизированное тестирование производилось при помощи фреймворка WebdriverIO. Это инструмент с открытым исходным кодом для автоматического тестирования веб-приложений [15].

Модульное (Unit) тестирование проведено на примере компонентов авторизации, получения систем и подсистем, получения событий, получения активных правил, получения данных для редактора активных правил.

Все компоненты прошли модульное тестирование. Результаты тестирования авторизации представлены на рисунке 26.

```
-----
[chrome 114.0.5735.106 mac os x #0-4] Running: chrome (v114.0.5735.106) on mac os x
[chrome 114.0.5735.106 mac os x #0-4] Session ID: a021a4783903ef9371b541dc3c1ba570
[chrome 114.0.5735.106 mac os x #0-4]
[chrome 114.0.5735.106 mac os x #0-4] » /tests/e2e/login.e2e.js
[chrome 114.0.5735.106 mac os x #0-4] Login page test
[chrome 114.0.5735.106 mac os x #0-4]    ✓ auth testing
[chrome 114.0.5735.106 mac os x #0-4]
[chrome 114.0.5735.106 mac os x #0-4] 1 passing (9.1s)
-----
```

Рисунок 26 – Результат модульного тестирования компонента авторизации

Скриншоты остальных результатов тестирования представлены в приложении Ж. Исходный код модульных тестов приведен в приложении И.

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 8 основных модулей и 1 сторонний фреймворк: App, Api, Store, Login, Systems, Events, ActiveRules, EditorActiveRules.

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);

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

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

Была рассчитана величина устойчивости для каждого компонента системы:

$$I_1 = \frac{7}{7+1} = 0,875;$$

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

$$I_3 = \frac{1}{1+1} = 0,5;$$

$$I_4 = \frac{1}{4+1} = 0,2;$$

$$I_5 = \frac{1}{11+1} = 0,083;$$

$$I_6 = \frac{1}{6+1} = 0,143;$$

$$I_7 = \frac{1}{8+1} = 0,11;$$

$$I_8 = \frac{1}{8+1} = 0,11.$$

Для расчета абстрактности компонентов использовалась формула, приведенная ниже:

$$A \text{ (Абстрактность)} = \frac{N_a}{N_c},$$

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);

N_c – количество классов в компоненте;

N_a – количество абстрактных классов в компоненте.

Данная метрика также была рассчитана для каждого компонента сервиса:

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

$$A_2 = \frac{0}{1} = 0;$$

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

$$A_4 = \frac{1}{3} = 0,33;$$

$$A_5 = \frac{1}{4} = 0,25;$$

$$A_6 = \frac{1}{3} = 0,33;$$

$$A_7 = \frac{1}{4} = 0,25;$$

$$A_8 = \frac{1}{5} = 0,2.$$

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

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

где A – абстрактность;

I – устойчивость.

Был проведен расчёт данной метрики для каждого компонента.

$$D'_1 = |0,2 + 0,875 - 1| = 0,075;$$

$$D'_2 = |1 + 0 - 1| = 0;$$

$$D'_3 = |1 + 0,5 - 1| = 0,4;;$$

$$D'_4 = |0,33 + 0,2 - 1| = 0,47;$$

$$D'_5 = |0,25 + 0,083 - 1| = 0,667;$$

$$D'_6 = |0,33 + 0,143 - 1| = 0,527;$$

$$D'_7 = |0,25 + 0,11 - 1| = 0,64;$$

$$D'_8 = |0,2 + 0,11 - 1| = 0,69.$$

Данный показатель возможно представить при помощи диаграммы, изображенной на рисунке 27. В данной координатной стеке 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.

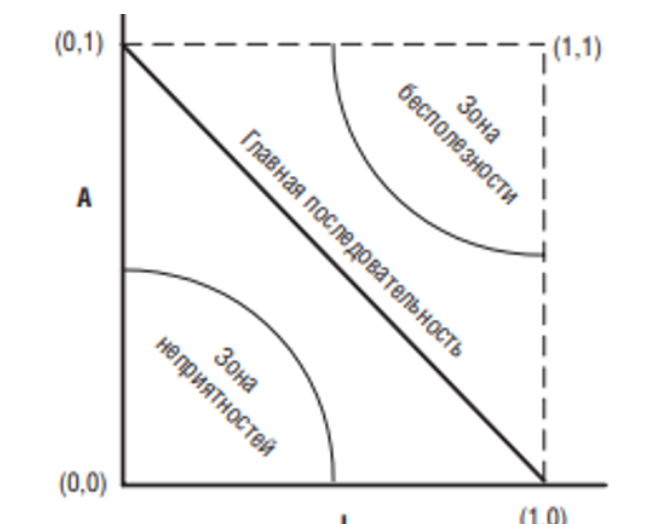


Рисунок 27 – Диаграмма возможных значений главной последовательности

По рассчитанным значениям для компонентов приложения был построен аналогичный график, изображенный на рисунке 28.

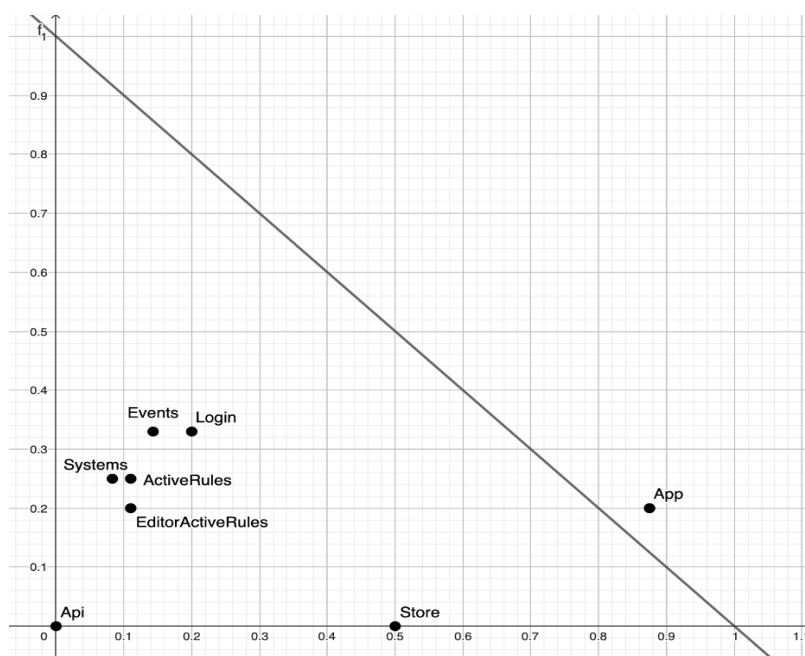


Рисунок 28 – Метрики кода на графе значений главной последовательности

Компоненты `Api` располагается далеко от главной последовательности и являются слишком устойчивыми. Это объясняется тем, что компонент реализует 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 компонента является верным.

4.3.2 Метрика Холстеда

Метрика Холстеда относится к метрикам, вычисляемым на основании анализа числа строк и синтаксических элементов исходного кода программы.

Для расчёта метрик Холстеда был взят компонент добавления нового активного правила. Код данного модуля представлен в листинге 4.

Листинг 4 – Код окна для добавления активного правила

```
import React, {FC, useMemo} from 'react';
import styles from './styles.module.scss';
import {Button, Form, Input, Modal, Select} from 'antd';
import {useAction} from '../../../hooks/useAction';
import {useAppDispatch, useTypedSelector} from
'../../../hooks/useTypedSelector';

interface AddActiveRuleModalInterface {
  isVisible: boolean;
  setIsVisible: (isVisible: boolean) => void;
  currentSystemCode: string;
}

const AddActiveRuleModal: FC<AddActiveRuleModalInterface> =
({isVisible, setIsVisible, currentSystemCode}) => {
  const {addActiveRule} = useAction();
  const dispatch = useAppDispatch();

  const handleCancel = () => {
    setIsVisible(false);
  };

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

  const eventOptions = useMemo(() => events
    .map(ev => {
      return {
        label: `${ev.description} - ${ev.categoryEvent}
```

```

event\n Code: ${ev.code}`,
      value: ev.code
    });
  }, [events]);

const onFinish = async (data: {
  description: string, condition: string,
  action: string, codeEvent: string, typeBind: string
}) => {
  const requestData = {
    description: data.description,
    condition: data.condition ? JSON.stringify({
      data: [
        {
          category: 'condition',
          description: data.condition,
          code: 'c00001'
        }
      ]
    }) : '',
    action: JSON.stringify({
      data: [
        {
          category: 'action',
          description: data.action,
          code: 'a00001',
        }
      ],
      edges: [
        {
          from: 'c00001',
          to: 'a00001'
        }
      ]
    })
  };

  const eventData = {
    codeEvent: data.codeEvent,
    typeBind: data.typeBind
  };

  await dispatch(() => addActiveRule(requestData, eventData,
currentSystemCode));

  setIsVisible(false);
};

return <>
  <Modal
    title="Добавить активное правило"
    open={isVisible}
  >

```

```

        onCancel={handleCancel}
        footer={[
            <Button type="primary" form="addSystemForm"
key="submit" htmlType="submit">
                Сохранить
            </Button>
        ]}
    >
    <Form
        name="addSystemForm"
        labelCol={{span: 6}}
        wrapperCol={{span: 16}}
        onFinish={onFinish}
        className={styles.form}
    >
        <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' }]}
        >
            <Input/>
        </Form.Item>
        <Form.Item
            label="Действие"
            name="action"
            rules={[{required: true, message: 'Заполните
поле' }]}
        >
            <Input/>
        </Form.Item>
        <Form.Item label="Событие"
            name="codeEvent"
            rules={[{required: true, message: 'Выберите
событие' }]}>
            <Select className={commonStyles.select}
                options={eventOptions}/>
        </Form.Item>

        <Form.Item label="Тип связи"
            name="typeBind"
            rules={[{required: true, message: 'Выберите
тип связи' }]}>
            <Select className={commonStyles.select}

```

```

        options={ [
            {
                value: 'Event to Rule',
                label: 'Событие к правилу'
            },
            {
                value: 'Rule to Event',
                label: 'Правило к событию'
            }
        ] } />
    </Form.Item>
</Form>
</Modal>
</>;
};

export default AddActiveRuleModal;

```

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
:	47
?	1
< >	24
=	9
()	17
.	26
=>	8
{ }	49
,	35
;	24
‘ ’	22
setVisible	4
addActiveRule	2

Продолжение таблицы 8

Операторы	Число вхождений
dispatch	2
handleCancel	2
onFinish	3
return	2
n1 = 18	N1 = 277

Таблица 9 – Операнды

Операнды	Число вхождений
isVisible	4
currentSystemCode	3
events	3
eventsOptions	2
requestData	2
eventData	2
data	13
n2 = 7	N2 = 29

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:

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

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:

- нажатие на клавишу клавиатуры (K): 0,28 секунд;
- нажатие на кнопку мыши (M): 0,1 секунда;
- перемещение курсора мыши (П): 1,1 секунды;
- перенос руки от мыши к клавиатуре и обратно (Р): 0,4 секунды;
- продолжительность выбора действия (Д): 1,2 секунды;
- время реакции системы (В): от 0,1 секунды.

Проведем тестирование рассмотренным методом функцию добавления активного правила в систему со следующими данными:

- название активного правила – «Включение увлажнителя»;
- названия условия активного правила – «Увлажнитель выключен»;
- названия действия активного правила – «Нажатие на кнопку off, выключение»;
- событие активного правила – «Измерение влажности»;
- тип связи – «Событие к правилу».

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 активное правило»
3	М	0,1	Нажатие кнопки мыши
4	В	0,1	Время реакции системы. Открывается всплывающее окно создания активного правила
5	П	1,1	Перемещение курсора к полю ввода названия активного правила
6	М	0,1	Нажатие на кнопку мыши. Перемещение фокуса на поле название
7	Р	0,4	Перенос руки с мыши на клавиатуру
8	К x 22	0,28x22	Ввод электронной почты из 21 символов и нажатия клавиши shift для ввода прописной буквы
9	Р	0,4	Перенос руки с клавиатуры на мышь
10	П	1,1	Перемещение курсора к полю ввода условия
11	М	0,1	Нажатие кнопки мыши. Перемещение фокуса на поле ввода условия
12	Р	0,4	Перенос руки с мыши на клавиатуру
13	К x 21	0,28x21	Ввод условия из 20 символов и нажатие 1 раз клавиши shift для ввода прописной буквы

Продолжение таблицы 10

№	Тип действия	Продолжительность	Комментарий
14	Р	0,4	Перенос руки с клавиатуры на мышь
15	П	1,1	Перемещение курсора к кнопке «Событие»
16	М	0,1	Нажатие кнопки мыши. Раскрытие списка доступных событий
17	П	1,1	Перемещение курсора к нужному событию
18	М	0,1	Нажатие кнопки мыши. Ввод события произошел
19	П	1,1	Перемещение курсора к кнопке «Тип связи»
20	М	0,1	Нажатие кнопки мыши. Раскрытие списка связей
21	П	1,1	Перемещение курсора к нужному типу связи
22	М	0,1	Нажатие кнопки мыши. Ввод типа связи произошел
23	П	1,1	Перемещение курсора к кнопке «Сохранить»
24	М	0,1	Нажатие кнопки мыши
25	В	0,3	Время реакции системы. Отправка данных на сервер и получение ответа. Последующее закрытие диалогового окна

В итоге, для реализации функции добавления нового активного правила в систему пользователь произвел 25 действия, затратив на данные операции суммарно 19,94 секунд. Объем затраченного времени на столь комплексную задачу невелик, что показывает хороший уровень построения юзабилити в разрабатываемых инструментальных средствах.

Заключение

В результате выполнения выпускной квалификационной работы были разработаны программные средства управления метаданными для сервиса потоковой обработки событий.

В ходе работы был проведен анализ предметной области и выделены функциональные возможности программных средств управления метаданными для сервиса потоковой обработки событий. Была описана архитектура ПС и сценарии работы. Также были описаны механизмы управления метаданными, их жизненный цикл и состав. Проведено планирование и оценка бюджета проекта при помощи средств MS Project. Затраты на проект составили 130200 рублей. Время разработки 56 рабочих дней.

Перед непосредственной разработкой был проведен этап проектирования программных средств, который включал в себя проектирование архитектуры и разработку функциональных требований. На этапе проектирования был обоснован выбор технологий и средств разработки, а также были выделены требования к аппаратно-программного обеспечению.

Основными этапами разработки стали: разработки архитектуры развертывания с использованием объединяющего три микрофронтенда Single Page Application, разработка средств регистрации и аутентификации пользователей, разработка пользовательского интерфейса, разработка средств управления системами, событиями и правилами, а также разработка различных видов визуальных представлений данных (списочное, табличное, тэговое).

Разработка проводилась в интегрированной среде разработки Webstorm на языке программирования TypeScript с использованием фреймворка React.

В заключительном этапе был проведен контроль качества разработанных программных средств. Для этого были составлены модульные тесты, а также проведено функциональное и модульное тестирования. Кроме того, были подсчитаны метрики устойчивости и абстрактности, метрики Холстеда, а также 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. 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.
2. Шибанов, С. В. Архитектура метаданных сервиса потоковой обработки событий и исполнения активных правил / С.В. Шибанов, А.С. Гусаров, Я.С. Шлепнев // Математическое моделирование и суперкомпьютерные технологии. – Н. Новгород : Изд-во НГУ, 2021 – С. 211-214.
3. Прошкина, Е. Н. Язык моделирования информационных процессов UML 2.0 / Е. Н. Прошкина, И. Ю. Балашова – Пенза : Изд-во ПГУ, 2018. – 112 с.
4. Шибанов, С. В. Сервис потоковой обработки событий и исполнения активных правил / С.В. Шибанов, Я.С. Шлепнёв // Математическое моделирование и суперкомпьютерные технологии. Труды XXI Международной конференции (Н. Новгород, 22-26 ноября 2021 г.) / Под ред. проф. Д.В. Баландина. – Нижний Новгород: Изд-во Нижегородского госуниверситета, 2021. – С. 403-407.
5. 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.
6. Database-per-service-pattern [Электронный ресурс]. URL <https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-data-persistence/database-per-service.html> (дата обращения 10.03.2023).
7. Шибанов, С. В. Мета модель сервиса потоковой обработки событий / С.В. Шибанов, Я.С. Шлепнёв, А.С. Гусаров – Пенза: Изд-во ПГУ, 2022 – С. 240 - 244.

8. Event stream processing [Электронный ресурс]. URL: <https://hazelcast.com/glossary/event-stream-processing/> (дата обращения 11.03.2023).
9. Rest API [Электронный ресурс]. URL: <https://starkovden.github.io/what-is-rest-api.html> (дата обращения 11.03.2023).
10. Sebastian Peyrott JWT Handbook // Auth0 Portal [Электронный ресурс]. URL: <https://auth0.com/resources/ebooks/jwt-handbook> (дата обращения 10.03.2023).
11. Использование MS Project для управления проектами по разработке ПО [Электронный ресурс]. URL: <https://habr.com/ru/post/151593/> (дата обращения: 20.05.2023).
12. Single-page application documentation [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/azure/active-directory/develop/index-spa> (дата обращения: 23.04.2023).
13. Redis documentation [Электронный ресурс]. URL: <https://redis.io/docs/manual/> (дата обращения: 13.04.2023).
14. Значения метрик кода [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/visualstudio/code-quality/code-metrics-values?view=vs-2022> (дата обращения: 23.05.2023).
15. WebdriverIO documentation [Электронный ресурс]. URL: <https://webdriver.io/docs/why-webdriverio> (дата обращения: 23.05.2023).

Приложение А
ГЛОССАРИЙ
(справочное)

Event condition action (ECA) – модель структуры активных правил, в которой содержатся три главных элемента: событие, условие, действие;

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

Create, Read, Update, Delete (CRUD) – акроним, обозначающий четыре базовые функции, используемые при работе с базами данных: создание (англ. create), чтение (read), модификация (update), удаление (delete);

JSON Web Token (JWT) – открытый стандарт (RFC 7519) для создания токенов доступа, основанный на формате JSON, который, как правило, используется для передачи данных для аутентификации в клиент-серверных приложениях;

Single Page Application (SPA) – веб-приложение, использующее единственный файл разметки в качестве оболочки для всех динамически встраиваемых веб-страниц;

Representational State Transfer (REST) – стиль архитектуры взаимодействия компонентов приложения, набор правил организации кода программы.

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

БЭМ методология – используется для именования стилей, основанная на компонентном подходе, чтобы обеспечить понятность и структурированность стилей.

Приложение Б
ЛИСТИНГ МОДЕЛЕЙ СИСТЕМ И СОБЫТИЙ
(рекомендуемое)

Листинг Б.1 – Файл SystemPage/index.tsx

```

import React, {useEffect, useState} from 'react';
import MainTemplate from '../..../common/MainTemplate';
import stylesCommon from '../..../common/styles/styles.module.scss';
import styles from './styles.module.scss';
import Managing from './components/RightPanel/Managing';
import Elements from './components/RightPanel/Elements';
import Information from './components/RightPanel/Information';
import AddButton from './components/AddButton';
import Header from '../..../common/components/Header';
import SystemElements from './components/SystemElements';
import {useAppDispatch, useTypedSelector} from
'../..../hooks/useTypedSelector';
import {useAction} from '../..../hooks/useAction';
import {setCurrentSystems, setSystemPagesWay} from
'../..../store/action-creators/systems';
import AddModal from './components/AddModal';
import {SystemService} from '../..../api/services/SystemService';

const SystemRightPanel = () => {
  return <>
    <div className={stylesCommon.rightPanelBlock}>
      <Header/>
      <hr className={stylesCommon.line}/>
      <Managing/>
      <hr className={stylesCommon.line}/>
      <Elements/>
      <hr className={stylesCommon.line}/>
      <Information/>
    </div>
  </>;
};

const SystemPage = () => {
  const {
    isLoading,
    error,
    currentSystem,
    systemPagesWay
  } = useTypedSelector(state => state.systemsValues);
  const {fetchSystems, setSystemPagesWay, setCurrentSystems} =
useAction();
  const dispatch = useAppDispatch();

  const [isAddModalVisible, setIsAddModalVisible] =
useState(false);

  useEffect(() => {
    dispatch(() => fetchSystems());
  }, []);

```

```

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

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

    const onDoubleClick = async (elementChildren: any[]) => {
      const children = await
SystemService.fetchSystemByCode(currentSystem.code);
      dispatch(() => setSystemPagesWay([...systemPagesWay, {
        name: systemPagesWay.length < 1 ? currentSystem.name :
` / ${currentSystem.name}`,
        code: currentSystem.code,
        systems: children.data.children
      }]]));
      setCurrentSystems(children.data.children);
    };

    return <MainTemplate blocks={<SystemRightPanel/>} id="systems-
page">
      <div className={styles.addButton}>
        <AddButton onClick={() =>
setIsAddModalVisible(true)}>/>
      </div>
      <SystemElements onDoubleClick={onDoubleClick}>/>
      <AddModal isVisible={isAddModalVisible}
setIsVisible={setIsAddModalVisible}>/>
    </MainTemplate>;
  };

export default SystemPage;

```

Листинг Б.2 – Файл SystemElemens/index.tsx

```

import React, {FC} from 'react';
import commonStyles from
'../../../../../common/styles/styles.module.scss';
import {Switch} from 'antd';
import {useTypedSelector} from
'../../../../../hooks/useTypedSelector';
import {useAction} from ' ../../../../../hooks/useAction';
import variables from ' ../../../../../variables.module.scss';

interface SystemElementsInterface {
  onDoubleClick: (elementChildren: any[]) => void;
}

const SystemElements: FC<SystemElementsInterface> =
({onDoubleClick}) => {

```

```

    const {currentSystem, currentSystems} = useTypedSelector(state
=> state.systemsValues);
    const {setCurrentSystem, setCurrentActiveRule} = useAction();

    return <div className={commonStyles.elementsContainer}
id="systems-container">
      {currentSystems?.map(element => (
        <div
          className={commonStyles.element}
          style={currentSystem.code === element.code
            ? {
              background: variables.yellowColor,
              color: variables.darkBlueColor
            }
            : undefined}
          key={element.code}
          id={`system-element-${element.code}`}
          onClick={() => {
            /*@ts-ignore*/
            setCurrentSystem({...element,
currentSystems});
            setCurrentActiveRule(null);
          }}
          // @ts-ignore
          onDoubleClick={() =>
onDoubleClick(element.children)}
        >
          <div className={commonStyles.text}>
            /*@ts-ignore*/
            <div>{element.name}</div>
            <div className={commonStyles.textId}
              style={currentSystem.code ===
element.code
                ? {
                  color: variables.greyColor
                }
                : undefined}
            >
              Id: {element.code}
            </div>
          </div>
          /*@ts-ignore*/
          <div className={commonStyles.switch}><Switch
size="small" checked={element?.IsCheck}/></div>
        </div>
      )})
    </div>;
  };

export default SystemElements;

```

Листинг Б.3 – Файл AddModal/index.tsx

```

import React, {FC} from 'react';
import styles from './styles.module.scss';
import {Button, Form, Input, Modal} from 'antd';
import {useAction} from '../../../hooks/useAction';
import {useAppDispatch, useTypedSelector} from
 '../../../hooks/useTypedSelector';

interface AddModalInterface {
  isVisible: boolean;
  setIsVisible: (isVisible: boolean) => void;
}

const AddModal: FC<AddModalInterface> = ({isVisible,
setIsVisible}) => {
  const {addSystem} = useAction();
  const dispatch = useAppDispatch();
  const {currentSystem, systems} = useTypedSelector(store =>
store.systemsValues);

  const handleCancel = () => {
    setIsVisible(false);
  };

  const onFinish = async (body: {name: string}) => {
    await dispatch(() => addSystem(body.name, currentSystem));

    setIsVisible(false);
  };

  return <>
    <Modal
      title="Добавить систему"
      open={isVisible}
      onCancel={handleCancel}
      footer={[
        <Button type="primary" form="addSystemForm"
key="submit" htmlType="submit">
          Сохранить
        </Button>
      ]}
    >
      <Form
        name="addSystemForm"
        labelCol={{span: 6}}
        wrapperCol={{span: 16}}
        onFinish={onFinish}
        className={styles.form}
      >
        <Form.Item

```

```

        label="Название"
        name="name"
        rules={[{required: true, message: 'Заполните
поле'}}]}
      >
        <Input/>
      </Form.Item>
    </Form>
  </Modal>
</>;
};

export default AddModal;

```

Листинг Б.4 – Файл RightPanel/index.tsx

```

import Icon from '../../common/components/Icon';
import React from 'react';
import stylesCommon from
'../../common/styles/styles.module.scss';
import { Link } from 'react-router-dom';
import variables from '../../variables.module.scss';
import {useTypedSelector} from
'../../hooks/useTypedSelector';

const Elements = () => {
  const {currentSystem} = useTypedSelector(state =>
state.systemsValues);
  const onClickLink = () => {
    localStorage.removeItem('currentSystemCode');
    localStorage.removeItem('currentSystemName');
    localStorage.setItem('currentSystemCode',
currentSystem.code);
    localStorage.setItem('currentSystemName',
currentSystem.name);
  };

  return (
    <>
      <div
className={stylesCommon.rightPanelBlockTitle}>Элементы
системы</div>
      <div className={stylesCommon.rightPanelBlockAction}>
        <Icon name="events"/>
        <div
className={stylesCommon.rightPanelBlockActionText}>
          <Link
            id="events-link"
            style={{textDecoration: 'none', color:
variables.yellowColor}}
            to="/events"
            onClick={onClickLink}

```

```

        >
        Список событий
      </Link>
    </div>
  </div>
  <div className={stylesCommon.rightPanelBlockAction}>
    <Icon name="study"/>
    <div
      className={stylesCommon.rightPanelBlockActionText}>
        <Link
          style={{textDecoration: 'none', color:
variables.yellowColor}}
          onClick={onClickLink}
          to="/active-rules"
          id="active-rule-link"
        >
          Список правил
        </Link>
      </div>
    </div>
  </>
);
};

export default Elements;

```

Листинг Б.5 – Файл SystemService.ts

```

import {AxiosResponse} from 'axios';
import api from '../http';
import {AddSystemInterface, DeleteSystemInterface,
SystemsInterface} from '../../store/types/systemsTypes';

export class SystemService {
  static async fetchSystems(attached: boolean = true):
Promise<AxiosResponse<SystemsInterface[]>> {
    return
    api.get<SystemsInterface[]>(`/system?attached=${attached}`);
  }

  static async fetchSystemByCode(systemCode: string, attached:
boolean = true): Promise<AxiosResponse<SystemsInterface>> {
    return
    api.get<SystemsInterface>(`/system/${systemCode}?attached=${attach
ed}`);
  }

  static async addSystem(body: AddSystemInterface):
Promise<AxiosResponse<any[]>> {
    return api.post<any[]>('/system', body);
  }
}

```

```

    static async addSystemAttachment(body: {
      codeParent: string,
      codeAttach: string
    }): Promise<AxiosResponse<any[]>> {
      return api.post<any[]>('/system/attachment', body);
    }

    static async deleteSystem(data: DeleteSystemInterface):
    Promise<AxiosResponse<any[]>> {
      return api.delete<any[]>('/system', {data});
    }
  }
}

```

Листинг Б.6 – Файл ComponentService.ts

```

import {AxiosResponse} from 'axios';
import api from '../http';
import {ComponentInterface} from
'../../store/types/componentTypes';

export class ComponentService {
  static async fetchComponent():
  Promise<AxiosResponse<ComponentInterface[]>> {
    return api.get<ComponentInterface[]>('/component');
  }

  static async addComponent(componentName: string):
  Promise<AxiosResponse<any[]>> {
    return api.post('/component', {description:
    componentName});
  }

  static async addComponentSystem(body: {
    codeComponent: string,
    codeSystem: string
  }): Promise<AxiosResponse<any[]>> {
    return api.post('/component/system', body);
  }
}

```

Листинг Б.7 – Файл action-creators/systems.ts

```

import {
  AddSystemInterface, DeleteSystemInterface,
  SystemPagesWayInterface,
  SystemsAction,
  SystemsActionTypes,
  SystemsInterface
} from '../types/systemsTypes';
import {Dispatch} from 'redux';
import {SystemService} from '../../api/services/SystemService';

```

```

import {ActiveRulesInterface} from '../types/activeRulesTypes';

export const setCurrentSystem = (system: SystemsInterface):
SystemsAction => {
    return {type: SystemsActionTypes.SET_CURRENT_SYSTEM, payload:
system};
};

export const setCurrentSystems = (systems: SystemsInterface[] |
ActiveRulesInterface[]): SystemsAction => {
    return {type: SystemsActionTypes.SET_CURRENT_SYSTEMS, payload:
systems};
};

export const setSystemPagesWay = (systemPagesWay:
SystemPagesWayInterface[]): SystemsAction => {
    return {type: SystemsActionTypes.SET_SYSTEM_PAGES_WAY,
payload: systemPagesWay};
};

export const fetchSystems = (): any => {
    return async (dispatch: Dispatch<SystemsAction>) => {
        try {
            dispatch({type: SystemsActionTypes.FETCH_SYSTEM});
            const response = await
SystemService.fetchSystems(true);
            if (response) {
                dispatch({type:
SystemsActionTypes.FETCH_SYSTEM_SUCCESS, payload: response.data});
            }
        } catch (e) {
            dispatch({type: SystemsActionTypes.FETCH_SYSTEM_ERROR,
payload: 'Произошла ошибка загрузки систем: ' + e});
        }
    };
};

export const addSystem = (name: string, system: SystemsInterface)
=> {
    return async (dispatch: Dispatch<SystemsAction>) => {
        try {
            const newSystem = await
SystemService.addSystem({name});
            if (newSystem) {
                //@ts-ignore
                await
SystemService.addSystemAttachment({codeParent: system.code,
codeAttach: newSystem.data.code});
                const updatedSystem = await
SystemService.fetchSystemByCode(system.code);
                dispatch(fetchSystems());
                if(updatedSystem) {

```

```

dispatch(setCurrentSystems(updatedSystem.data.children));
    }
  } catch (e) {
    console.error('Произошла ошибка добавления системы: ',
e);
  }
};

};

export const deleteSystem = (body: DeleteSystemInterface,
parentCode?: string) => {
  return async (dispatch: Dispatch<SystemsAction>) => {
    try {
      await SystemService.deleteSystem(body);
      if(parentCode) {
        const updatedParentSystem = await
SystemService.fetchSystemByCode(parentCode);
        dispatch(fetchSystems());
        if (updatedParentSystem) {
          dispatch(setCurrentSystems(updatedParentSystem.data.children));
        }
      } catch (e) {
        console.error('Произошла ошибка удаления системы: ',
e);
      }
    }
  };
};

```

Листинг Б.8 – Файл action-creators/components.ts

```

import {Dispatch} from 'redux';
import {ComponentAction, ComponentActionTypes} from
'../types/componentTypes';
import {ComponentService} from
'../../api/services/ComponentService';
import {SystemsAction} from '../types/systemsTypes';
import {fetchSystems} from '../systems';

export const fetchComponentParams = (): any => {
  return async (dispatch: Dispatch<ComponentAction>) => {
    try {
      dispatch({type:
ComponentActionTypes.FETCH_COMPONENT});
      const response = await
ComponentService.fetchComponent();
      if (response) {
        dispatch({type:
ComponentActionTypes.FETCH_COMPONENT_SUCCESS, payload:

```

```

response.data));
    }
    } catch (e) {
        dispatch({type:
ComponentActionTypes.FETCH_COMPONENT_ERROR, payload: 'Произошла
ошибка загрузки контекста событий: ' + e});
    }
};
};

```

Листинг Б.9 – Файл EventsPage/index.tsx

```

import React, {useEffect} from 'react';
import {useAppDispatch, useTypedSelector} from
'../../hooks/useTypedSelector';
import {useAction} from ' ../../hooks/useAction';
import EventsElements from './components/EventsElements';
import RightPanel from './components/RightPanel';
import MainTemplate from ' ../../common/MainTemplate';

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

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

        currentSystemCode && dispatch(() =>
fetchEventsBySystemCode(currentSystemCode));
    }, [currentSystem]);

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

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

    return <MainTemplate blocks={<RightPanel/>} id="events-page">
        <EventsElements />

```

```

    </MainTemplate>;

};

export default EventsPage;

```

Листинг Б.10 – Файл EventsElements/index.tsx

```

import React from 'react';
import commonStyles from
'../../../../../common/styles/styles.module.scss';
import {useTypedSelector} from
'../../../../../hooks/useTypedSelector';
import variables from ' ../../../../../variables.module.scss';
import {useAction} from ' ../../../../../hooks/useAction';

const EventsElements = () => {
    const {events, currentEvent} = useTypedSelector(state =>
state.eventsValues);
    const {setCurrentEvent} = useAction();

    return <>
        <div className={commonStyles.elementsContainer}
id="events-container">
            {events?.map(event => (
                <div className={commonStyles.element}
                style={currentEvent.code === event.code
                    ? {
                        background: variables.yellowColor,
                        color: variables.darkBlueColor
                    }
                    : undefined}
                onClick={() => setCurrentEvent({...event})}
                key={event.code}
                id={event.description}
            >
                <div className={commonStyles.text}>
                    <div>{event.description}</div>
                    <div
                        className={commonStyles.textId}
                        style={currentEvent.code ===
event.code
                            ? {
                                color: variables.greyColor
                            }
                            : undefined}
                    >
                        Id: {event.code}
                    </div>
                </div>
            </div>
        </div>
    ) )}

```

```

        </div>
    </>;
};

export default EventsElements;

```

Листинг Б.11 – Файл EventsElements/index.tsx

```

import React, {FC, useEffect, useMemo} from 'react';
import styles from './styles.module.scss';
import {Button, Form, Input, Modal} from 'antd';
import Select from 'react-select';
import {useAction} from '../../../hooks/useAction';
import {useAppDispatch, useTypedSelector} from
'../../../hooks/useTypedSelector';

interface AddEventModalInterface {
    isVisible: boolean;
    setIsVisible: (isVisible: boolean) => void;
    currentSystemCode: string;
}

const AddEventModal: FC<AddEventModalInterface> = ({isVisible,
setIsVisible, currentSystemCode}) => {
    const {addEvent, fetchContextParams, fetchComponentParams} =
useAction();
    const {contextParams} = useTypedSelector(store =>
store.contextParamsValues);
    const {component} = useTypedSelector(store =>
store.componentValues);
    const dispatch = useAppDispatch();

    useEffect(() => {
        dispatch(() => fetchContextParams());
        dispatch(() => fetchComponentParams());
    }, [dispatch]);

    const contextParamsOptions = useMemo(() => {
        return contextParams.map(param => {
            return {
                label: param.name,
                value: param.code
            };
        });
    }, [contextParams]);

    const componentOptions = useMemo(() => {
        return component.map(comp => {
            return {
                label: comp.description,
                value: comp.code
            };
        });
    }, [component]);

```

```

        };
    });
}, [component]);

const handleCancel = () => {
    setIsVisible(false);
};

const onFinish = async (data: {
    description: string,
    contextParamCode: { label: string, value: string },
    codeComponent: { label: string, value: string },
}) => {
    const requestData = {
        ...data,
        contextParamCode: data.contextParamCode.value,
        codeComponent: data.codeComponent.value
    };

    await dispatch(() => addEvent(requestData,
currentSystemCode));

    setIsVisible(false);
};

return <>
    <Modal
        title="Добавить событие"
        open={isVisible}
        onCancel={handleCancel}
        footer={[
            <Button type="primary" form="addSystemForm"
key="submit" htmlType="submit" id="save-event-button">
                Сохранить
            </Button>
        ]}
    >
        <Form
            name="addSystemForm"
            labelCol={{span: 8}}
            wrapperCol={{span: 14}}
            onFinish={onFinish}
            className={styles.form}
        >
            <Form.Item
                label="Название"
                name="description"
                rules={[{required: true, message: 'Заполните
поле' }]}
            >
                <Input id="events-input-name"/>
            </Form.Item>

```

```

        <Form.Item label="Контекст"
                    name="contextParamCode"
                    rules={[{required: true, message:
'Выберите контекст'}}]>
            <Select
                id="events-select-context"
                options={contextParamsOptions}
            />
        </Form.Item>
        <Form.Item label="Компонент"
                    name="codeComponent"
                    rules={[{required: true, message:
'Выберите компонент'}}]>
            <Select options={componentOptions}
                id="events-select-component"
            />
        </Form.Item>
    </Form>
</Modal>
</>;
};

export default AddEventModal;

```

Листинг Б.12 – Файл EventsService.ts

```

import {AxiosResponse} from 'axios';
import api from '../http';
import {AddEventInterface, DeleteEventInterface, EventsInterface}
from '../store/types/eventsTypes';

export class EventsService {
    static async fetchEventsBySystemCode(code: string = '00016'):
    Promise<AxiosResponse<EventsInterface[]>> {
        return
        api.get<EventsInterface[]>(`/event/system/${code}`);
    }

    static async addEvent(body: AddEventInterface):
    Promise<AxiosResponse<any[]>> {
        return api.post<any[]>('/event/atomic', body);
    }

    static async deleteEvent(data: DeleteEventInterface):
    Promise<AxiosResponse<any[]>> {
        return api.delete<any[]>('/event/atomic', {data});
    }
}

```

Листинг Б.13 – Файл `acriion-creators/events.ts`

```

import {Dispatch} from 'redux';
import {
  AddEventInterface,
  DeleteEventInterface,
  EventsAction,
  EventsActionTypes,
  EventsInterface
} from '../types/eventsTypes';
import {EventsService} from '../../api/services/EventsService';

export const setCurrentEvent = (event: EventsInterface):
EventsAction => {
  return {type: EventsActionTypes.SET_CURRENT_EVENT, payload:
event};
};

export const fetchEventsBySystemCode = (code: string): any => {
  return async (dispatch: Dispatch<EventsAction>) => {
    try {
      dispatch({type:
EventsActionTypes.FETCH_EVENTS_SYSTEM_CODE});
      const response = await
EventsService.fetchEventsBySystemCode(code);
      if (response) {
        dispatch({type:
EventsActionTypes.FETCH_EVENTS_SYSTEM_CODE_SUCCESS, payload:
response.data});
      }
    } catch (e) {
      dispatch({type:
EventsActionTypes.FETCH_EVENTS_SYSTEM_CODE_ERROR, payload:
'Произошла ошибка загрузки систем: ' + e});
    }
  };
};

export const addEvent = (body: AddEventInterface,
currentSystemCode: string) => {
  return async (dispatch: Dispatch<EventsAction>) => {
    try {
      await EventsService.addEvent(body);
      dispatch(fetchEventsBySystemCode(currentSystemCode));
    } catch (e) {
      console.error('Произошла ошибка добавления события: ',
e);
    }
  };
};

```

```
export const deleteEvent = (body: DeleteEventInterface,  
currentSystemCode: string) => {  
  return async (dispatch: Dispatch<EventsAction>) => {  
    try {  
      await EventsService.deleteEvent(body);  
      dispatch(fetchEventsBySystemCode(currentSystemCode));  
    } catch (e) {  
      console.error('Произошла ошибка удаления события: ',  
e);  
    }  
  };  
};
```

Приложение В
ЛИСТИНГ МОДЕЛИ АКТИВНЫХ ПРАВИЛ
(рекомендуемое)

Листинг В.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 В.2 – Файл AddActiveRuleModal/index.tsx

```

import React, {FC, useMemo} from 'react';
import styles from './styles.module.scss';
import {Button, Form, Input, Modal, Select} from 'antd';
import {useAction} from '../../../hooks/useAction';
import {useAppDispatch, useTypedSelector} from
'../../../hooks/useTypedSelector';
import commonStyles from
'../../../common/styles/styles.module.scss';

interface AddActiveRuleModalInterface {
  isVisible: boolean;
  setIsVisible: (isVisible: boolean) => void;
  currentSystemCode: string;
}

const AddActiveRuleModal: FC<AddActiveRuleModalInterface> =
(({isVisible, setIsVisible, currentSystemCode}) => {
  const {addActiveRule} = useAction();
  const dispatch = useAppDispatch();

  const handleCancel = () => {
    setIsVisible(false);
  };

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

  const eventOptions = useMemo(() => events
    .map(ev => {
      return {
        label: `${ev.description} - ${ev.categoryEvent}
event\n Code: ${ev.code}`,
        value: ev.code
      };
    }), [events]);

```

```

const onFinish = async (data: {
  description: string, condition: string,
  action: string, codeEvent: string, typeBind: string
}) => {
  const requestData = {
    description: data.description,
    condition: data.condition ? JSON.stringify({
      data: [
        {
          category: 'condition',
          description: data.condition,
          code: 'c00001'
        }
      ]
    }) : '',
    action: JSON.stringify({
      data: [
        {
          category: 'action',
          description: data.action,
          code: 'a00001',
        }
      ],
      edges: [
        {
          from: 'c00001',
          to: 'a00001'
        }
      ]
    })
  };

  const eventData = {
    codeEvent: data.codeEvent,
    typeBind: data.typeBind
  };

  await dispatch(() => addActiveRule(requestData, eventData,
currentSystemCode));

  setIsVisible(false);
};

return <>
  <Modal
    title="Добавить активное правило"
    open={isVisible}
    onCancel={handleCancel}
    footer={[
      <Button type="primary" form="addSystemForm"
key="submit" htmlType="submit">

```

```

        Сохранить
    </Button>
  ]}
>
  <Form
    name="addSystemForm"
    labelCol={{span: 6}}
    wrapperCol={{span: 16}}
    onFinish={onFinish}
    className={styles.form}
  >
    <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'}}]
    >
      <Input/>
    </Form.Item>
    <Form.Item
      label="Действие"
      name="action"
      rules={[{required: true, message: 'Заполните
поле'}}]
    >
      <Input/>
    </Form.Item>
    <Form.Item label="Событие"
      name="codeEvent"
      rules={[{required: true, message:
'Выберите событие'}}]
    >
      <Select className={commonStyles.select}
        options={eventOptions}/>
    </Form.Item>

    <Form.Item label="Тип связи"
      name="typeBind"
      rules={[{required: true, message:
'Выберите тип связи'}}]
    >
      <Select className={commonStyles.select}
        options=[
          {
            value: 'Event to Rule',
            label: 'Событие к правилу'

```

```

        },
        {
            value: 'Rule to Event',
            label: 'Правило к событию'
        }
    ]}]/>
</Form.Item>
</Form>
</Modal>
</>;
};

export default AddActiveRuleModal;

```

Листинг В.3 – Файл RuleTags/index.tsx

```

import React from 'react';
import commonStyles from
'../../../../../../common/styles/styles.module.scss';
import {useAppDispatch, useTypedSelector} from
'../../../../../../hooks/useTypedSelector';
import RulePerformanceSelect from '../../../../RulePerformanceSelect';
import {setCurrentActiveRule} from '../../../../store/action-
creators/activeRules';
import {useAction} from '../../../../hooks/useAction';
import variables from '../../../../variables.module.scss';
import {useNavigate} from 'react-router-dom';

const RuleTags = () => {
    const {activeRules, currentActiveRule} =
useTypedSelector(state => state.activeRulesValues);
    const {setCurrentActiveRule} = useAction();
    const dispatch = useAppDispatch();
    const navigate = useNavigate();

    return <>
        <div className={commonStyles.toolbar}>
            <RulePerformanceSelect/>
        </div>
        <div className={commonStyles.elementsContainer}>
            {activeRules?.map(rule => (
                <div
                    className={commonStyles.element}
                    style={currentActiveRule?.code === rule.code
                        ? {
                            background: variables.yellowColor,
                            color: variables.darkBlueColor
                        }
                        : undefined}
                    onClick={() => dispatch(() =>
setCurrentActiveRule(rule))}
                    onDoubleClick={() => {

```

```

localStorage.removeItem('currentActiveRuleObject');
                                currentActiveRule &&
localStorage.setItem('currentActiveRuleObject',
JSON.stringify(currentActiveRule));
                                navigate('/editor-active-rules');
                                }}
                                key={rule.code}
                                id={`active-rule-element-${rule.code}`}
                                >
                                <div className={commonStyles.text}>
                                <div>{rule.description}</div>
                                <div
                                className={commonStyles.textId}
                                style={currentActiveRule?.code ===
rule.code
                                ? {
                                color: variables.greyColor
                                }
                                : undefined}
                                >
                                Id: {rule.code}
                                </div>
                                </div>
                                </div>
                                </div>
                                )}}
                                </div>
                                </>;
};

export default RuleTags;

```

Листинг В.4 – Файл ActiveRulesService.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);
    }
}

```

Листинг В.5 – Файл action-creators/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 В.6 – 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 В.7– 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 В.8– Файл EditorRulesList/index.tsx

```

import React from 'react';
import MainTemplate from '../.../.../common/MainTemplate';
import commonStyles from
'../.../.../.../common/styles/styles.module.scss';
import Header from '../.../.../.../common/components/Header';
import Managing from
'../.../.../ActiveRulesPage/components/RightPanel/Managing';
import Icon from '../.../.../.../common/components/Icon';
import variables from '../.../.../.../variables.module.scss';
import Events from './components/Events';
import Conditions from './components/Conditions';
import Actions from './components/Actions';
import EditorRulePerformanceSelect from
'../.../EditorRulePerformanceSelect';

const EditorRuleList = ({}) => {
    const RulesListRightPanel = () => {
        return (<>
            <div className={commonStyles.rightPanelBlock}>
                <Header/>
                <hr className={commonStyles.line}/>
                <Managing/>
                <hr className={commonStyles.line}/>
            </div>
        </>
    );
};

```

```

        <div
className={commonStyles.rightPanelBlockTitle}>Управление
элементом</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={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}>
                            Удалить
                        </div>
                    </div>
                </div>
            </>);
    };

    return <MainTemplate blocks={<RulesListRightPanel/>}
id="editor-active-rules-container">
        <div className={commonStyles.toolbar}>
            <EditorRulePerformanceSelect/>
        </div>
        <Events/>
        <Conditions/>

```

```

        <Actions/>
    </MainTemplate>;
};

export default EditorRuleList;

```

Листинг В.9 – Файл Conditions.ts

```

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 : ''
            };
        } else {
            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;
        }
      });

      return <div key={cond.code}
className={styles.card__content}>
        <div>
          {cond.category &&
            <div
className={cn(styles.card__category,
styles.card__category_condition)}>
{cond.category.at(0)?.toUpperCase()}
            </div>
          {cond.description} {cond.code}
        </div>
        <div className={styles.card__tags}>
          Подключенные события:
{filteredEventEdges.map((ev: string) =>
            <div key={ev}

```

```

className={cn(styles.card__category,
styles.card__category_eventTag)}>
    {ev}
  </div>
)}
</div>
<div className={styles.card__tags}>
  Результирующие воздействия:
{filteredActionEdges.map((ev: string) =>
  <div key={ev}
className={cn(styles.card__category,
styles.card__category_actionTag)}>
    {ev}
  </div>
)}
</div>

```

Приложение Г

ПРИМЕР СИСТЕМЫ ДЛЯ ТЕСТИРОВАНИЯ ПРИЛОЖЕНИЯ

(рекомендуемое)

Название приложения: Система мониторинга и контроля безопасности и обеспечения жизнедеятельности предприятия.

Назначение: Мониторинг и контроль пропускного режима сотрудников предприятия, поддержка системы жизнеобеспечения предприятия (температурный режим, обогрев и кондиционирование помещений, контроль электропотребления).

Общее описание: 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.

Функциональные требования: осуществляется мониторинг и контроль пропускного режима:

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

б) 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;

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

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

д) 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);

е) 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:

а) 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;

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

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

г) 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:

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

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

в) 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.

Реализация бизнес-логики мониторинга и управления: 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:

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

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

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

г) 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 Д
РЕЗУЛЬТАТЫ ФУНКЦИОНАЛЬНОГО ТЕСТИРОВАНИЯ
(рекомендуемое)

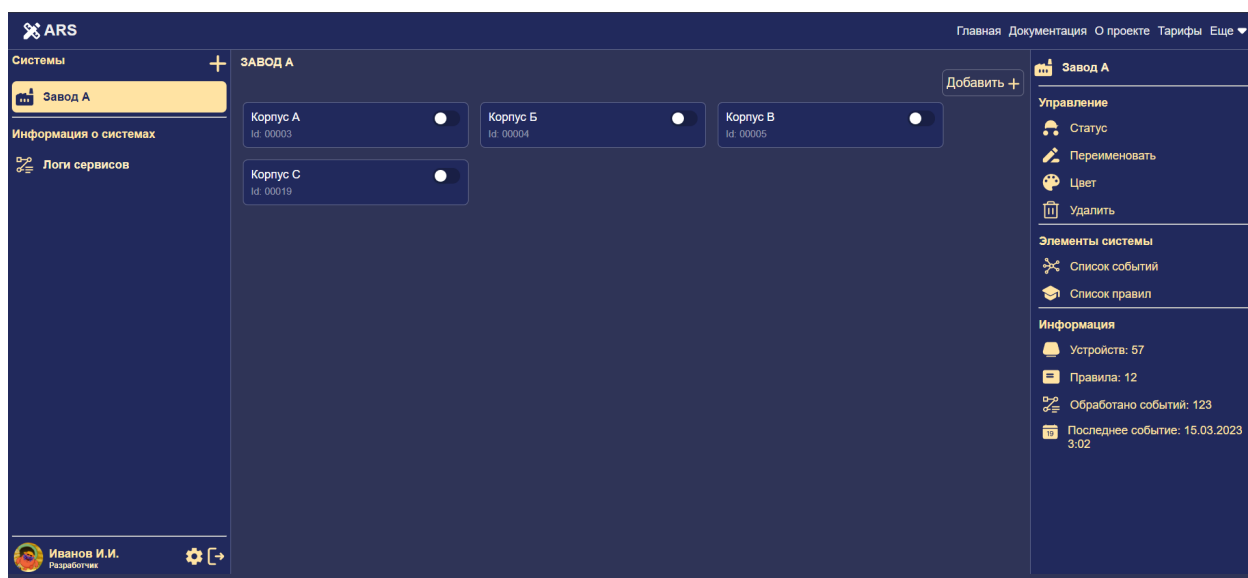


Рисунок Д.1 – Результат теста создания новой подсистемы

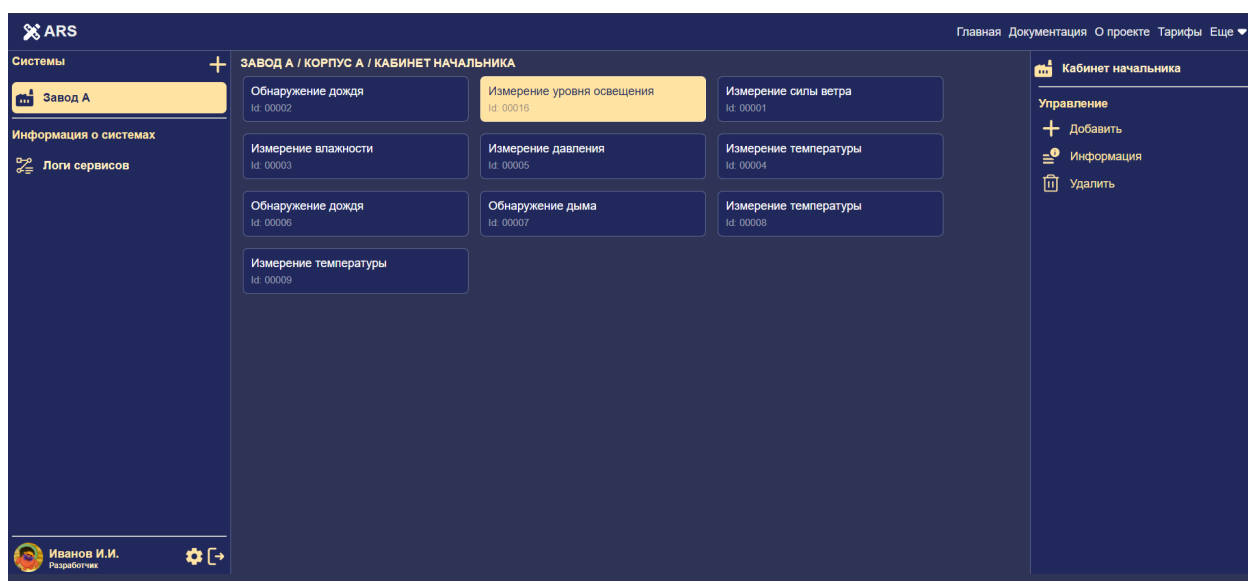


Рисунок Д.2 – Результат теста создания нового события

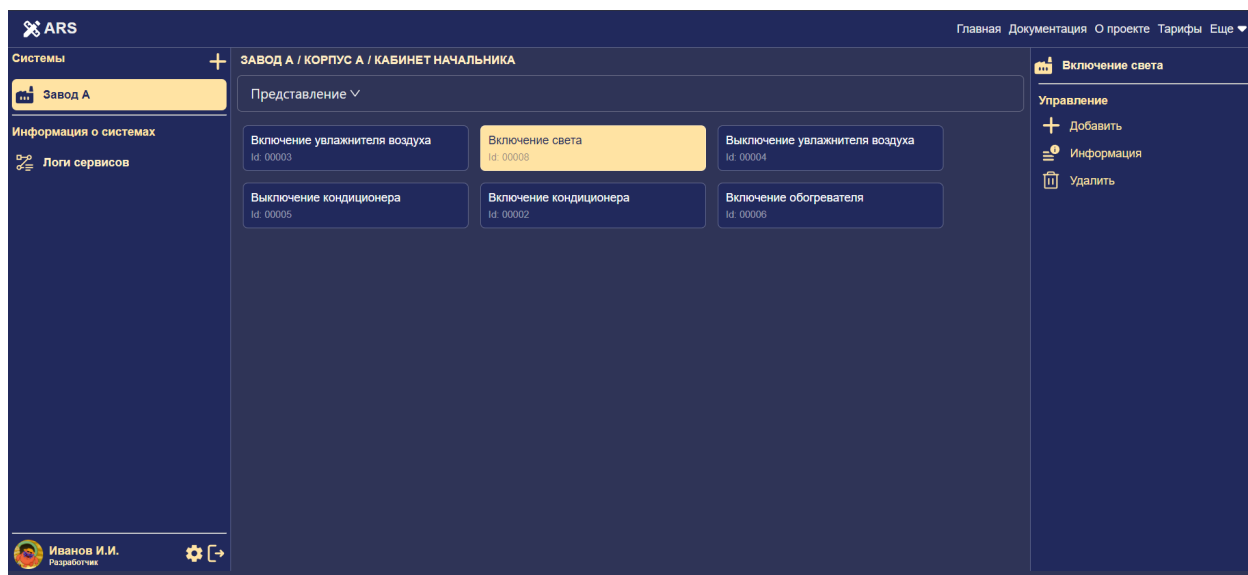


Рисунок Д.3 – Результат теста создания нового активного правила

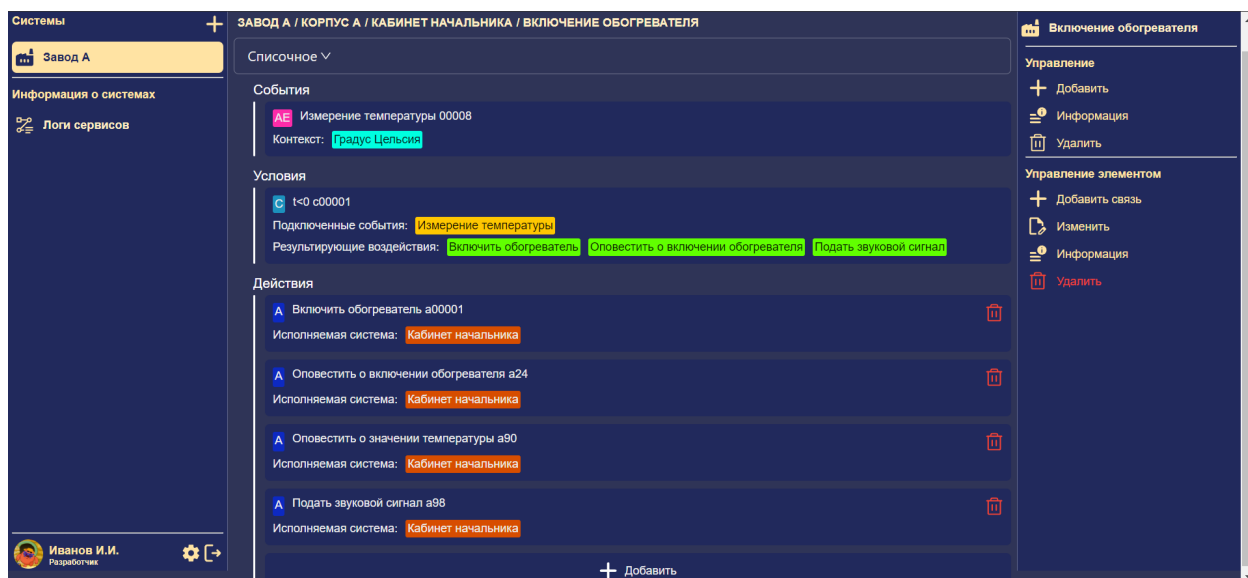


Рисунок Д.4 – Результат теста создания нового действия активного правила

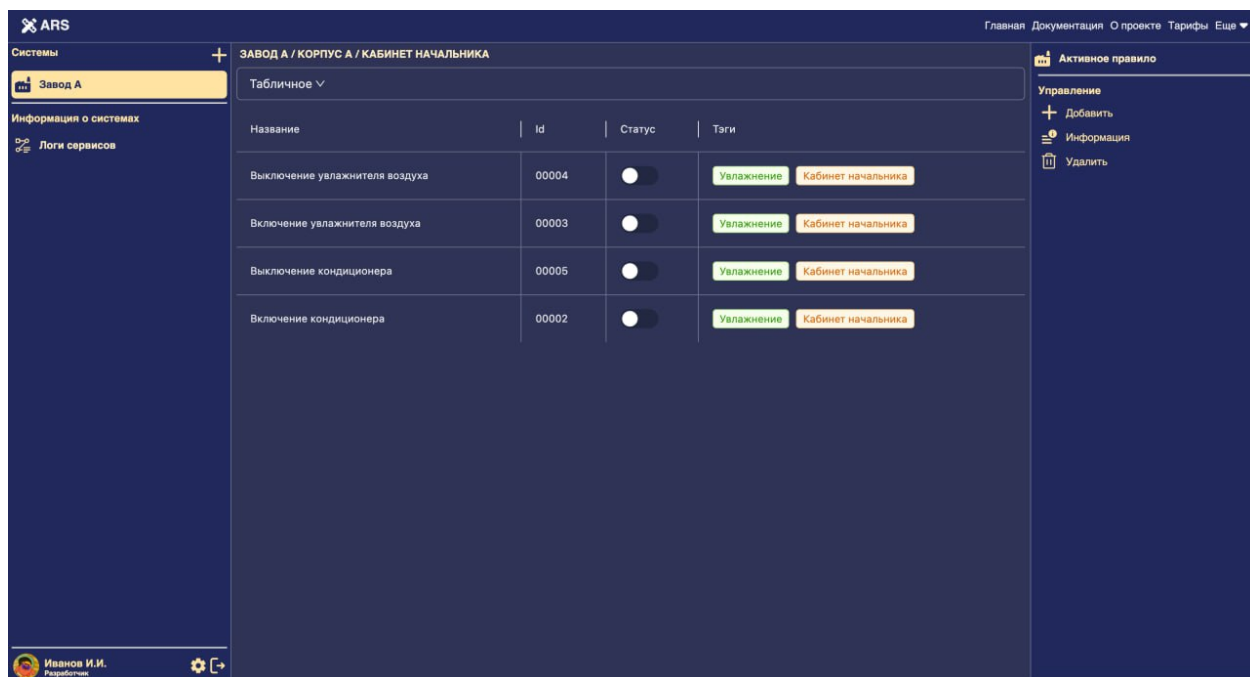


Рисунок Д.5 – Результат теста переключения активных правил с тэгового на табличное представление

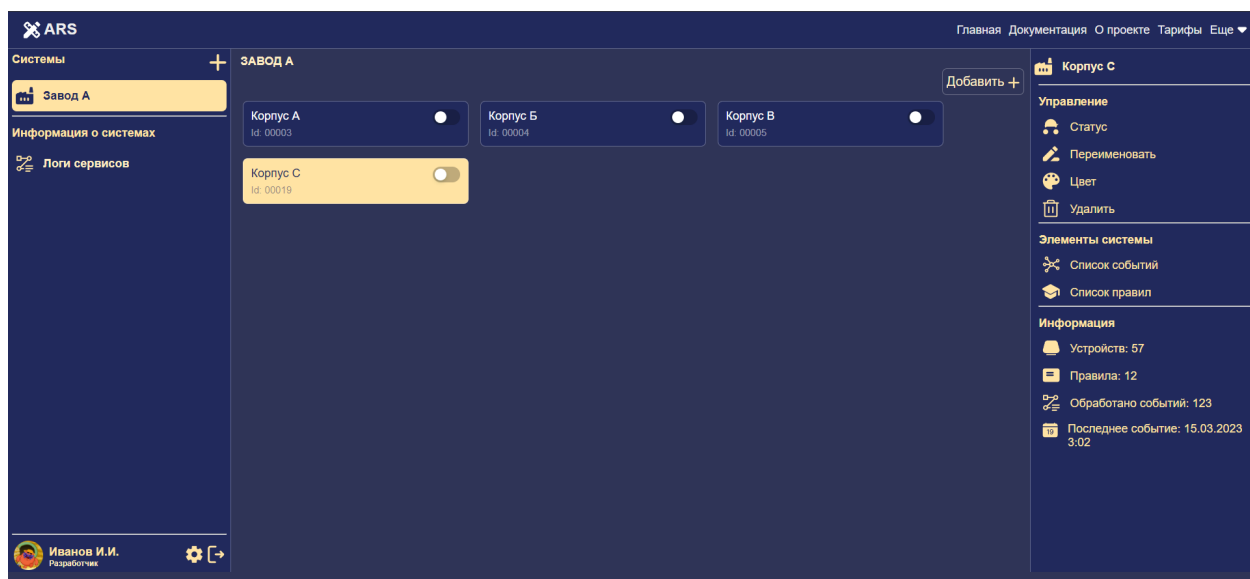


Рисунок Д.6 – Карточки до удаления подсистемы “Корпус С”.

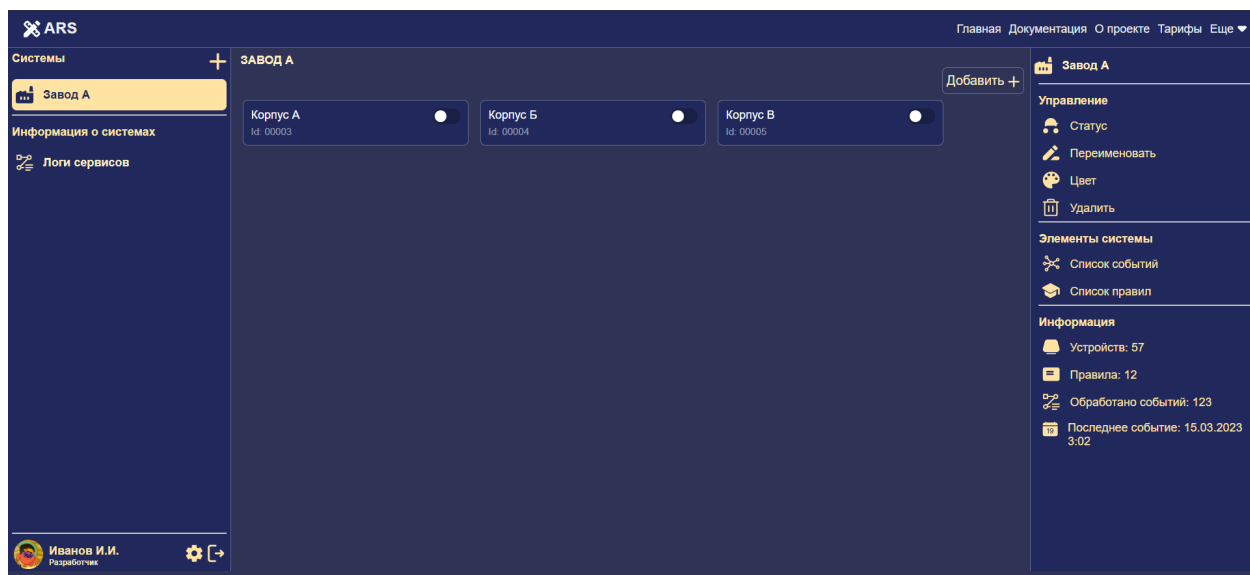


Рисунок Д.7 – Результат теста удаления подсистемы “Корпус С”.

Приложение Е
РЕЗУЛЬТАТЫ КОНФИГУРАЦИОННОГО ТЕСТИРОВАНИЯ
(рекомендуемое)

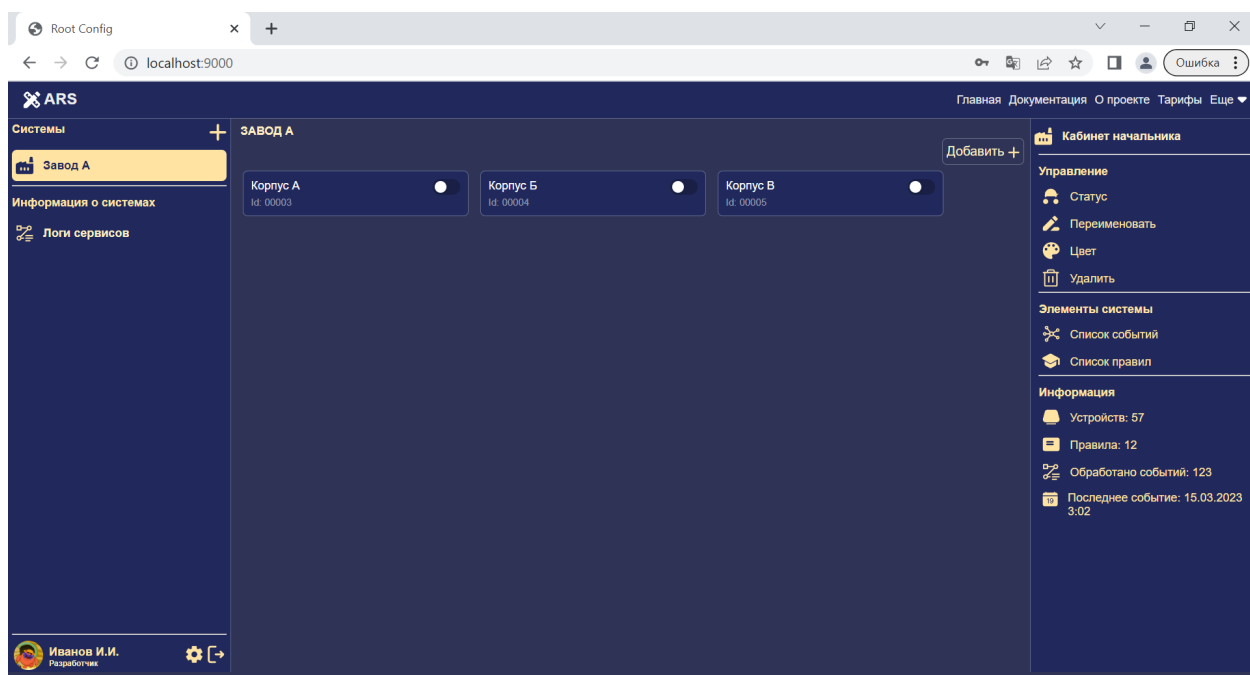


Рисунок Е.1 – Отображение систем и подсистем в браузере Google Chrome

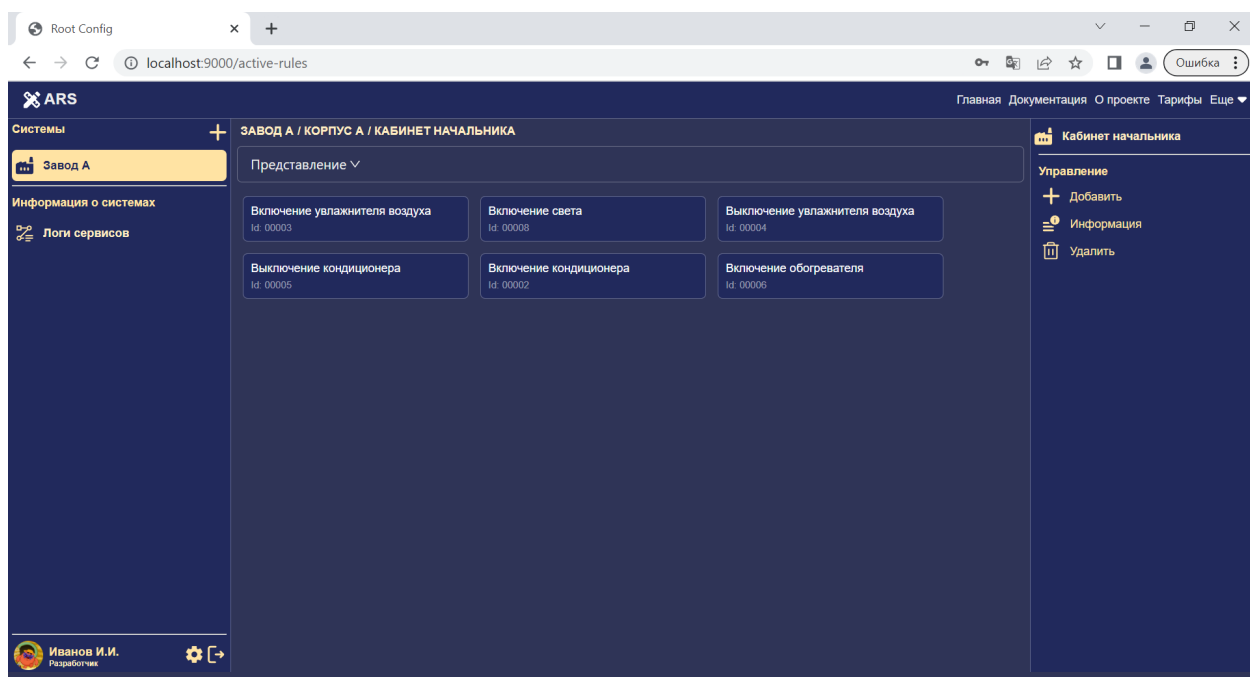


Рисунок Е.2 – Отображение активных правил системы в браузере Google Chrome

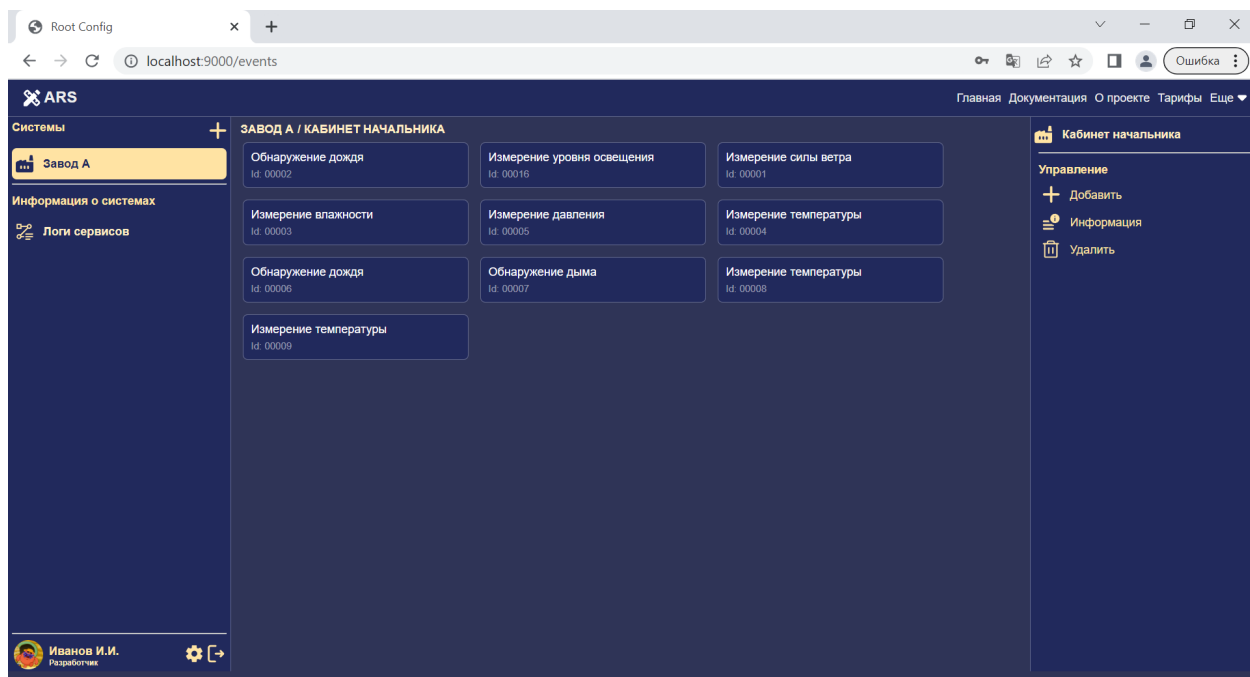


Рисунок Е.3 – Отображение событий систем в браузере Google Chrome

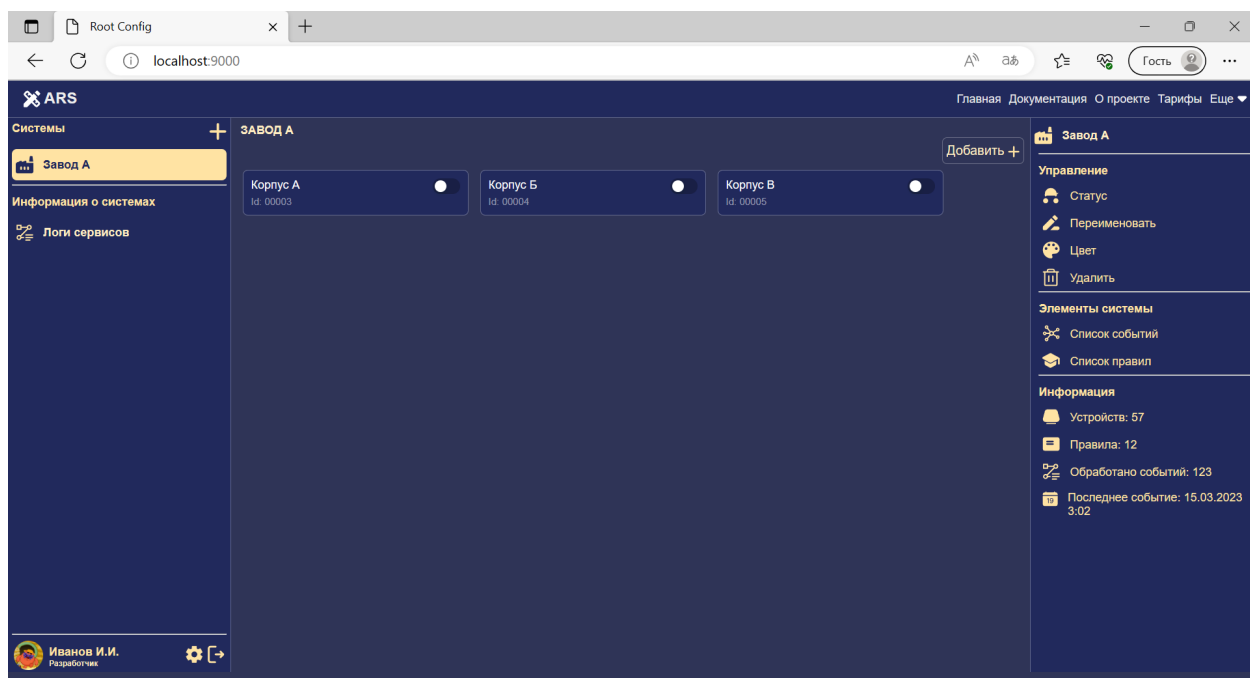


Рисунок Е.4 – Отображение систем и подсистем в браузере Microsoft Edge

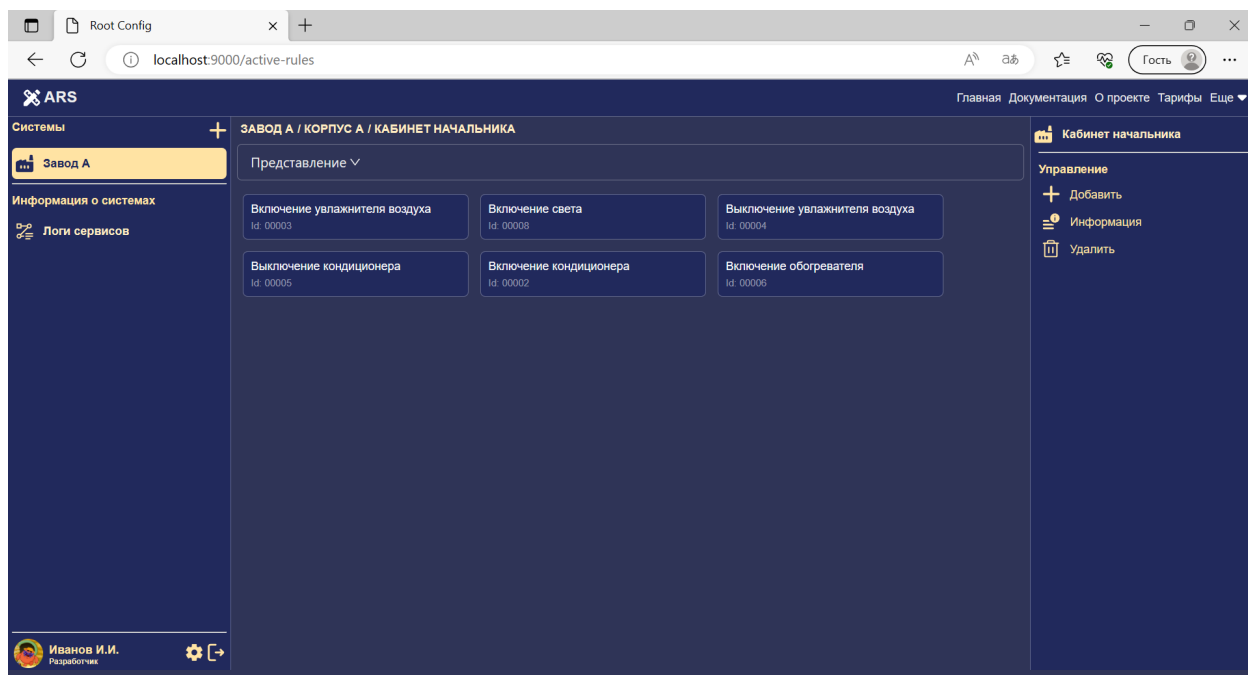


Рисунок Е.5 – Отображение активных правил системы в браузере Microsoft Edge

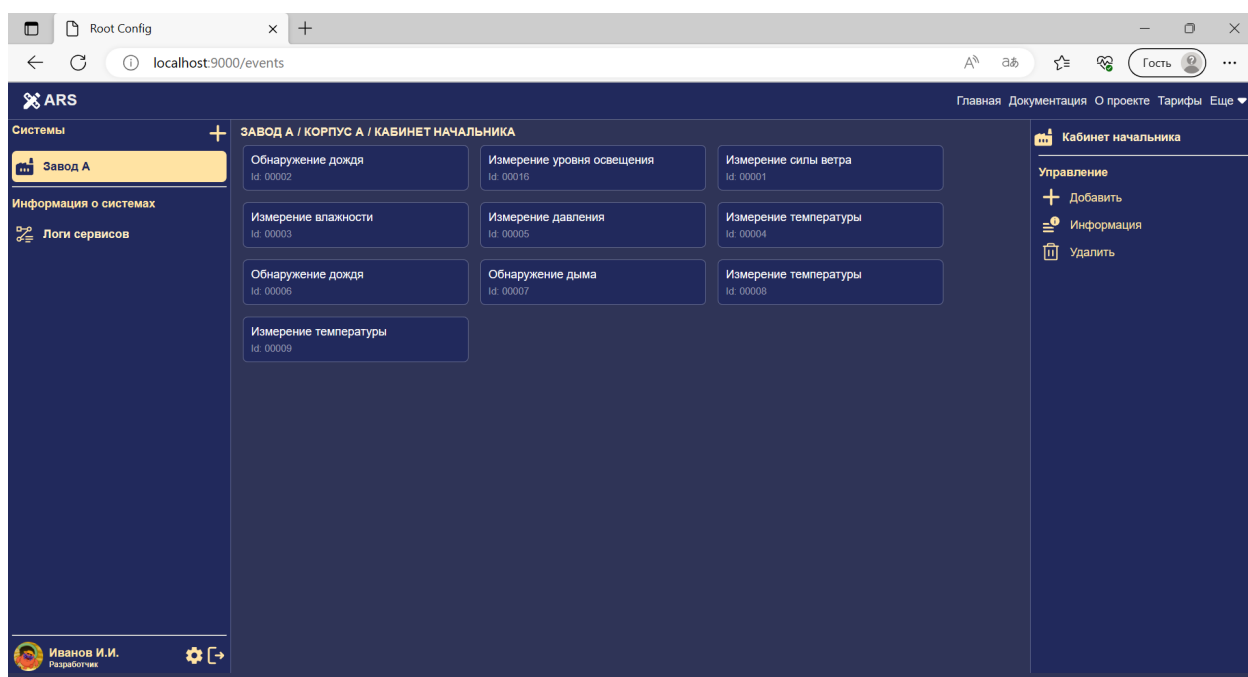


Рисунок Е.6 – Отображение событий систем в браузере Microsoft Edge

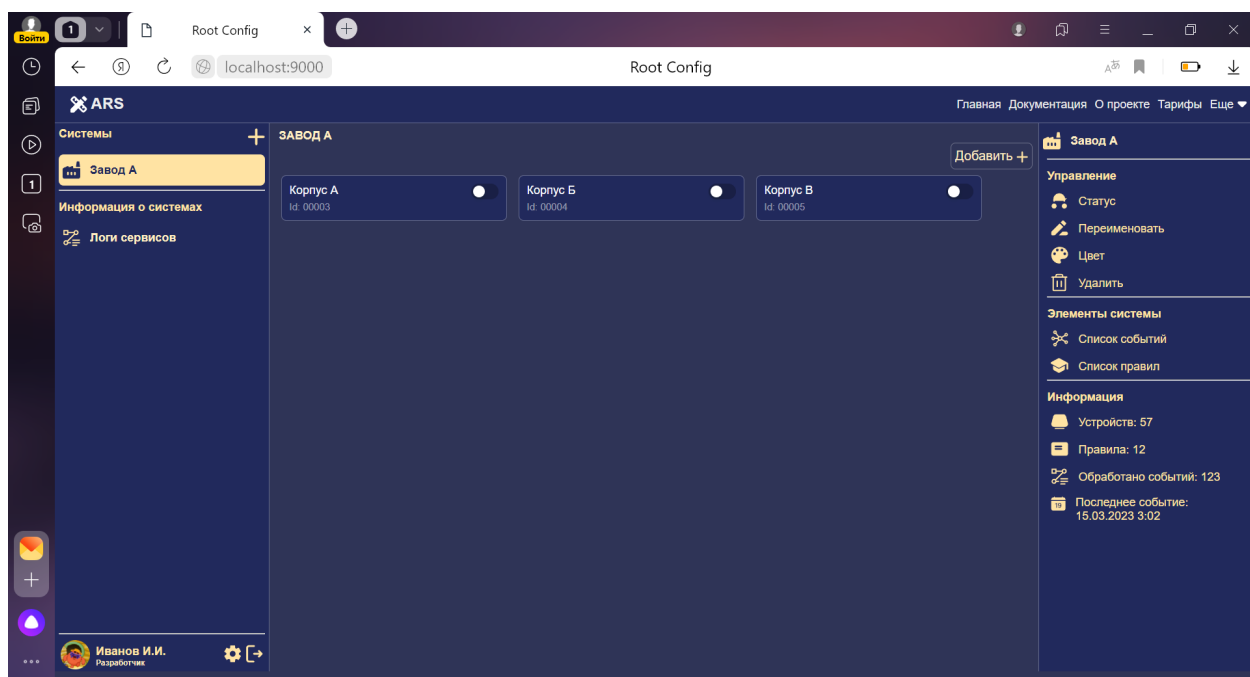


Рисунок Е.7 – Отображение систем и подсистем в браузере Яндекс.Браузер

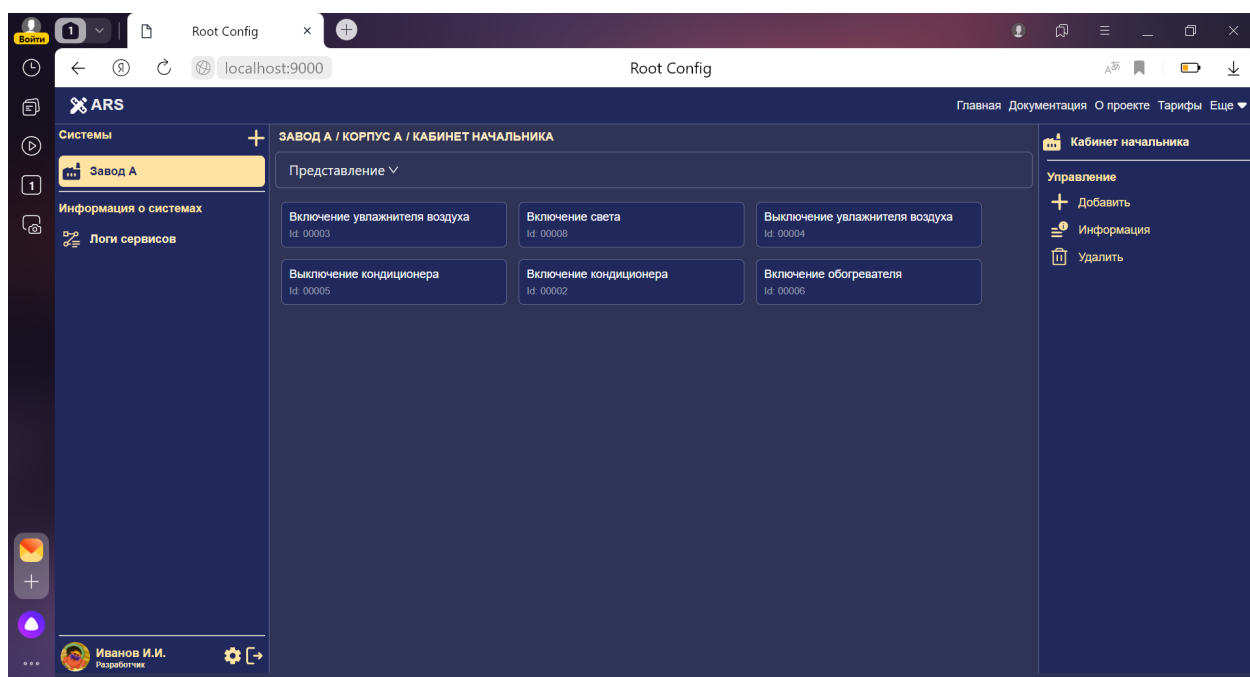


Рисунок Е.8 – Отображение активных правил системы в браузере
Яндекс.Браузер

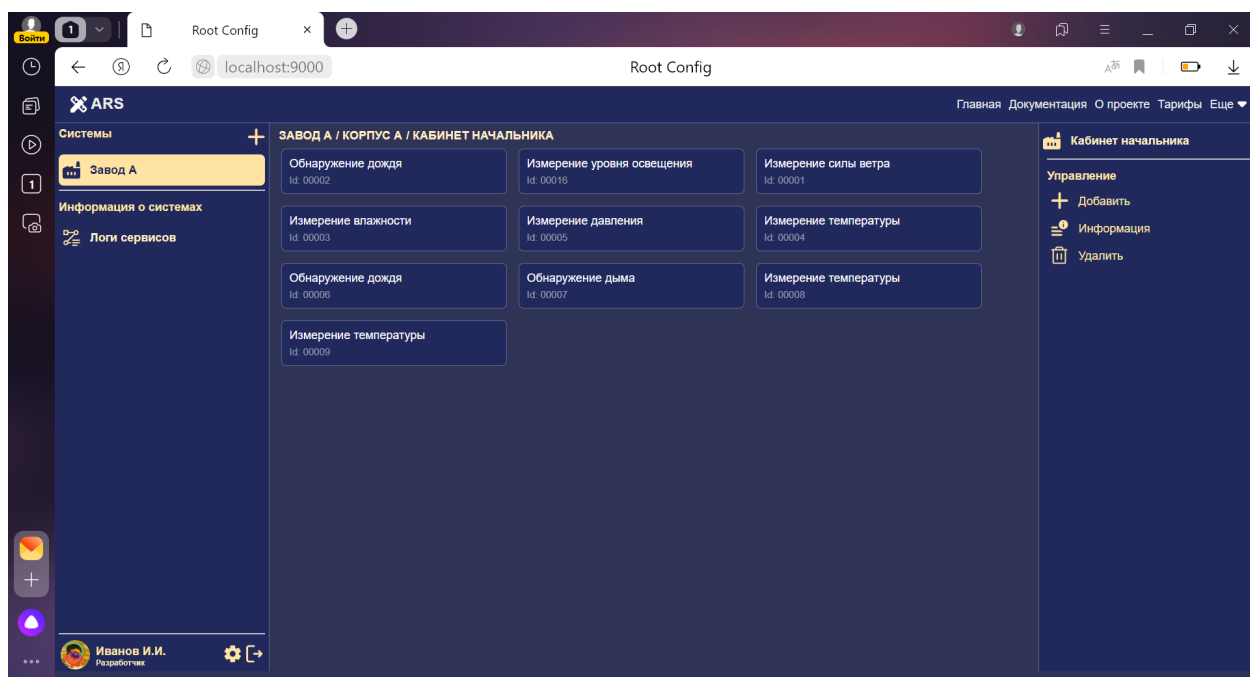


Рисунок Е.9 – Отображение событий систем в браузере Яндекс.Браузер

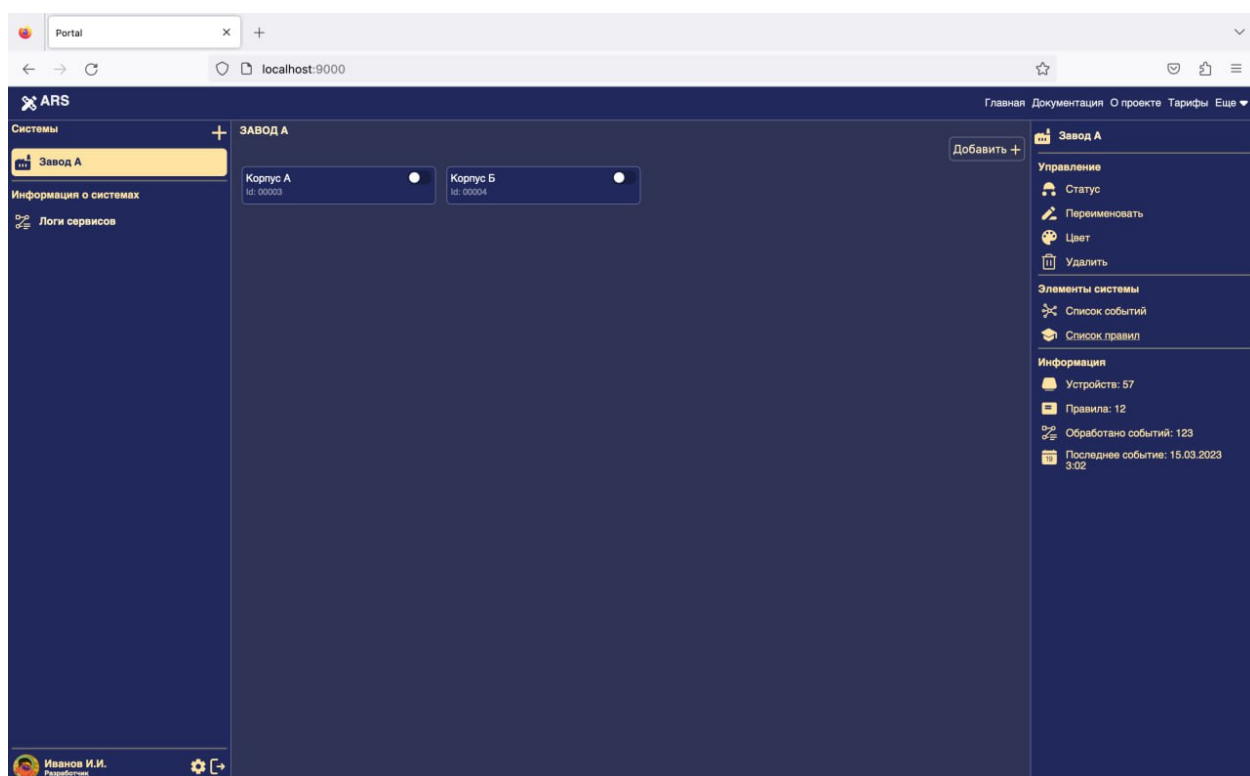


Рисунок Е.10 – Отображение систем и подсистем в браузере Firefox Browser

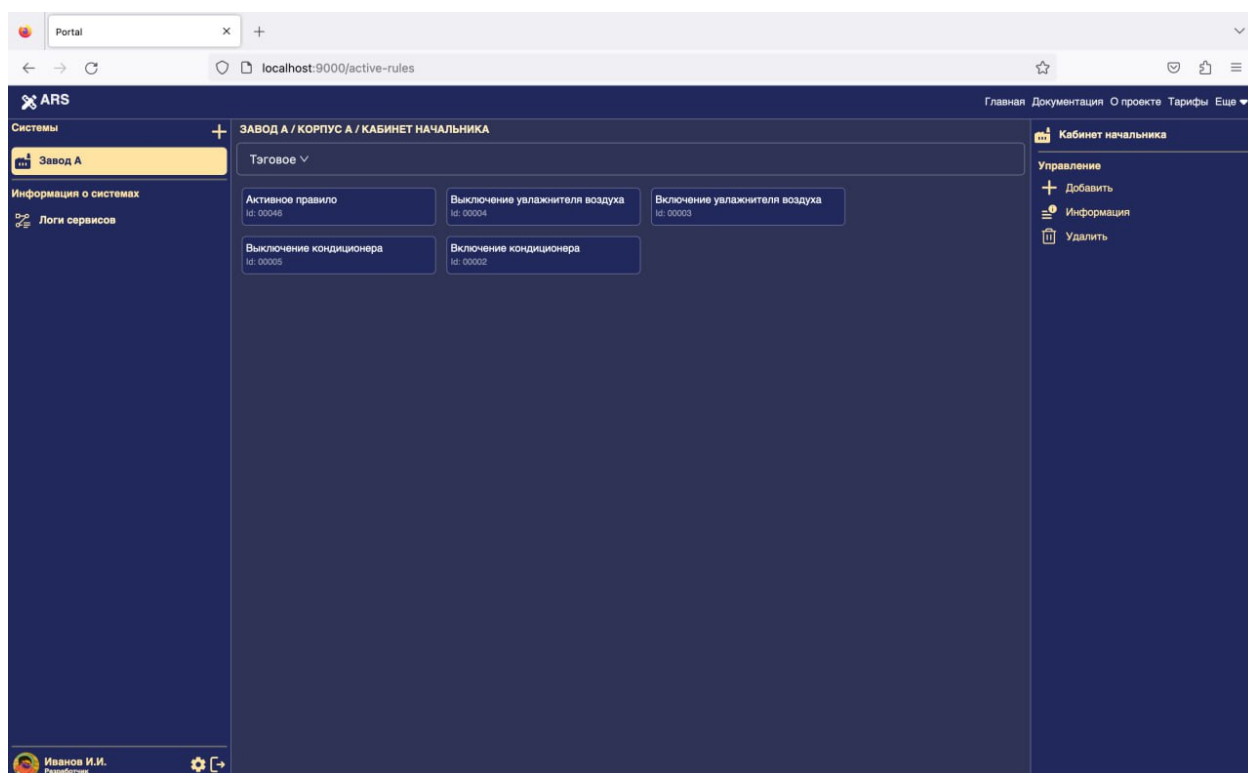


Рисунок Е.11 – Отображение активных правил системы в браузере Firefox Browser

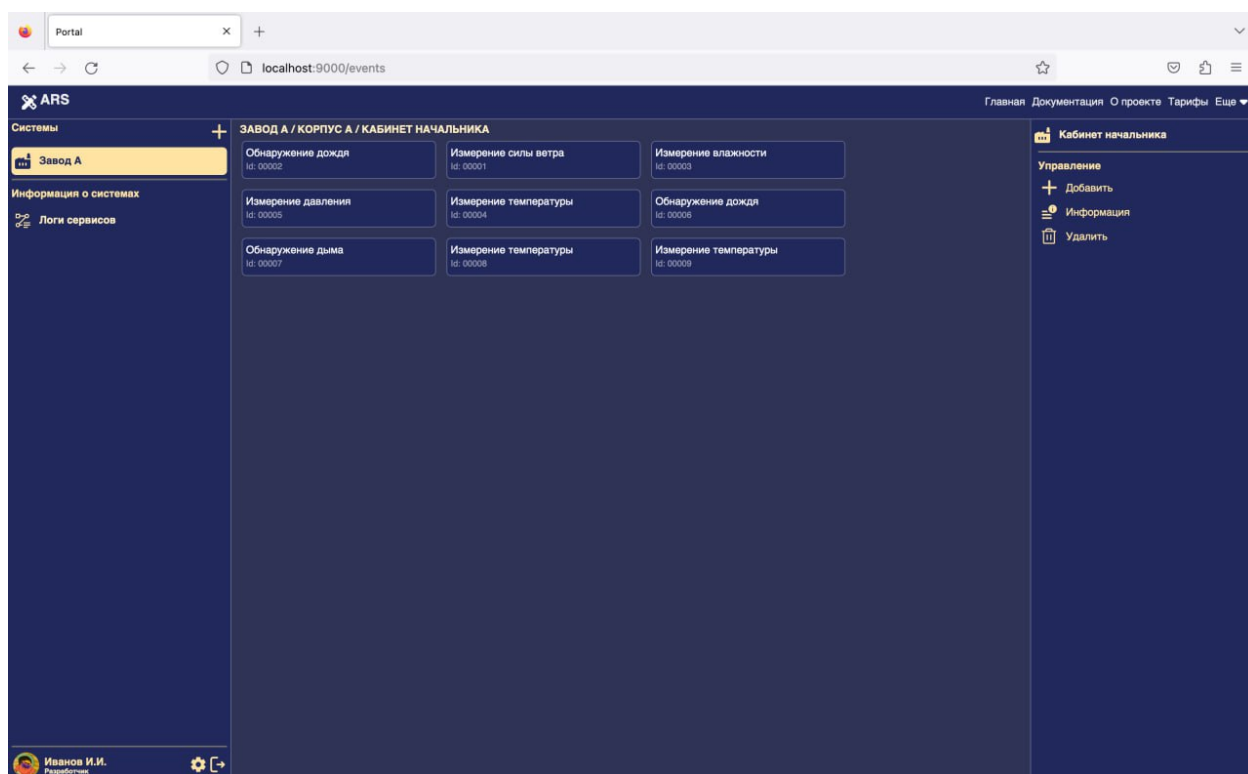


Рисунок Е.12 – Отображение событий систем в браузере Firefox Browser

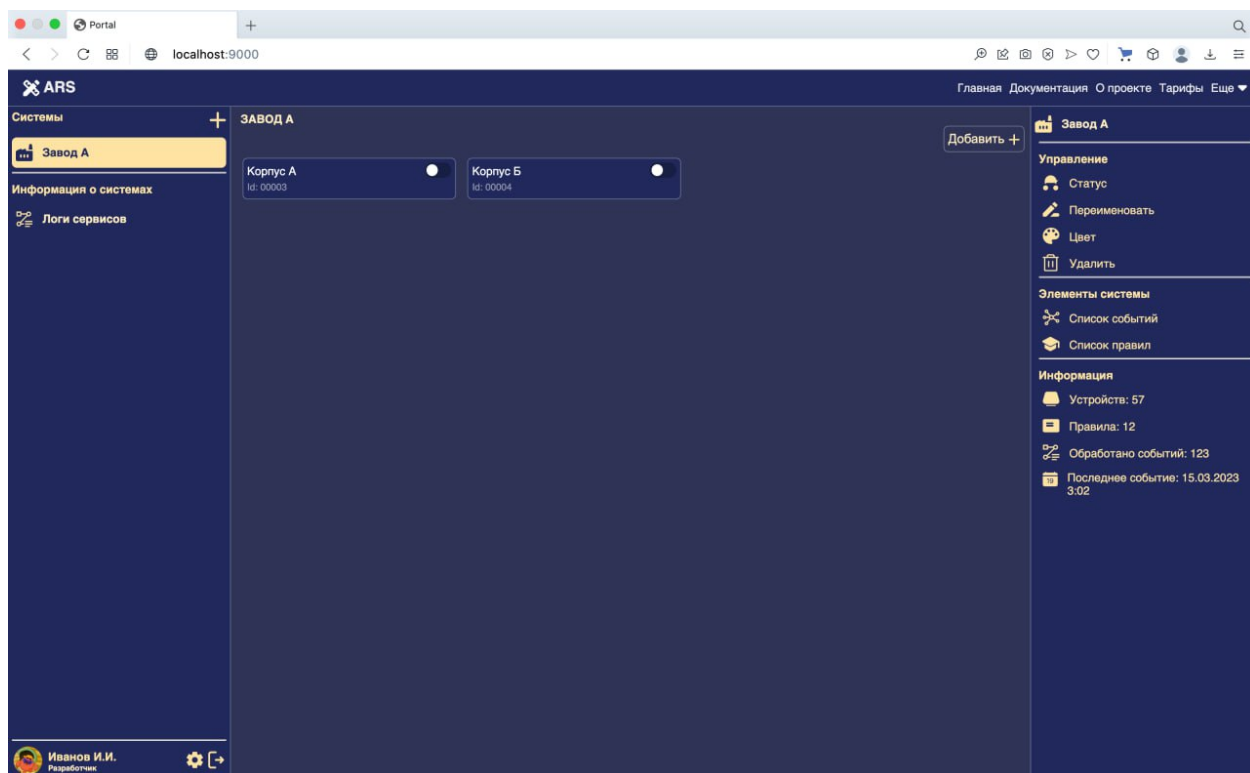


Рисунок Е.13 – Отображение систем и подсистем в браузере Орега

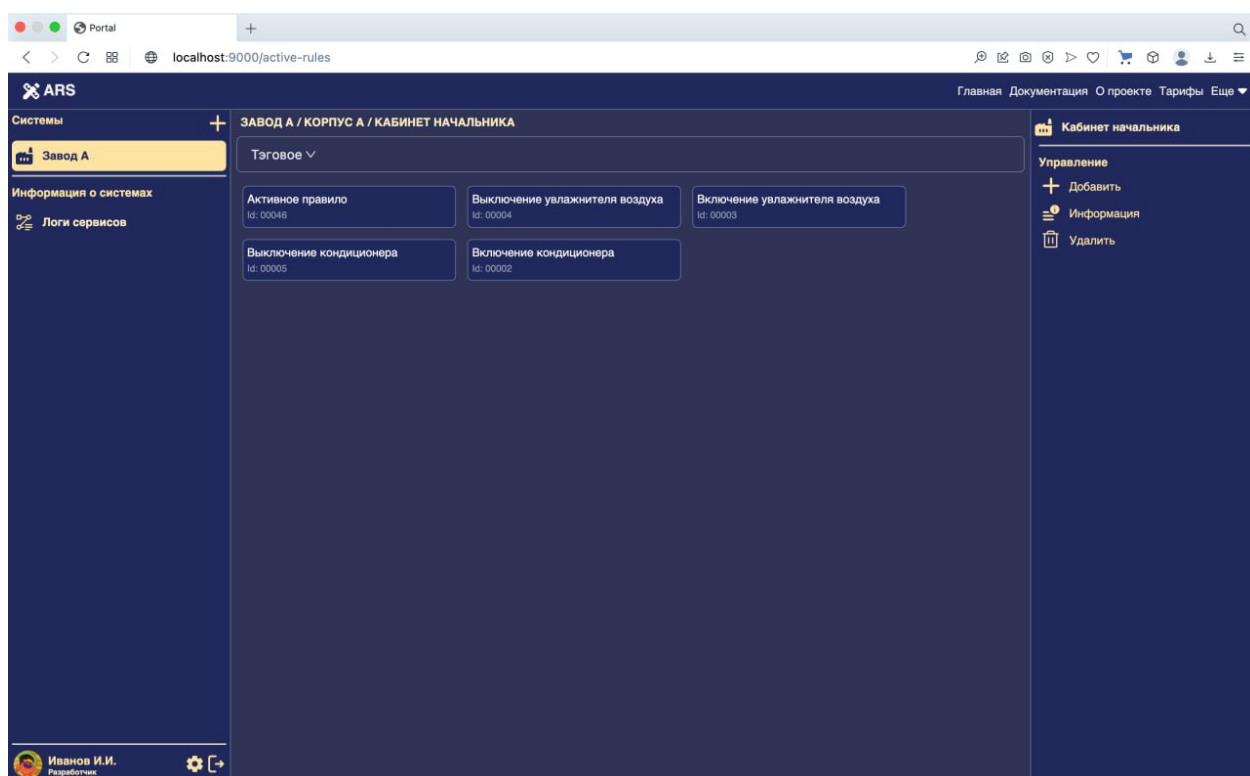


Рисунок Е.14 – Отображение активных правил системы в браузере Опера

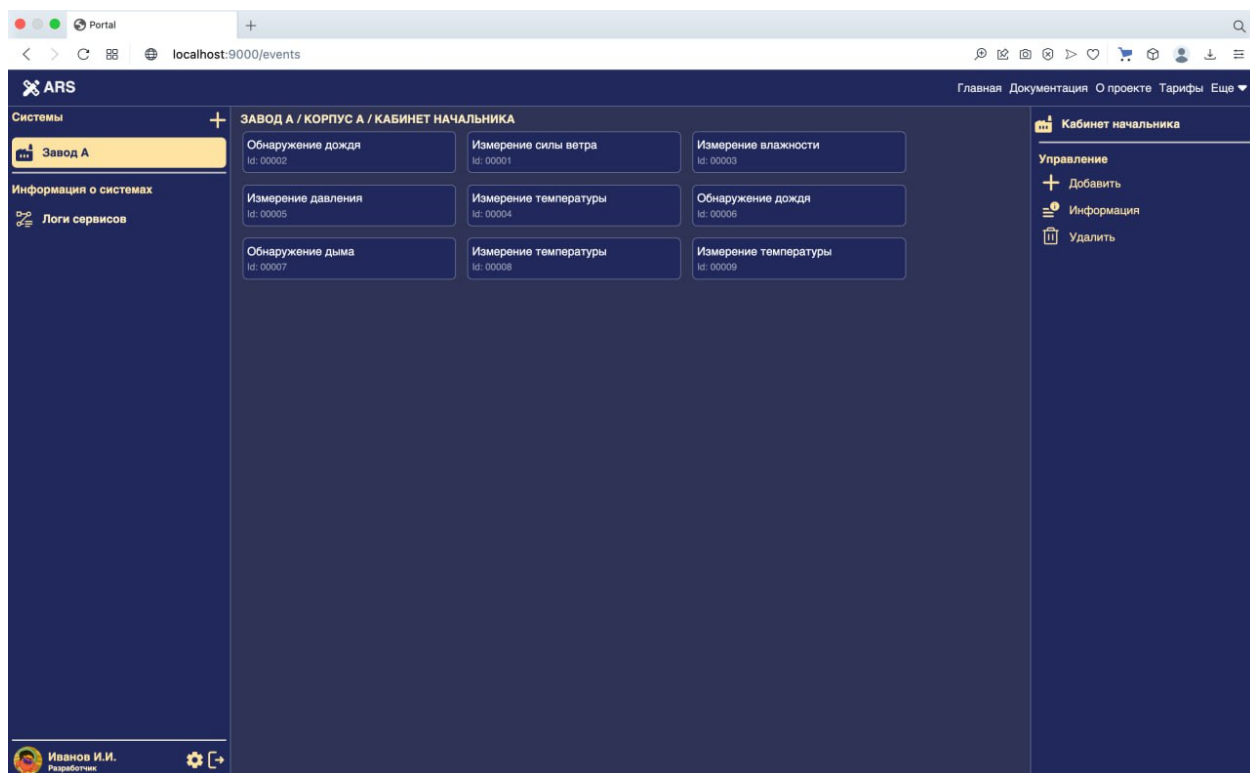


Рисунок Е.15 – Отображение событий систем в браузере Орега

Приложение Ж
РЕЗУЛЬТАТЫ МОДУЛЬНОГО ТЕСТИРОВАНИЯ
(рекомендуемое)

```

[chrome 114.0.5735.106 mac os x #0-5] Running: chrome (v114.0.5735.106) on mac os x
[chrome 114.0.5735.106 mac os x #0-5] Session ID: 6dc497b6eb2dc277f3b7a029810cc30a
[chrome 114.0.5735.106 mac os x #0-5]
[chrome 114.0.5735.106 mac os x #0-5] » /tests/e2e/systems.e2e.js
[chrome 114.0.5735.106 mac os x #0-5] System page test
[chrome 114.0.5735.106 mac os x #0-5]      ✓ load systems
[chrome 114.0.5735.106 mac os x #0-5]      ✓ get subsystems
[chrome 114.0.5735.106 mac os x #0-5]
[chrome 114.0.5735.106 mac os x #0-5] 2 passing (9.5s)

```

Рисунок Ж.1 – Результат модульного тестирования раздела систем и подсистем

```

-----
[chrome 114.0.5735.106 mac os x #0-2] Running: chrome (v114.0.5735.106) on mac os x
[chrome 114.0.5735.106 mac os x #0-2] Session ID: b335c89965acb85095bc0237d16ed4df
[chrome 114.0.5735.106 mac os x #0-2]
[chrome 114.0.5735.106 mac os x #0-2] » /tests/e2e/events.e2e.js
[chrome 114.0.5735.106 mac os x #0-2] Events page test
[chrome 114.0.5735.106 mac os x #0-2]      ✓ load events
[chrome 114.0.5735.106 mac os x #0-2]      ✓ get subsystems
[chrome 114.0.5735.106 mac os x #0-2]      ✓ get events
[chrome 114.0.5735.106 mac os x #0-2]      ✓ add event
[chrome 114.0.5735.106 mac os x #0-2]      ✓ delete event
[chrome 114.0.5735.106 mac os x #0-2]
[chrome 114.0.5735.106 mac os x #0-2] 5 passing (22.4s)

```

Рисунок Ж.2 – Результат модульного тестирования раздела событий

```

-----
[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 Ж.3 – 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 Ж.4 – Результат модульного тестирования раздела редактора активных правил

Приложение И
ЛИСТИНГ МОДУЛЬНЫХ ТЕСТОВ
(рекомендуемое)

Листинг И.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 И.2 – Тесты на модуль аутентификации

```

const LoginPage = require('../pages/login.page');

describe('Login page test', () => {
    it('auth testing', async () => {
        await LoginPage.open();

        await LoginPage.login('admin@admin.com', 'admin');
    });
});

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

class AuthPage extends Page {

    get inputEmail () {
        return $('#email');
    }

    get inputPassword () {
        return $('#password');
    }

    get btnLogin () {
        return $('#login-button');
    }

    async login (email, password) {
        await this.inputEmail.setValue(email);
        await this.inputPassword.setValue(password);
        await this.btnLogin.click();
    }

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

```

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

Листинг И.3 – 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 И.4 – Тесты на модуль событий

```

const SystemsPage = require('../pages/systems.page');
const LoginPage = require('../pages/login.page');
const EventsPage = require('../pages/events.page');

describe('Events page test', () => {
    it('load events', 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 events', async () => {
        await EventsPage.getEvent($('#system-element-00016'));
        await EventsPage.loadEvents();
    });

    it('add event', async () => {
        await EventsPage.addEvent('Новое событие', $('#react-
select-2-option-1'), $('#react-select-3-option-0'));
    });

    it('delete event', async () => {
        await EventsPage.deleteEvent('Новое событие');
    });

```

```

    });
  });

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

class EventsPage extends Page {
  get eventsContainer() {
    return $('#events-container');
  }

  get eventsLink() {
    return $('#events-link');
  }

  get addButton() {
    return $('#add-event');
  }

  get inputName() {
    return $('#events-input-name');
  }

  get selectContext() {
    return $('#events-select-context');
  }

  get selectComponent() {
    return $('#events-select-component');
  }

  get saveEventButton() {
    return $('#save-event-button');
  }

  get deleteEventButton() {
    return $('#delete-event-button');
  }

  async loadEvents() {
    try {
      await this.eventsContainer.waitForDisplayed({timeout:
2000});
    } catch (e) {
      throw new Error('Не удалось загрузить события');
    }
  }

  async getEvent(element) {
    await element.waitForDisplayed({timeout: 3000});
    await element.click();
    await this.eventsLink.click();
  }
}

```

```

    async addEvent(name, context, component) {
        await this.addButton.click();
        await this.inputName.setValue(name);

        await this.selectContext.click();
        await context.waitForDisplayed({timeout: 3000});
        await context.click();

        await this.selectComponent.click();
        await component.waitForDisplayed({timeout: 3000});
        await component.click();

        await this.saveEventButton.click();
    }

    async deleteEvent(event) {
        const header = await $('div=${event}');
        await $(header).click();
        await this.deleteEventButton.click();
        await browser.pause(10000);
    }

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

module.exports = new EventsPage();

```

Листинг И.5 – Тесты на модуль систем

```

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

describe('System page test', () => {
    it('load systems', 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'));
    });
});

```

```

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

class SystemsPage extends Page {
  get systemsContainer () {
    return $('#systems-container');
  }

  get systemElement() {
    return $('#system-element-00003');
  }

  async loadSystems () {
    try {
      await this.systemsContainer.waitForDisplayed({timeout:
2000});
    } catch (e) {
      throw new Error('Не удалось загрузить системы');
    }
  }

  async getSubsystem (element) {
    await element.doubleClick();
    await this.systemsContainer.waitForDisplayed({timeout:
2000});
  }

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

module.exports = new SystemsPage();

```